CHƯƠNG 1: GIỚI THIỆU

1.1. BỐI CẢNH VÀ ĐỘNG LỰC NGHIÊN CỨU

1.1.1. Thách thức giáo dục trực tuyến và nhu cầu cá nhân hóa ở quy mô lớn

Trong hơn một thập kỷ qua, giáo dục trực tuyến đã trải qua giai đoạn tăng trưởng bùng nổ, cả về quy mô người học lẫn mức độ đa dạng của hình thức đào tạo. Các báo cáo thị trường gần đây cho thấy quy mô thị trường e-learning toàn cầu đã đạt tới hàng chục tỷ USD và được dự báo tiếp tục tăng trưởng mạnh trong giai đoạn 2025–2033, với tốc độ tăng trưởng kép hàng năm ở mức hai chữ số. Cùng với đó, các nền tảng MOOC như Coursera, Udemy, edX, Khan Academy ghi nhận hàng trăm triệu lượt người học đăng ký và tham gia khóa học mỗi năm, với số lượng video, bài kiểm tra, chứng chỉ hoàn thành tăng liên tục theo từng kỳ báo cáo.

Sự tăng trưởng này phản ánh một sự chuyển dịch sâu sắc trong cách con người tiếp cận tri thức: từ mô hình lớp học truyền thống, phụ thuộc nặng nề vào không gian vật lý và thời gian cố định, sang mô hình học tập linh hoạt, có thể diễn ra ở bất cứ đâu và bất cứ lúc nào. Tuy nhiên, đi kèm với sự mở rộng quy mô là bài toán "cá nhân hóa ở quy mô lớn". Khi số lượng người học tăng lên hàng trăm triệu, mô hình một giáo viên kèm cặp một nhóm nhỏ học viên nhanh chóng trở nên không khả thi về mặt nguồn lực.

Báo cáo của UNESCO những năm gần đây chỉ ra rằng phần lớn học viên trên các nền tảng học trực tuyến mong muốn có các lộ trình học tập được điều chỉnh phù hợp với trình độ, mục tiêu và bối cảnh cá nhân. Đồng thời, UNESCO cũng ước tính thế giới đang thiếu hàng chục triệu giáo viên nếu muốn đạt được các mục tiêu phổ cập giáo dục cơ bản trong giai đoạn 2030, cho thấy khoảng cách rất lớn giữa nhu cầu học tập cá nhân hóa và năng lực đáp ứng của hệ thống giáo dục truyền thống. Trong bối cảnh đó, việc ứng dụng các công nghệ mới – đặc biệt là Đồ thị Tri thức (Knowledge Graph – KG), Mô hình Ngôn ngữ Lớn (Large Language Model – LLM) và kiến trúc AI Đa tác nhân (Multi-Agent AI) – trở thành một hướng đi tiềm năng để thu hẹp khoảng cách giữa quy mô và mức độ cá nhân hóa.

1.1.2. Hạn chế của các hệ thống hiện tại: từ LMS, adaptive engine đến rule-based recommender

Mặc dù các nền tảng LMS và adaptive learning hiện nay đã tích hợp một số cơ chế cá nhân hóa, phần lớn vẫn bị giới hạn bởi ba nhóm vấn đề chính.

Thứ nhất, cấu trúc tri thức nền tảng còn đơn giản. Nhiều hệ thống chỉ sử dụng cấu trúc danh mục hoặc phân cấp tuyến tính, trong đó mỗi khóa học được gắn với một vài thẻ hoặc thuộc tính cơ bản. Cách biểu diễn này không phản ánh đầy đủ các mối quan hệ tiên quyết, phụ thuộc, tương

đồng ngữ nghĩa hay đường đi thay thế giữa các khái niệm. Hệ quả là các lộ trình học tập được đề xuất mang tính "gợi ý nội dung" nhiều hơn là "thiết kế lộ trình tri thức".

Thứ hai, khả năng tương tác tự nhiên với người học còn hạn chế. Phần lớn hệ thống dựa trên các biểu mẫu hoặc giao diện tĩnh, buộc người học phải chọn lựa từ các danh sách cố định thay vì diễn đạt nhu cầu bằng ngôn ngữ tự nhiên. Khả năng "hiểu" bối cảnh học tập thực sự của người học – ví dụ như "đã biết SQL cơ bản, cần học nhanh để phục vụ công việc phân tích dữ liệu trong 1 tháng" – còn rất hạn chế.

Thứ ba, cơ chế thích ứng động trong quá trình học thường không đủ sâu. Khi người học gặp khó khăn ở một khái niệm, nhiều hệ thống chỉ cung cấp thêm bài tập hoặc nội dung bổ sung tại chỗ, thay vì có khả năng tái thiết kế lại toàn bộ lộ trình (ví dụ: quay lại các khái niệm nền tảng, đề xuất đường đi thay thế, hoặc điều chỉnh độ sâu nội dung). Hệ thống thường thiếu các cơ chế *remediation* có cấu trúc, thiếu cả khả năng cung cấp lộ trình thay thế dựa trên tri thức nền.

1.1.3. Cơ hội từ KG + LLM + Multi-Agent AI

Có ba xu hướng công nghệ trọng yếu trong giai đoạn 2023–2025 mở ra cơ hội giải quyết các hạn chế vừa nêu.

Thứ nhất, Đồ thị Tri thức (Knowledge Graph – KG) cho phép biểu diễn tri thức dưới dạng các nút và cạnh, nắm bắt phong phú các loại quan hệ như tiên quyết (*REQUIRES, IS_PREREQUISITE_OF*), trình tự (*NEXT*), khắc phục (*REMEDIATES*), đường đi thay thế (*HAS_ALTERNATIVE_PATH*), tương đồng ngữ nghĩa (*SIMILAR_TO*), hay cấu trúc khái niệm (*IS_SUBCONCEPT_OF*). Với nền tảng như Neo4j, hệ thống có thể truy vấn và phân tích con đường học tập logic, nhiều bước dựa trên cấu trúc tri thức rõ ràng.

Thứ hai, Mô hình Ngôn ngữ Lớn (LLM) như GPT-5.1, Claude Opus, Gemini Pro cho phép hiểu và sinh ngôn ngữ tự nhiên ở mức độ tinh vi, giúp hệ thống "nghe – hiểu – phản hồi" theo cách gần với tương tác giữa người với người. Khi kết hợp với framework như LlamaIndex, LLM có thể truy cập KG thông qua Property Graph Index, đồng thời kết hợp với kho vector để thực hiện truy vấn ngữ nghĩa, từ đó vừa tận dụng được cấu trúc tri thức vừa khai thác được thông tin từ tài liệu giáo trình sẵn có.

Thứ ba, kiến trúc AI Đa tác nhân (Multi-Agent AI) đang nổi lên như một khuôn mẫu chủ đạo cho các hệ thống AI phức tạp. Thay vì một "siêu agent" làm mọi thứ, hệ thống được chia thành nhiều agent chuyên trách: phân tích người học, lập kế hoạch lộ trình, hướng dẫn học, đánh giá tiến bộ, tạo ghi chú tri thức, v.v. Các agent phối hợp qua cơ chế chuyển giao nhiệm vụ, chia sẻ trạng thái chung và kết nối sự kiện, giúp hệ thống linh hoạt, có khả năng lập luận kế hoạch theo từng vai trò chuyên sâu.

Trong luận văn này, sự kết hợp KG, LLM và Multi-Agent AI được khai thác để kiến tạo một hệ thống đề xuất lộ trình học tập cá nhân hóa không chỉ "gợi ý khóa học", mà còn có thể "thiết kế lộ trình tri thức" theo đúng bối cảnh, trình độ và mục tiêu của từng cá nhân.

1.1.4. Ba nghiên cứu cốt lõi năm 2025: Harvard, Dartmouth, LearnLM

Năm 2025 đánh dấu ba dòng nghiên cứu có tính bước ngoặt, cung cấp nền tảng khoa học cho việc thiết kế hệ thống trong luận văn này.

Thứ nhất, nghiên cứu của Harvard do Kestin và cộng sự công bố trên Scientific Reports (Nature Publishing Group) chứng minh rằng "AI tutoring", khi được thiết kế dựa trên các nguyên tắc sư phạm vững chắc, có thể vượt trội so với phương pháp "active learning" truyền thống về hiệu quả học tập và hiệu suất học theo thời gian. Nghiên cứu sử dụng RCT trên 194 sinh viên, cho thấy nhóm sử dụng "AI tutor" đạt hiệu quả học tập trung vị cao hơn đáng kể, với độ ảnh hưởng từ khoảng 0.73 đến trên 1 độ lệch chuẩn – một mức ảnh hưởng được xem là rất lớn trong nghiên cứu giáo dục. Điểm cốt lõi là AI tutor không chỉ "thông minh" về mô hình, mà được thiết kế xoay quanh 7 nguyên tắc sư phạm, bao gồm: khuyến khích học chủ động, quản lý tải nhận thức, đi từng bước nhỏ, khuyến khích tự suy nghĩ, nuôi dưỡng tư duy phát triển, phản hồi cá nhân hóa, và kiểm soát sự mơ hồ.

Thứ hai, nghiên cứu của Thesen & Park tại Dartmouth về "AI Can Deliver Personalized Learning at Scale" giới thiệu khái niệm Precision Education và nhấn mạnh vai trò của kiến trúc "Retrieval-Augmented Generation (RAG)" trong việc nâng cao độ tin cậy và tính chính xác của trợ lý giảng dạy AI. Bằng cách lưu trữ tài liệu khóa học trong cơ sở dữ liệu vector và buộc mô hình phải truy xuất thông tin liên quan trước khi tạo sinh nội dung trả lời, Dartmouth chứng minh rằng mức độ sử dụng, niềm tin của người học và hiệu quả học tập đều được cải thiện rõ rệt. Đặc biệt, trong giai đoạn cận kỳ thi, hệ thống ghi nhận lượng truy cập tăng hơn 300% so với mô hình AI không có cơ chế truy xuất thông tin này.

Thứ ba, LearnLM – một dòng mô hình được Google DeepMind giới thiệu trong hệ sinh thái Gemini – tập trung tích hợp nguyên lý khoa học học tập vào trong thiết kế mô hình và trải nghiệm tương tác. LearnLM nhấn mạnh các nguyên tắc như quản lý tải nhận thức, kích hoạt sự tò mò, thích ứng với người học, khuyến khích năng lực tự phản tư duy và duy trì mối liên hệ giữa hoạt động học với mục tiêu dài hạn. Đây là cầu nối trực tiếp giữa lý thuyết khoa học giáo dục và cách thiết kế lời nhắc, workflow, và cơ chế tương tác của các agent trong hệ thống.

Luận văn lấy ba nghiên cứu này làm trụ cột: Harvard cung cấp "bộ nguyên tắc sư phạm có kiểm chứng thực nghiệm", Dartmouth cung cấp bằng chứng "kiến trúc RAG cho Precision Education và khái niệm grounded trust", LearnLM cung cấp "khung khoa học học tập" để thiết kế.

1.1.5. Động lực nghiên cứu: từ lộ trình nội dung sang lộ trình tri thức cá nhân hóa

Xuất phát từ bối cảnh trên, động lực nghiên cứu không chỉ là "nâng cấp một hệ thống gợi ý lộ trình học tập", mà là chuyển dịch từ:

- Hệ thống đề xuất dựa trên KG tĩnh, ít tương tác, sang
- Hệ thống đa tác nhân có khả năng:
    - Tự động xây dựng KG từ tài liệu,
    - Duy trì trạng thái học tập của từng người học trong thời gian dài,
    - Đề xuất lộ trình tri thức thích ứng,
    - Tạo ra "artifacts tri thức" giúp người học xây dựng Personal Knowledge Graph và "bộ não thứ hai" của riêng mình.

Luận văn hướng tới việc mô hình hóa không chỉ "hành trình đi qua các node tri thức trong KG", mà còn cả "hành trình xây dựng tri thức cá nhân", thông qua Personal KG, KAG Agent và kiến trúc Dual-KG (Course KG và Personal KG) đảm bảo quá trình tạo sinh đều dựa trên truy xuất tri thức có kiểm chứng, đồng thời tuân theo cácàn tắc sư phạm đã được thực nghiệm và xác định hiệu quả.

1.3. ĐỐI TƯỢNG NGHIÊN CỨU

Nghiên cứu tập trung vào sáu tác nhân AI (agents) và ba thành phần kiến trúc (Personal KG, KAG Agent, Dual-KG) như là đối tượng trung tâm.

1.3.1. Knowledge Extraction

Knowledge Extraction Agent là tác nhân ngoại tuyến (offline), chịu trách nhiệm:

- Đọc tài liệu giáo trình (lecture notes, slides, textbook) và trích xuất các khái niệm (concepts) cùng quan hệ (relationships) theo schema KG đã định nghĩa.
- Thực hiện các bước: entity/relation extraction, schema alignment, chuẩn hóa khái niệm, loại trùng (node merging theo ngữ nghĩa), và sinh ra file dữ liệu để nạp vào Neo4j.
- Tự động hóa quy trình SPR Generator và SPR Validation, thay thế việc xây dựng KG thủ công.

Các pipeline và logic của agent này: prompt, flow, tiêu chí hợp nhất node, và đảm bảo tính tương thích với các agent runtime là các tiêu chí nghiên cứu quan trọng.

1.3.2. Nhóm tác nhân runtime phục vụ học tập

Bốn tác nhân runtime hoạt động khi người học tương tác với hệ thống:

1. **Learner Profiler Agent**
    a. Phân tích thông tin đầu vào: mục tiêu học tập, bối cảnh, thời gian, phong cách học, kết quả pre-test.
    b. Cập nhật hồ sơ người học trong Central State và Personal KG: mức độ thành thạo, concept đã hoàn thành, concept còn yếu.
2. **Path Planner Agent**
    a. Sử dụng MOPO-like planning (model-based, policy-guided) thay cho thuật toán A* cứng nhắc, để lập lộ trình học tập cá nhân hóa dựa trên Course KG và Personal KG.
    b. Tôn trọng các ràng buộc tiên quyết, tận dụng remediation và alternative paths, tối ưu theo nhiều tiêu chí (thời gian, độ khó, độ phù hợp với mục tiêu).
3. **Tutor Agent (phiên bản 2.0 – tích hợp Harvard, LearnLM)**
    a. Thực hiện vai trò AI tutor đối thoại với người học, tuân thủ 7 nguyên tắc Harvard và các guideline của LearnLM về quản lý tải nhận thức và kích hoạt tư duy.
    b. Kết hợp phương pháp Socratic/Reverse Socratic, đặt câu hỏi, gợi ý và scaffold thay vì chỉ đưa đáp án, đồng thời sử dụng tài liệu được RAG truy xuất để đảm bảo nội dung chính xác.image.jpg
4. **Evaluator Agent (phiên bản 2.0 – đánh giá thích ứng)**
    a. Thiết kế các câu hỏi đánh giá theo concept trong Course KG, phân loại lỗi sai theo “kiểu misconception” và cập nhật vào Personal KG.
    b. Đưa quyết định proceed/remediate/alternate, gửi tín hiệu cho Path Planner để điều chỉnh lộ trình, và kích hoạt KAG Agent tạo artifact khi cần.

1.3.3. Knowledge Artifact

KAG Agent là tác nhân đóng vai trò cầu nối giữa học tập và quản lý tri thức cá nhân (PKM):

- Nhận input là transcript các phiên tương tác giữa người học và Tutor/Evaluator (và, khi cần, tài liệu được RAG truy xuất).

- Tạo ra các "atomic notes" chuẩn hóa: mỗi note gắn với một concept, bao gồm định nghĩa, ví dụ, lỗi thường gặp, liên kết đến concept liên quan và meta-thông tin về bối cảnh học (session, ngày, mục tiêu).

- Ghi lại note vào Personal KG dưới dạng node/relationship mới, đồng thời gợi ý liên kết với các note cũ để xây dựng second brain theo tinh thần Zettelkasten và PKM hiện đại.

Như vậy, KAG Agent chuyển sản phẩm của quá trình học (đối thoại, giải thích, ví dụ) thành "artifacts" lâu dài, giúp người học không chỉ "vượt qua bài học hiện tại" mà còn "xây dựng tri thức bền vững".

1.3.4. Ba thành phần VER3 mới: Personal KG, KAG Agent, Dual-KG

Bên cạnh sáu agents, ba thành phần kiến trúc chính:

1. **Personal KG (PKG)**

    a. Một KG riêng cho từng người học, lưu trữ các concept đã học, mức độ thành thạo, lỗi sai, artifacts, liên kết đến Course KG.

    b. Được cập nhật liên tục bởi Profiler, Evaluator và KAG.

2. **KAG Agent**

    a. Đã mô tả ở trên, là tác nhân hiện thực hóa PKM và second brain.

3. **Dual-KG Architecture**

    a. Course KG + Personal KG cùng tồn tại, liên kết chặt chẽ: mỗi node trong Personal KG thường tham chiếu đến một node trong Course KG nhưng có thêm lớp annotation cá nhân.

    b. Path Planner, Tutor, Evaluator luôn hoạt động trên nền Dual-KG, thay vì chỉ Course KG, để đạt được mức độ cá nhân hóa sâu hơn.

1.4. PHẠM VI VÀ GIỚI HẠN NGHIÊN CỨU

1.4.1. Phạm vi

- **Đối tượng phục vụ:** Học viên thuộc lĩnh vực Hệ thống Thông tin Quản lý (MIS) ở bậc đại học/sau đại học, có nhu cầu học các chủ đề như SQL, Quản lý tri thức, Business Intelligence, trong bối cảnh học trực tuyến.

- **Công nghệ:** Sử dụng Neo4j Aura cho Course KG và Personal KG; LlamaIndex (Property Graph Index, VectorStoreIndex, AgentWorkflow) để kết nối LLM với KG; LLM thương mại (GPT-4o/Gemini/Claude) cho reasoning và sinh nội dung.

- **Chức năng:** Tập trung vào pipeline xây dựng KG, thiết kế và hiện thực hóa 6 agents, kiến trúc Dual-KG, 3-layer grounding và chạy thử nghiệm với một số kịch bản học tập tiêu biểu.

- **Lĩnh vực thử nghiệm:** Ba học phần/dạng nội dung trong MIS được chọn làm case study chính, với quy mô KG ở mức vừa phải để có thể hiện thực trong khuôn khổ một luận văn thạc sĩ.

1.4.2. Giới hạn

- **Giới hạn dữ liệu:** Chất lượng Course KG phụ thuộc vào tài liệu giáo trình hiện có; không bao quát tất cả các lĩnh vực hoặc dạng khóa học.

- **Giới hạn mô hình:** Hệ thống sử dụng LLM thương mại dưới dạng black box, không tự huấn luyện mô hình mới; việc kiểm soát sự mơ hồ dựa vào 3 layer xác thực với tài liệu gốc nhưng không cam kết loại bỏ hoàn toàn lỗi.

- **Giới hạn đánh giá:** Thực nghiệm chủ yếu là offline/simulated (hoặc quy mô nhỏ), chưa phải RCT với số lượng lớn sinh viên như Harvard hay Dartmouth.

- **Giới hạn phạm vi ứng dụng:** Hệ thống được thiết kế cho các môn có cấu trúc tri thức tương đối rõ, ít xử lý tri thức mơ hồ, cảm xúc, hoặc tình huống sư phạm phức tạp ngoài nội dung học thuật.

1.5. ĐÓNG GÓP KHOA HỌC

Nghiên cứu này đóng góp vào lĩnh vực giáo dục cá nhân hóa và AI trong giáo dục qua các điểm sau:

Đóng góp 1: First Integration of Harvard Pedagogical Principles into KG-based Learning Path System

Nghiên cứu Harvard (Kestin et al., 2025) chứng minh 7 nguyên tắc sư phạm hiệu quả trong AI tutoring với effect size 0.73-1.3 SD. Tuy nhiên, chưa có nghiên cứu nào tích hợp các nguyên tắc này vào hệ thống đề xuất lộ trình học tập dựa trên Knowledge Graph. Nghiên cứu này đóng góp framework tích hợp Harvard principles vào Tutor Agent prompts, grounded trong cấu trúc tri thức của KG.THESIS_VER1.docx

Đóng góp 2: RAG + Knowledge Graph Hybrid for Educational Trust

Nghiên cứu Dartmouth (Thesen & Park, 2025) chứng minh RAG tăng trust từ người học. Nghiên cứu này mở rộng concept bằng cách kết hợp RAG với Neo4j KG, tạo ra "Double Grounding" - grounded in both curated documents AND structured knowledge. Điều này giúp giảm hallucinations đồng thời đảm bảo tính logic của lộ trình học tập.THESIS_VER1.docx

Đóng góp 3: Multi-Agent System for Precision Education

Dartmouth đề xuất khái niệm Precision Education nhưng chưa triển khai cụ thể với multi-agent architecture. Nghiên cứu này implement precision education với 6 specialized agents (bao gồm KAG Agent [MỚI VER2]), mỗi agent chuyên môn hóa một nhiệm vụ trong quy trình học tập cá nhân hóa, phối hợp thông qua LlamaIndex AgentWorkflow.THESIS_VER1.docx

*Đóng góp 4: A vs Agentic Empirical Comparison**

Chưa có nghiên cứu so sánh thuật toán A* với LLM-based path planning trong education domain. Nghiên cứu này cung cấp empirical comparison với các metrics chuẩn (Learning Gain, Adaptability, Time Efficiency), đóng góp vào hiểu biết về ưu/nhược điểm của mỗi phương pháp.THESIS_VER1.docx

Đóng góp 5: Automated KG Construction Pipeline for Education

Quy trình xây dựng KG tự động từ tài liệu giáo dục sử dụng LLM-powered Knowledge Extraction Agent chưa được nghiên cứu rộng rãi trong literature. Nghiên cứu này giới thiệu pipeline: Document → Entity/Relation Extraction → Schema Alignment → Semantic Deduplication → Neo4j Loading, thay thế quy trình thủ công tốn thời gian.THESIS_VER1.docx

Đóng góp 6: Sử dụng tương đồng ngữ nghĩa Jaccard Similarity

Việc áp dụng tương đồng Jaccard xác định điểm bắt đầu và điểm đích dựa trên các thẻ ngữ nghĩa (semantic_tags), được mở rộng với hàm expand_tags(), giúp nâng cao độ chính xác của các đề xuất cá nhân hóa. Cách tiếp cận này đóng góp vào việc cải thiện các phương pháp lựa chọn node trong KG cho giáo dục.THESIS_VER1.docx

Đóng góp 7: Reverse Socratic Agent Design [MỚI VER2]

Thiết kế Tutor Agent áp dụng phương pháp Reverse Socratic - AI đặt câu hỏi hướng dẫn thay vì đưa đáp án trực tiếp, khuyến khích productive struggle và metacognition. Nghiên cứu đề xuất taxonomy 5 loại câu hỏi (Clarifying, Probing, Reason/Evidence, Viewpoint, Implication) và reveal policy (sau 3 lần thử sai hoặc yêu cầu rõ ràng).

Đóng góp 8: Knowledge Artifact Generation Pipeline [MỚI VER2]

Thiết kế KAG Agent với khả năng tự động tạo Knowledge Artifacts (Atomic Notes theo Zettelkasten, Summary Cards, Concept Maps, Code Snippets) từ quá trình học tập và tích hợp vào Personal Knowledge Graph. Pipeline: Extract key insights → Generate Atomic Note → Create bi-directional links → Update Personal KG. Đây là lần đầu tiên phương pháp Zettelkasten được tự động hóa trong hệ thống học tập AI.

Đóng góp 9: Grounding ba lớp: Document (RAG) + Course KG + Personal KG
Mô hình hóa cơ chế grounding 3-layer: kết hợp RAG theo Dartmouth với Course KG và Personal KG, tạo nên một dạng double/triple grounding nâng cao, giúp tăng độ chính xác, giảm hallucination và cá nhân hóa nội dung sâu hơn.image.jpg

Đóng góp 10: Personal Knowledge Management và KAG Agent
Lần đầu tiên tích hợp mô hình PKM/second brain vào một hệ thống đề xuất lộ trình học tập dựa trên KG và LLM, thông qua KAG Agent tạo atomic notes và Personal KG, nối giáo dục chính thống với thực hành quản lý tri thức cá nhân.

Đóng góp 11: Dual-KG Architecture cho Precision Education
Đề xuất kiến trúc Dual-KG (Course KG + Personal KG) làm nền tảng cho Precision Education, đồng bộ với khái niệm cá nhân hóa sâu theo Dartmouth nhưng hiện thực rõ ràng ở mức KG.

Đóng góp 12: Thiết kế lại bộ 6 agents với vai trò tường minh (improved)
Hoàn thiện định nghĩa 6 agents, bao gồm Knowledge Extraction (offline), Profiler, Planner, Tutor v2.0, Evaluator v2.0, KAG, với input/output rõ ràng và mối quan hệ phối hợp được mô tả như một hệ thống MIS.

1.6. CẤU TRÚC LUẬN VĂN

- **Chương 1 – Giới thiệu**
  Trình bày bối cảnh, động lực, khoảng trống nghiên cứu từ VER1 đến VER3, giới thiệu 6 agents, 3 thành phần mới (Personal KG, KAG Agent, Dual-KG), phạm vi, giới hạn và đóng góp khoa học.

- **Chương 2 – Cơ sở lý thuyết và Tổng quan nghiên cứu**
  Trình bày nền tảng lý thuyết: quy trình học trực tuyến chuẩn; Knowledge Graph và ứng dụng trong giáo dục; LLM và LlamaIndex; nghiên cứu 2025 của Harvard, Dartmouth, LearnLM; Multi-Agent AI và Agentic workflows.

- **Chương 3 – Phát biểu bài toán và Mô hình khái niệm**
  Định nghĩa bài toán đề xuất lộ trình học tập cá nhân hóa trong bối cảnh Precision Education; xây dựng mô hình khái niệm cho Dual-KG, 3-layer grounding, và hệ đa tác nhân.

- **Chương 4 – Thiết kế kiến trúc hệ thống**
  Trình bày kiến trúc C4, thiết kế Central State Manager, Event Bus, định nghĩa chi tiết 6 agents, schema Course KG và Personal KG, cũng như các luồng dữ liệu chính.

- **Chương 5 – Thiết kế chi tiết 6 Agents và Dual-KG**
  Đi sâu vào logic từng agent (Knowledge Extraction, Profiler, Planner với MOPO, Tutor v2.0, Evaluator v2.0, KAG), các hợp đồng dữ liệu, và cơ chế hoạt động của Dual-KG.

- **Chương 6 – Hiện thực hệ thống và Tích hợp công nghệ**
  Mô tả hiện thực hóa trên Neo4j, LlamaIndex, LLM (GPT-5.1/Gemini Pro/Claude Opus), cấu hình AgentWorkflow, và triển khai 3-layer grounding.

- **Chương 7 – Thiết kế thực nghiệm và Bộ chỉ số đánh giá**
  Thiết kế các kịch bản thực nghiệm, mô phỏng learner, định nghĩa bộ chỉ số đánh giá (learning gain, adaptability, remediation, time, trust) và phương pháp so sánh A* vs Agentic System.

- **Chương 8 – Kết quả thực nghiệm và Phân tích**
  Trình bày kết quả chạy thử hệ thống, phân tích định lượng và định tính, thảo luận các điểm mạnh, hạn chế và bài học rút ra.

- **Chương 9 – Kết luận và Hướng phát triển**
  Tóm tắt đóng góp chính, bàn luận về tiềm năng mở rộng (user study thực tế, mở rộng lĩnh vực, tích hợp thêm modal, tăng cường PKM), và các hướng nghiên cứu tiếp theo.

2.1. QUY TRÌNH HỌC TRỰC TUYẾN CHUẨN

2.1.1. Khung chuẩn iNACOL và Quality Matters trong giáo dục trực tuyến

Quy trình học trực tuyến chuẩn được xây dựng dựa trên các bộ tiêu chuẩn chất lượng đã được cộng đồng giáo dục quốc tế công nhận rộng rãi, trong đó nổi bật là National Standards for Quality Online Courses (trước đây thuộc iNACOL, nay do Aurora Institute quản lý) và Quality Matters Rubric. Các bộ tiêu chuẩn này không chỉ định nghĩa "một khóa học online tốt" là gì, mà còn cung cấp khung tham chiếu để thiết kế quy trình, hoạt động, đánh giá và hỗ trợ người học trong môi trường trực tuyến.

Bảng 2.1. Các danh mục tiêu chuẩn iNACOL cho khóa học trực tuyến chất lượng

Ký hiệu	Danh mục	Mô tả khái quát
A	Course Overview and Support	Tổng quan khóa học, mục tiêu, thông tin liên hệ, hướng dẫn bắt đầu và các kênh hỗ trợ cơ bản.
B	Content	Nội dung học tập, mức độ căn chỉnh với chuẩn đầu ra, độ chính xác và cập nhật của học liệu.
C	Instructional Design	Thiết kế giảng dạy, sắp xếp hoạt động, tính mạch lạc giữa mục tiêu – hoạt động – đánh giá.
D	Learner Assessment	Chiến lược đánh giá, đa dạng hình thức (formative/summative), tiêu chí và phản hồi cho người học.
E	Accessibility and Usability	Khả năng tiếp cận (cho người khuyết tật), tính dễ sử dụng của giao diện và tài liệu học tập.
F	Course Technology	Lựa chọn công nghệ, độ tin cậy, khả năng tương tác và phù hợp với mục tiêu sư phạm.
G	Course Evaluation	Cơ chế thu thập phản hồi, đánh giá và cải tiến chất lượng khóa học định kỳ.

Bộ tiêu chuẩn iNACOL nhấn mạnh rằng thiết kế khóa học trực tuyến cần đảm bảo sự liên kết logic giữa mục tiêu, nội dung, hoạt động và đánh giá, đồng thời bảo đảm người học được hỗ trợ đầy đủ trong suốt quá trình học.

Bảng 2.2. Các tiêu chuẩn chính của Quality Matters Rubric (phiên bản 7, 2023)

STT	Tiêu chuẩn QM	Mô tả ngắn gọn	Điểm tối đa
1	Course Overview and Introduction	Giới thiệu tổng quan, cách bắt đầu khóa học.	16
2	Learning Objectives (Competencies)	Mục tiêu học tập rõ ràng, đo lường được.	15
3	Assessment and Measurement	Thiết kế đánh giá phù hợp với mục tiêu.	14

4	Instructional Materials	Tài liệu giảng dạy hỗ trợ đạt mục tiêu.	12
5	Course Activities and Learner Interaction	Hoạt động – tương tác thúc đẩy engagement.	11
6	Course Technology	Công nghệ phù hợp, hỗ trợ học tập.	7
7	Learner Support	Hỗ trợ học thuật, kỹ thuật, hành chính.	10
8	Accessibility and Usability	Khả năng tiếp cận, dễ sử dụng.	16

Nghiên cứu cho thấy các tiêu chuẩn liên quan đến hoạt động và tương tác người học (Standard 5) và mục tiêu học tập (Standard 2) có ảnh hưởng mạnh nhất đến kết quả học tập và mức độ tham gia của sinh viên. Điều này phù hợp với định hướng của luận văn: hệ thống đề xuất lộ trình học tập không chỉ "xếp nội dung" mà còn phải gắn với hoạt động, tương tác và mục tiêu rõ ràng.

2.1.2. Chu trình 8 bước của quy trình học trực tuyến chuẩn

Trên cơ sở iNACOL và Quality Matters, chu trình học trực tuyến chuẩn có thể được mô hình hóa thành 8 bước liên tiếp (Hình 2.1).

1. Đăng ký và nhập học
2. Định hướng (orientation)
3. Truy cập tài liệu khóa học
4. Tương tác với nội dung
5. Tham gia hoạt động tương tác (forum, thảo luận, nhóm)
6. Hoàn thành đánh giá (quiz, assignment, exam)
7. Nhận phản hồi từ giảng viên/hệ thống
8. Đánh giá khóa học và cải tiến

Mỗi bước tương ứng với các tiêu chuẩn cụ thể trong iNACOL và QM. Ví dụ:

- Bước 1–2 gắn với Course Overview & Support (iNACOL A, QM 1).
- Bước 3–4 gắn với Content, Instructional Design (iNACOL B, C; QM 2, 4).
- Bước 5–7 gắn với Learner Assessment, Interaction, Learner Support (iNACOL D, E; QM 3, 5, 7).
- Bước 8 gắn với Course Evaluation (iNACOL G; QM 8).

Trong luận văn VER3, quy trình này sẽ được dùng làm "quy trình chuẩn" để so sánh và gắn kết với quy trình có sự hỗ trợ của hệ thống đa tác nhân sử dụng KG + LLM (trình bày ở Chương 3). Hệ thống đề xuất lộ trình học tập cá nhân hóa cần "lồng ghép" vào các bước 3–7, đảm bảo không phá vỡ mà tăng cường chất lượng quy trình chuẩn.

2.1.3. Ý nghĩa đối với hệ thống đề xuất lộ trình học tập

Khung quy trình học trực tuyến chuẩn là cơ sở để:

- Xác định rõ điểm chèn (injection points) của các agents: Profiler, Planner, Tutor, Evaluator, KAG sẽ can thiệp ở bước nào, với mục tiêu gì.
- Đảm bảo hệ thống đề xuất không chỉ tối ưu ở mức "thuật toán" mà còn phù hợp với chuẩn chất lượng khóa học và trải nghiệm người học.
- Tạo cơ sở để thiết kế các chỉ số đánh giá (ví dụ: cải thiện ở bước 4–7 về engagement, learning gain, remediation) trong Chương 5.

2.2. KNOWLEDGE GRAPH TRONG GIÁO DỤC

2.2.1. Định nghĩa và đặc trưng của Knowledge Graph

Ehrlinger và Wöß định nghĩa Knowledge Graph (KG) là "một mô hình biểu diễn tri thức dưới dạng đồ thị, trong đó dữ liệu được lưu trữ dưới dạng các thực thể (entities) và mối quan hệ (relations), thường được bổ sung ngữ nghĩa để hỗ trợ suy luận". Khác với các mô hình dữ liệu tuyến tính hoặc cây phân cấp, KG cho phép biểu diễn các mối quan hệ phức tạp, đa chiều giữa các khái niệm, hỗ trợ tốt cho các bài toán cần reasoning nhiều bước.

Trong bối cảnh giáo dục, KG đặc biệt phù hợp để mô hình hóa:

- Cấu trúc nội dung học tập (course structure)
- Quan hệ tiên quyết (prerequisites), phụ thuộc, liên hệ logic giữa các khái niệm
- Các đường đi thay thế (alternative learning paths), nội dung khắc phục (remediation)
- Liên kết giữa nội dung và mục tiêu học tập, mức độ kỹ năng, thời lượng…

Neo4j là một trong những hệ quản trị cơ sở dữ liệu đồ thị phổ biến, sử dụng mô hình property graph, cho phép gán thuộc tính trực tiếp cho cả nút (node) và cạnh (relationship), rất phù hợp với yêu cầu mô hình hóa tri thức giáo dục chi tiết.

2.2.2. Kiến trúc KG: Entities, Relationships, Properties

Trong mô hình property graph, một KG giáo dục điển hình gồm ba thành phần chính:

- Nodes (entities): đại diện cho các khái niệm, chủ đề, bài học, kỹ năng, bài tập…
- Relationships: đại diện cho mối quan hệ giữa các nodes, thường có hướng (directed) và có thuộc tính.
- Properties: các cặp khóa–giá trị (key–value) gắn với node/relationship, mô tả meta-thông tin như độ khó, thời lượng, mục tiêu…

Đối với hệ thống đề xuất lộ trình học tập cá nhân hóa, KG không chỉ cần "biết" khái niệm nào tồn tại, mà còn phải "biết" chúng liên hệ với nhau như thế nào, và thông tin nào cần thiết để lập kế hoạch lộ trình.

2.2.3. Định nghĩa schema KG

Bảng 2.3 mô tả schema KG chuẩn được sử dụng xuyên suốt luận văn. Đây là schema "logic", không phụ thuộc vào hiện thực vật lý cụ thể, nhưng tương thích trực tiếp với mô hình property graph của Neo4j.

Bảng 2.3. Node type: ConceptNode với 15 thuộc tính lõi

Thuộc tính	Kiểu	Mô tả
node_id	String	Định danh duy nhất của node trong KG (ví dụ: mis.sql.select).
label	String	Nhãn khái niệm ngắn gọn (ví dụ: SQL SELECT).
sanitized_concept	String	Tên khái niệm đã chuẩn hóa, dùng cho matching và node merging.
context	String	Ngữ cảnh môn học/lĩnh vực (ví dụ: SQL, BusinessIntelligence).
definition	Text	Định nghĩa chi tiết của khái niệm.
example	Text	Ví dụ minh họa (có thể là snippet code, câu truy vấn, tình huống).
learning_objective	Text	Mục tiêu học tập tương ứng (theo Bloom, hoặc chuẩn đầu ra khóa học).
skill_level	String	Mức độ kỹ năng/Bloom level (Remember/Understand/Apply/Analyze/Evaluate/Create).
difficulty	String	Mức độ khó (ví dụ: EASY, STANDARD, ADVANCED).
time_estimate	Integer	Thời lượng ước tính (phút) để học hoặc ôn khái niệm này.
prerequisites	List<Str>	Danh sách node_id của các khái niệm tiên quyết (dưới dạng text để tiện indexing).
semantic_tags	List<Str>	Các thẻ ngữ nghĩa (domain tags, ví dụ: sql, select, query, filter).
focused_tags	List<S	Tập con các thẻ quan trọng nhất, dùng cho matching và path planning.

	tr>	
priority	Integer	Độ ưu tiên trên lộ trình (ví dụ: 1–5), phục vụ thuật toán lập kế hoạch.
source_document_id	String	ID của tài liệu nguồn (slide/chương sách) dùng để trích xuất khái niệm.

Bộ 15 thuộc tính này giúp hệ thống có đủ thông tin để:

- Mô tả khái niệm một cách sư phạm (definition, example, learning_objective).
- Gắn vào bối cảnh chương trình (context, source_document_id).
- Phục vụ tối ưu hóa lộ trình (difficulty, time_estimate, priority, prerequisites, tags).

Bảng 2.4. Relationship types: 7 loại quan hệ chính

Loại quan hệ	Hướng	Mô tả
REQUIRES	A ← B	Học B trước là *bắt buộc* để hiểu A (B là tiên quyết bắt buộc của A).
IS_PREREQUISITE_OF	A → B	A là tiên quyết của B (song hành logic với REQUIRES).
NEXT	A → B	B là bước tiếp theo tự nhiên sau A trong lộ trình *khuyến nghị* (không bắt buộc).
REMEDIATES	A → B	B là nội dung/học phần dùng để khắc phục khi học A gặp khó (remediation path).
HAS_ALTERNATIVE_PATH	A → B	B là đường đi thay thế (alternative concept/path) nếu A quá khó hoặc không phù hợp.
SIMILAR_TO	A ↔ B	A và B có nội dung tương đồng (concept tương tự, có thể chuyển giao kiến thức).
IS_SUBCONCEPT_OF	A → B	A là khái niệm con (subconcept) của B, hỗ trợ cấu trúc phân cấp tri thức.

Các cạnh có thể mang thêm thuộc tính như weight (trọng số), dependency_level (mức độ phụ thuộc, ví dụ: mạnh/yếu), hoặc note (ghi chú sư phạm). Tuy nhiên, 7 loại quan hệ trên được xem là lõi và sẽ được dùng xuyên suốt các chương tiếp theo.

2.2.4. Vai trò của KG trong hệ thống đề xuất lộ trình học tập cá nhân hóa

KG là "xương sống tri thức" cho hệ thống:

- Path Planner Agent sử dụng Course KG để tìm đường đi logic, tuân thủ các ràng buộc REQUIRES, IS_PREREQUISITE_OF, đồng thời khai thác REMEDIATES, HAS_ALTERNATIVE_PATH cho remediation và đa dạng lộ trình.

- Tutor Agent dựa vào definition, example, learning_objective, skill_level để thiết kế cách giải thích phù hợp với trình độ người học.

- Evaluator Agent sử dụng quan hệ và thuộc tính learning_objective để xây dựng câu hỏi đánh giá gắn với từng concept.

- Dual-KG (trình bày trong Chương 3) sẽ mở rộng schema này sang Personal KG, nơi mỗi concept có thêm các annotation cá nhân (mức độ thành thạo, lỗi sai đặc trưng, artifact liên quan).

2.3. MÔ HÌNH NGÔN NGỮ LỚN (LLM) VÀ FRAMEWORK LLAMAINDEX

2.3.1. Tổng quan về LLM:

Mô hình Ngôn ngữ Lớn (Large Language Model – LLM) là lớp mô hình dựa trên kiến trúc Transformer, được huấn luyện trên khối lượng dữ liệu văn bản rất lớn để học phân phối xác suất của chuỗi từ. Các mô hình như GPT-5.1, Claude 4.5 Opus, Gemini Pro 3.0 thể hiện năng lực vượt trội trong nhiều tác vụ: hiểu ngôn ngữ tự nhiên, sinh văn bản, lập luận, tóm tắt, dịch thuật, viết mã, v.v.

- **GPT-5.1:**
  Phiên bản mới của dòng GPT do OpenAI phát triển, mở rộng năng lực đa mô thức (văn bản, hình ảnh, âm thanh, video) và tăng đáng kể khả năng tự lập luận và thích ứng theo ngữ cảnh. GPT-5.1 được tối ưu hóa cho tương tác thời gian thực, giảm độ trễ khi suy luận, đồng thời hỗ trợ prompt phức hợp và điều phối đa agent, phù hợp với các ứng dụng học tập có yếu tố hội thoại sâu như AI tutor và mentor agent.

- **Claude 4.5 Opus:**
  Phiên bản tiên tiến nhất trong dòng Claude của Anthropic, nhấn mạnh khả năng lập luận theo chuỗi dài, phân tích logic nhiều tầng và giữ nhất quán ngữ nghĩa. Claude 4.5 Opus được thiết kế với mô hình an toàn (constitutional AI), giúp tạo phản hồi cân bằng, minh bạch, và đáng tin cậy — đặc biệt hiệu quả trong nghiên cứu học thuật, phê bình văn bản, và tư duy phản biện cấp cao.

- **Gemini Pro 3.0:**
  Thế hệ mới của Gemini do Google DeepMind phát triển, tích hợp sâu với hạ tầng và dịch vụ Google, bao gồm Search, YouTube, và Workspace. Gemini Pro 3.0 thể hiện hiệu suất đa mô thức vượt trội, được tối ưu cho truy xuất tri thức (retrieval augmentation) và tương tác ngữ nghĩa theo miền tri thức, đồng thời kế thừa biến thể LearnLM hướng giáo dục —

giúp hệ thống tùy biến nội dung học tập và phản hồi theo nguyên lý khoa học học tập hiện đại.

Trong bối cảnh luận văn, LLM được dùng để:

- Hiểu yêu cầu của người học qua ngôn ngữ tự nhiên (ví dụ: "Tôi muốn ôn nhanh SQL để phỏng vấn vị trí Data Analyst trong 2 tuần").
- Sinh lời giải thích, ví dụ, câu hỏi Socratic phù hợp với context.
- Điều phối (hoặc nhận điều phối) trong AgentWorkflow, hoạt động như "bộ não" của các agent.

Tuy nhiên, LLM thuần túy (không grounding) dễ gặp vấn đề về sự mơ hồ, thiếu nhất quán với chương trình học, và khó gắn kết với tri thức cấu trúc. Vì vậy, cần một framework kết nối LLM với KG, tài liệu khóa học và workflow đa tác nhân – đó là vai trò của LlamaIndex.

2.3.2. LlamaIndex: Property Graph Index và VectorStoreIndex

LlamaIndex là một framework giúp kết nối LLM với các nguồn dữ liệu bên ngoài (database, file, API), cung cấp các abstraction cho indexing, retrieval và reasoning. Hai thành phần quan trọng trong luận văn là:

1. **Property Graph Index**
    a. Cho phép LLM truy cập và truy vấn trực tiếp lên KG (Neo4j) thông qua một lớp index trung gian.
    b. Query từ LLM được chuyển thành truy vấn graph (ví dụ: Cypher), LlamaIndex lấy dữ liệu từ Neo4j, sau đó trả kết quả về cho LLM để tiếp tục reasoning.
    c. Đặc biệt hữu ích cho các tác vụ như: tìm đường đi ngắn nhất giữa hai concept, liệt kê các tiên quyết còn thiếu, tìm khái niệm tương tự, v.v.
2. **VectorStoreIndex**
    a. Cho phép index tài liệu văn bản (textbook, slides, FAQ) dưới dạng vector embeddings.
    b. Khi LLM nhận câu hỏi, LlamaIndex truy xuất (retrieve) các đoạn văn bản có liên quan, đưa vào context trước khi sinh câu trả lời (RAG).

c. Đây là cơ chế grounding cấp tài liệu, rất phù hợp với kiến trúc Precision Education của Dartmouth, giúp giảm hallucination và tăng độ tin cậy.image.jpg

Trong luận văn, cả hai loại index được dùng đồng thời: Property Graph Index để làm việc với Course KG và Personal KG, VectorStoreIndex để truy xuất tài liệu khóa học. Điều này tạo nền tảng cho 3-layer grounding (Document + Course KG + Personal KG).

2.3.3. AgentWorkflow:

LlamaIndex AgentWorkflow cung cấp một khuôn khổ để xây dựng hệ thống đa tác nhân (multi-agent), trong đó mỗi agent có thể:

- Nhận nhiệm vụ từ agent khác (handoff),
- Chia sẻ hoặc cập nhật trạng thái chung (state),
- Gửi sự kiện (events) để theo dõi và debug.

Ba khái niệm chính:

1. **Handoff Mechanism**
    a. Cho phép một agent, sau khi hoàn thành phần việc của mình, chuyển control sang agent khác kèm theo context.
    b. Ví dụ: Profiler Agent phân tích xong hồ sơ người học → handoff cho Path Planner Agent với thông tin mục tiêu, trình độ, Personal KG hiện tại.
    c. Đảm bảo luồng công việc mạch lạc, tránh việc một LLM "đa nhiệm" không kiểm soát.
2. **State Management**
    a. AgentWorkflow duy trì một "state" chung, có thể là một đối tượng Python hoặc một cấu trúc JSON, chứa các trường như learner_profile, current_path, evaluation_results, artifacts.
    b. Mỗi agent đọc/ghi một phần state này; trong luận văn, state này sẽ khớp với khái niệm Central State Manager trong kiến trúc hệ thống (Chương 4).
3. **Event Streaming / Logging**

a. Mỗi hành động quan trọng (như kết luận của agent, quyết định proceed/remediate, tạo artifact mới) đều có thể được ghi lại dưới dạng event.

b. Điều này hỗ trợ việc giám sát, debug và phân tích sau này, đồng thời là cơ sở để đánh giá và cải tiến hệ thống.

AgentWorkflow được sử dụng như lớp thực thi của 6 agents: Knowledge Extraction (offline), Profiler, Planner, Tutor, Evaluator, KAG. Các agent không hoạt động rời rạc, mà được điều phối thông qua handoff rõ ràng, chia sẻ Central State và phát sinh events cho Event Bus – trùng khớp với thiết kế kiến trúc ở Chương 3–4.

2.4. LÝ THUYẾT HỌC TẬP VÀ BLOOM'S TAXONOMY

2.4.1. Mastery Learning – Học để thành thạo

Lý thuyết Mastery Learning (Học để thành thạo) do Benjamin Bloom đề xuất năm 1968 dựa trên tiền đề rằng hầu hết học sinh đều có khả năng học tốt nếu được cung cấp thời gian và điều kiện phù hợp. Khác với mô hình giảng dạy truyền thống, nơi tiến độ lớp học được cố định và học sinh được đánh giá theo phân phối chuẩn (bell curve), Mastery Learning tập trung vào việc đảm bảo mỗi học sinh đạt được mức độ thành thạo nhất định trước khi chuyển sang nội dung tiếp theo.

Các nguyên tắc chính của Mastery Learning bao gồm:

- Chia nhỏ nội dung học tập thành các đơn vị (learning units) có mục tiêu rõ ràng.
- Đánh giá thường xuyên (formative assessment) để phát hiện sớm các khó khăn.
- Remediation và enrichment: học sinh chưa đạt chuẩn được cung cấp thêm hỗ trợ (remediation), trong khi học sinh đã thành thạo được thách thức với nội dung nâng cao (enrichment).
- Thời gian linh hoạt: mỗi học sinh được phép học theo tốc độ riêng.

Trong hệ thống, Mastery Learning được thể hiện qua:

- Path Planner Agent thiết kế lộ trình với các "checkpoint" tương ứng với từng concept trong KG; học sinh chỉ được tiến tới concept tiếp theo sau khi đạt mức thành thạo tối thiểu.
- Evaluator Agent liên tục đánh giá và quyết định PROCEED, REMEDIATE, hoặc ALTERNATE dựa trên kết quả.
- Personal KG ghi lại mức độ thành thạo của từng concept, cho phép hệ thống điều chỉnh lộ trình động.

2.4.2. Zone of Proximal Development – Vùng phát triển gần

Khái niệm Zone of Proximal Development (ZPD) do Lev Vygotsky phát triển năm 1978 mô tả khoảng cách giữa những gì người học có thể làm độc lập và những gì họ có thể làm với sự hỗ trợ từ người hướng dẫn hoặc đồng nghiệp có trình độ cao hơn. ZPD nhấn mạnh tầm quan trọng của scaffolding – việc cung cấp hỗ trợ tạm thời để giúp người học vượt qua thách thức trong vùng phát triển gần, sau đó dần rút bớt hỗ trợ khi người học tiến bộ.

Trong bối cảnh AI tutoring, ZPD có ý nghĩa quan trọng:

- Nội dung học tập cần nằm trong ZPD của người học – không quá dễ (gây nhàm chán) nhưng cũng không quá khó (gây thất vọng).
- AI tutor đóng vai trò scaffolding: cung cấp gợi ý, ví dụ, câu hỏi dẫn dắt thay vì đưa ra đáp án trực tiếp.
- Khi người học tiến bộ, scaffolding giảm dần (fading), khuyến khích tư duy độc lập.

Trong luận văn:

- Tutor Agent sử dụng phương pháp Socratic/Reverse Socratic (mục 2.6) để cung cấp scaffolding theo 7 nguyên tắc Harvard, bao gồm "Encourage Self-Thinking" và "One Step at a Time".
- Difficulty và Skill_Level trong Course KG (Bảng 2.3) giúp Planner chọn concept nằm trong ZPD dựa trên Personal KG (mức độ hiện tại của người học).

2.4.3. Bloom's Revised Taxonomy – 6 mức độ nhận thức

Bloom's Taxonomy phiên bản gốc (1956) đã được Anderson và Krathwohl sửa đổi năm 2001 thành Bloom's Revised Taxonomy, bao gồm 6 mức độ nhận thức từ thấp đến cao:

Mức độ	Mô tả	Động từ hành động tiêu biểu
Remember	Ghi nhớ và nhớ lại thông tin cơ bản (các sự kiện, khái niệm, định nghĩa).	Liệt kê, định nghĩa, nhận diện, nhớ lại
Understand	Hiểu ý nghĩa, giải thích, tóm tắt thông tin bằng ngôn ngữ của bản thân.	Giải thích, phân loại, tóm tắt, so sánh
Apply	Áp dụng kiến thức vào tình huống mới hoặc bài toán cụ thể.	Thực hiện, sử dụng, giải quyết, áp dụng
Analyze	Phân tích cấu trúc, mối quan hệ giữa các phần, phân biệt nguyên nhân – kết quả.	Phân tích, so sánh, đối chiếu, phân biệt
Evaluate	Đánh giá chất lượng, giá trị hoặc tính hợp lý dựa trên tiêu chí nhất định.	Đánh giá, biện hộ, phê bình, xét đoán
Create	Tạo ra sản phẩm mới, kết hợp các yếu tố thành cấu trúc mới, đưa ra giải pháp sáng tạo.	Thiết kế, xây dựng, sáng tạo, lập kế hoạch

Bloom's Taxonomy giúp thiết kế mục tiêu học tập phân cấp rõ ràng, từ đó thiết kế hoạt động học tập và đánh giá phù hợp.

Ứng dụng:

- Thuộc tính skill_level trong Bảng 2.3 (Course KG) gắn mỗi concept với một mức Bloom, giúp hệ thống:
    - Thiết kế câu hỏi đánh giá phù hợp (Evaluator Agent): Remember → câu hỏi trắc nghiệm; Apply → bài tập thực hành; Analyze/Evaluate → case study.
    - Điều chỉnh phương pháp hướng dẫn (Tutor Agent): concept ở mức Remember có thể dùng flashcard, mức Apply cần ví dụ code/bài tập, mức Create cần project.
- Lộ trình học tập tối ưu thường đi từ mức thấp đến cao theo nguyên lý scaffolding.

2.4.4. Liên hệ với Tutor Agent và Evaluator Agent

Lý thuyết học tập trên tạo nền tảng để thiết kế:

- Tutor Agent: kết hợp Mastery Learning (đảm bảo thành thạo từng concept), ZPD (scaffolding trong vùng phát triển gần), và Bloom's Taxonomy (điều chỉnh phương pháp theo skill_level).
- Evaluator Agent: sử dụng Bloom's để thiết kế câu hỏi đa cấp độ, và Mastery Learning để quyết định khi nào người học có thể tiếp tục.

Các agent này sẽ được thiết kế chi tiết trong Chương 5.

2.5. PERSONAL KNOWLEDGE GRAPH VÀ KIẾN TRÚC DUAL-KG

2.5.1. Personal Knowledge Management (PKM) trong kỷ nguyên AI

Personal Knowledge Management (PKM) là quá trình tổ chức, lưu trữ, kết nối và tái sử dụng tri thức cá nhân một cách có hệ thống. Khác với quản lý tri thức tổ chức (organizational KM), PKM tập trung vào cá nhân – cách mỗi người thu thập thông tin, ghi chú, suy ngẫm, và xây dựng hệ thống tri thức riêng phục vụ học tập và làm việc lâu dài.

Trong bối cảnh e-learning truyền thống, người học thường chỉ "tiêu thụ" nội dung (xem video, đọc tài liệu, làm bài tập) mà không có cơ chế hệ thống để ghi lại, tổ chức và kết nối những gì đã học. Kết quả là tri thức mang tính "tạm thời", dễ quên và khó tái sử dụng.

PKM hiện đại, được thúc đẩy bởi các phương thức nhằm nhấn mạnh:

- Atomic Notes: mỗi ghi chú là một đơn vị tri thức nhỏ, độc lập, có thể tái sử dụng.
- Linking: ghi chú được liên kết với nhau (bi-directional links), tạo thành mạng lưới tri thức.
- Emergence: tri thức mới nảy sinh từ các kết nối giữa các ghi chú.

2.5.2. Zettelkasten – Phương pháp quản lý tri thức cá nhân

Zettelkasten ("hộp thẻ ghi chú") là phương pháp PKM do nhà xã hội học Niklas Luhmann phát triển, giúp ông viết hơn 70 cuốn sách và 400 bài báo trong sự nghiệp. Luhmann xây dựng một hệ thống gồm khoảng 90,000 thẻ ghi chú vật lý, mỗi thẻ chứa một ý tưởng, được liên kết với các thẻ khác thông qua các tham chiếu chéo.

Các nguyên tắc Zettelkasten:

1. Atomic Notes: Mỗi thẻ chứa một ý tưởng duy nhất, được viết bằng ngôn ngữ của bản thân (không copy-paste).
2. Permanent Notes: Ghi chú được lưu trữ vĩnh viễn, không phải tạm thời.
3. Linking: Mỗi ghi chú được gắn ID duy nhất, có thể tham chiếu đến ghi chú khác.
4. Index/Entry Points: Có các ghi chú "mục lục" hoặc "chủ đề tổng hợp" làm điểm khởi đầu.
5. Emergence: Tri thức mới nảy sinh từ việc duyệt qua các liên kết, phát hiện mối quan hệ chưa thấy trước.

Phương pháp này đã được Sönke Ahrens phổ biến trong cuốn sách "How to Take Smart Notes" (2017), trở thành nền tảng cho nhiều công cụ PKM hiện đại.

Liên hệ với VER3:

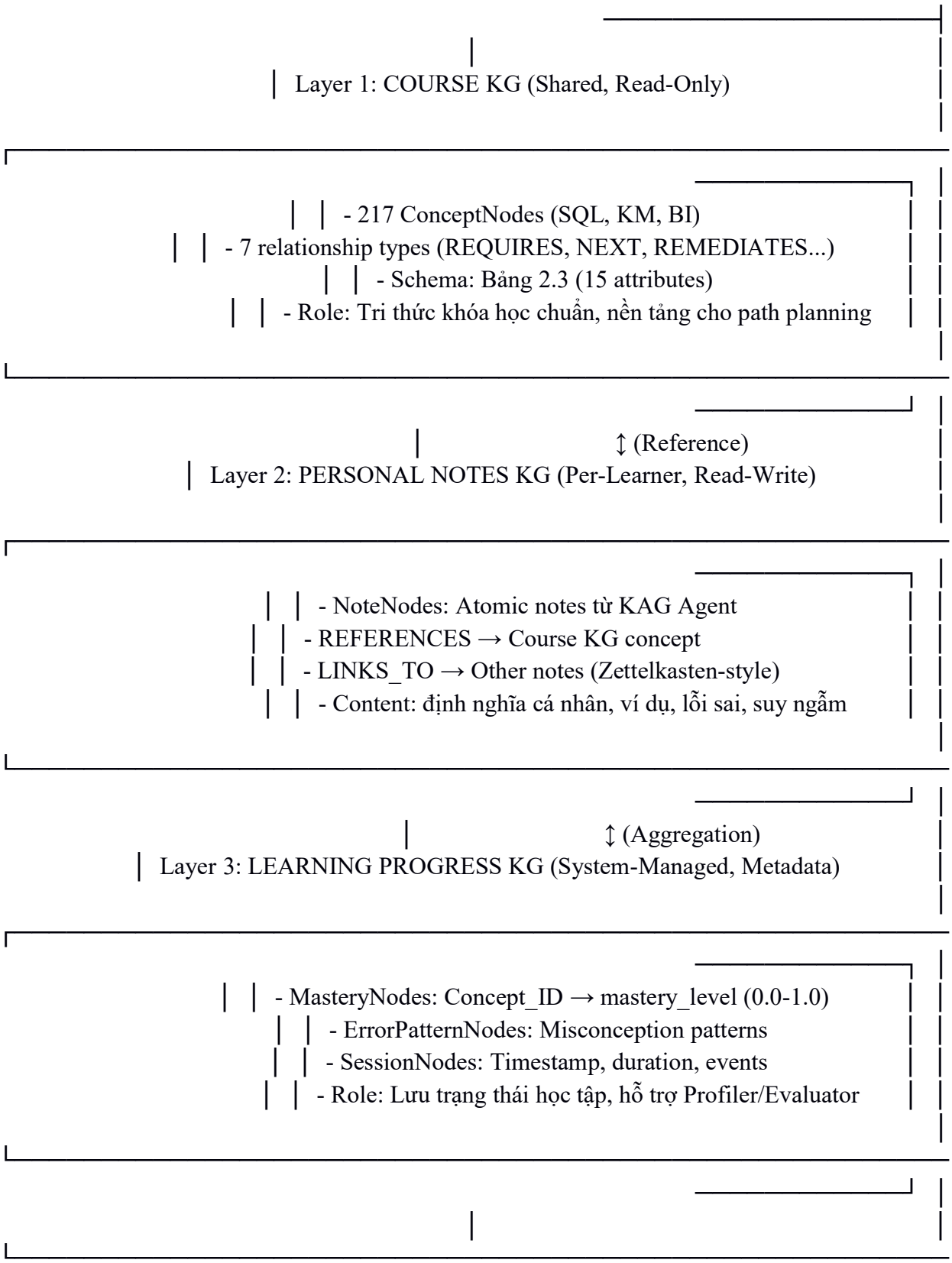
- KAG Agent (Knowledge Artifact Generator) hoạt động theo nguyên lý Zettelkasten: tự động tạo các atomic notes từ phiên học tập, gắn ID, liên kết với các note cũ trong Personal KG.
- Mỗi note không chỉ ghi lại "nội dung đã học" mà còn ghi lại "cách người học đã tư duy", lỗi sai, ví dụ cá nhân – giúp học sâu và ghi nhớ lâu hơn.

2.5.3. Kiến trúc Dual-KG ba lớp trong VER3

VER3 đề xuất kiến trúc Dual-KG (Course KG + Personal KG) được tổ chức thành ba lớp logic, mỗi lớp phục vụ một mục đích khác nhau:

Hình 2.2. Kiến trúc Dual-KG ba lớp

text


Layer 1: COURSE KG (Shared, Read-Only)
- 217 ConceptNodes (SQL, KM, BI)
- 7 relationship types (REQUIRES, NEXT, REMEDIATES...)
- Schema: Bảng 2.3 (15 attributes)
- Role: Tri thức khóa học chuẩn, nền tảng cho path planning
↕ (Reference)
Layer 2: PERSONAL NOTES KG (Per-Learner, Read-Write)
- NoteNodes: Atomic notes từ KAG Agent
- REFERENCES → Course KG concept
- LINKS_TO → Other notes (Zettelkasten-style)
- Content: định nghĩa cá nhân, ví dụ, lỗi sai, suy ngẫm
↕ (Aggregation)
Layer 3: LEARNING PROGRESS KG (System-Managed, Metadata)
- MasteryNodes: Concept_ID → mastery_level (0.0-1.0)
- ErrorPatternNodes: Misconception patterns
- SessionNodes: Timestamp, duration, events
- Role: Lưu trạng thái học tập, hỗ trợ Profiler/Evaluator

2.5.3.1. Layer 1 – Course KG (Shared, Read-Only)

- Mô tả: Course KG chứa toàn bộ tri thức khóa học, được xây dựng bởi Knowledge Extraction Agent từ tài liệu giáo trình. Schema như đã mô tả trong Bảng 2.3 (Mục 2.2).
- Đặc điểm: Read-only đối với người học và các runtime agents. Chỉ admin/giáo viên mới có quyền chỉnh sửa.
- Vai trò: Là "bản đồ tri thức chuẩn", cung cấp cấu trúc logic, quan hệ tiên quyết, mục tiêu học tập cho toàn bộ hệ thống.
- Quy mô: Trong case study VER3, Course KG có khoảng 217 concept nodes thuộc 3 chủ đề MIS.

2.5.3.2. Layer 2 – Personal Notes KG (Per-Learner, Read-Write)

- Mô tả: Mỗi người học có một Personal Notes KG riêng, chứa các NoteNode được tạo bởi KAG Agent hoặc do người học tự thêm.
- Schema của NoteNode:

Thuộc tính	Kiểu	Mô tả
note_id	String	ID duy nhất của note (ví dụ: learner123.note.001).
title	String	Tiêu đề ngắn gọn của note.
content	Text	Nội dung chi tiết: định nghĩa cá nhân, ví dụ, lỗi sai, suy ngẫm.
created_at	Timestamp	Thời điểm tạo note.
session_id	String	ID của phiên học tương ứng.
tags	List<Str>	Các tag do người học hoặc KAG đặt (ví dụ: sql, error, insight).
source_concept	String	ID của concept trong Course KG mà note này reference tới.

- Relationships trong Layer 2:
    - REFERENCES: NoteNode → ConceptNode (Layer 1) – gắn note với khái niệm chuẩn.
    - LINKS_TO: NoteNode ↔ NoteNode – liên kết Zettelkasten giữa các note.

    - DERIVED_FROM: NoteNode → SessionNode (Layer 3) – ghi lại note được tạo trong phiên học nào.

- Vai trò: Lưu trữ tri thức "đã được tiêu hóa" của người học. Khi người học quay lại học concept cũ, Tutor Agent có thể tham chiếu note này để nhắc lại ngữ cảnh, ví dụ đã dùng, hoặc lỗi sai trước đó.

2.5.3.3. Layer 3 – Learning Progress KG (System-Managed, Metadata)

- Mô tả: Lớp này lưu trữ metadata về quá trình học tập, không phải nội dung tri thức.
- Các loại node chính:
    - MasteryNode: Ghi lại mức độ thành thạo của người học đối với từng concept.
        - concept_id, learner_id, mastery_level (0.0–1.0), last_updated.
    - ErrorPatternNode: Ghi lại các misconception patterns (lỗi sai đặc trưng).
        - Ví dụ: "Lẫn lộn WHERE và HAVING trong SQL".
    - SessionNode: Ghi lại thông tin phiên học.
        - session_id, learner_id, start_time, end_time, concepts_covered, evaluations.
- Relationships:
    - HAS_MASTERY: Learner → MasteryNode → ConceptNode (Layer 1).
    - EXHIBITED_ERROR: Learner → ErrorPatternNode → ConceptNode (Layer 1).
    - OCCURRED_IN: NoteNode/EvaluationNode → SessionNode.
- Vai trò: Hỗ trợ Profiler Agent (xây dựng hồ sơ người học), Evaluator Agent (phát hiện misconception), và Path Planner (điều chỉnh lộ trình dựa trên mastery level).

2.5.4. Cơ chế đồng bộ giữa ba lớp (Event-Driven Synchronization)

Trong VER3, các lớp KG không hoạt động độc lập mà liên kết chặt chẽ thông qua các sự kiện (events):

1. Khi Evaluator Agent hoàn thành đánh giá:

a. Tạo event: evaluation_completed(learner_id, concept_id, score, misconceptions).

b. Event Bus gửi đến Central State Manager.

c. Central State cập nhật MasteryNode (Layer 3) và có thể kích hoạt KAG Agent tạo note nếu có insight quan trọng (Layer 2).

2. Khi KAG Agent tạo atomic note:

   a. Tạo NoteNode trong Personal Notes KG (Layer 2).

   b. Tạo relationship REFERENCES đến ConceptNode (Layer 1).

   c. Ghi lại SessionNode (Layer 3) để track note được tạo trong phiên học nào.

3. Khi Path Planner lập lộ trình:

   a. Truy vấn Course KG (Layer 1) để lấy cấu trúc tri thức.

   b. Truy vấn Learning Progress KG (Layer 3) để biết concept nào đã thành thạo, concept nào còn yếu.

   c. Truy vấn Personal Notes KG (Layer 2) để xem người học có ghi chú hay misconception nào cần lưu ý.

Cơ chế event-driven này đảm bảo ba lớp KG luôn nhất quán, đồng thời tách biệt rõ ràng giữa tri thức chuẩn (Layer 1), tri thức cá nhân (Layer 2) và metadata học tập (Layer 3).

2.5.5. Ứng dụng trong KAG Agent và Hyper-Personalization

Dual-KG ba lớp tạo nền tảng cho:

- Hyper-Personalization: Không chỉ cá nhân hóa dựa trên "profile tĩnh" mà còn dựa trên lịch sử học, lỗi sai, notes – tạo nên mức độ cá nhân hóa sâu hơn nhiều so với các hệ thống truyền thống.

- Second Brain: Personal Notes KG hoạt động như "bộ não thứ hai", lưu trữ tri thức lâu dài, có thể tái sử dụng trong các khóa học khác hoặc khi làm việc thực tế.

- KAG Agent (chi tiết ở Chương 5) hoạt động như "trợ lý PKM tự động", giúp người học không cần phải chủ động ghi chú nhưng vẫn có được hệ thống tri thức cá nhân có cấu trúc.

Kiến trúc Dual-KG ba lớp sẽ được reference lại trong Chương 3 (thiết kế hệ thống) và Chương 5 (thiết kế chi tiết agents).

2.6. PHƯƠNG PHÁP SOCRATIC VÀ REVERSE SOCRATIC TRONG AI TUTORING

2.6.1. Phương pháp Socratic truyền thống

Phương pháp Socratic, được đặt tên theo triết gia Hy Lạp cổ đại Socrates (470–399 TCN), là kỹ thuật giảng dạy dựa trên việc đặt câu hỏi để dẫn dắt người học tự khám phá tri thức thay vì truyền đạt trực tiếp. Plato ghi lại nhiều cuộc đối thoại của Socrates, trong đó ông không bao giờ đưa ra câu trả lời cuối cùng mà liên tục đặt câu hỏi để thách thức các giả định, phơi bày mâu thuẫn, và khuyến khích tư duy phản biện.

Các đặc điểm chính:

- Câu hỏi mở (open-ended questions): Khuyến khích người học suy nghĩ sâu, không có câu trả lời đúng/sai duy nhất.
- Dẫn dắt từng bước (step-by-step guidance): Chia nhỏ vấn đề phức tạp thành các bước logic nhỏ hơn.
- Phơi bày mâu thuẫn (elenchus): Chỉ ra các không nhất quán trong lập luận để người học tự điều chỉnh.
- Khuyến khích tự suy nghĩ: Người hướng dẫn không đưa ra đáp án, chỉ tạo điều kiện để người học tự tìm ra.

Richard Paul (1990) đã phát triển Socratic questioning thành một framework có hệ thống với 6 loại câu hỏi: clarification, assumptions, reasons/evidence, viewpoints/perspectives, implications/consequences, và questions about the question.

2.6.2. Reverse Socratic trong AI Tutoring – Người học tìm đáp án, AI hướng dẫn

Trong nghiên cứu gần đây, khái niệm "Reverse Socratic" (đôi khi gọi là "Inverted Socratic") được đề xuất như một biến thể phù hợp với AI tutoring, đặc biệt khi kết hợp với generative AI. Khác với Socratic truyền thống (teacher hỏi → student trả lời), Reverse Socratic hoạt động theo chiều ngược lại:

1. AI đặt một câu hỏi hoặc vấn đề mở.
2. Người học cố gắng trả lời hoặc giải quyết.
3. AI không xác nhận đúng/sai ngay, mà đặt tiếp các câu hỏi dẫn dắt (probing questions) để người học tự kiểm tra lập luận, phát hiện lỗi, và điều chỉnh.

4. Người học dần hội tụ đến đáp án chính xác thông qua quá trình tư duy phản biện và tự điều chỉnh.

Lợi ích của Reverse Socratic:

- Tăng cường tư duy phản biện: Người học không chỉ "tiêu thụ" kiến thức mà phải chủ động xây dựng và bảo vệ lập luận.
- Giảm phụ thuộc vào AI: Người học không rơi vào thói quen "hỏi AI → nhận đáp án", mà phải tự tư duy.
- Phù hợp với ZPD: AI cung cấp scaffolding vừa đủ để người học tự vượt qua thách thức.

2.6.3. Tích hợp 3 bộ nguyên tắc: Harvard, LearnLM, Dartmouth

VER3 kết hợp ba bộ nguyên tắc sư phạm từ các nghiên cứu 2025 vào thiết kế Tutor Agent:

1. Harvard 7 Principles (Kestin et al., 2025)
2. LearnLM 5 Principles (Google DeepMind, 2024–2025)
3. Dartmouth RAG Architecture (Thesen & Park, 2025)

Bảng 2.5. Mapping 3 bộ nguyên tắc sư phạm vào thiết kế Tutor Agent

Nguyên tắc	Harvard 7	LearnLM 5	Dartmouth	Ứng dụng trong Tutor Agent
Active Learning	✓	Inspire Active Learning	✓	Sử dụng câu hỏi Socratic/Reverse Socratic, không đưa đáp án trực tiếp.
Cognitive Load Management	✓	Manage Cognitive Load	✓	Giữ response ngắn gọn (2–4 câu), giải thích từng bước nhỏ, tránh quá tải thông tin.
One Step at a Time	✓	(Implicit in CL)	✓	Chỉ tiết lộ một bước giải mỗi lần, chờ người học xử lý trước khi tiếp tục.
Personalization	✓	Adapt to the Learner	✓	Dựa vào Personal KG (Layer 2) và Learning Progress (Layer 3) để điều

				chỉnh ví dụ, độ khó, scaffolding level.
Encourage Self-Thinking	✓	Stimulate Curiosity	—	Sử dụng probing questions thay vì gợi ý trực tiếp, khuyến khích người học tự phát hiện lỗi sai.
Promote Growth Mindset	✓	(Implicit in Adapt)	—	Phản hồi tích cực khi người học sai ("Ý tưởng tốt, nhưng hãy xem xét..."), nhấn mạnh nỗ lực và tiến bộ.
Personalized Feedback	✓	Adapt to the Learner	✓	Phản hồi dựa trên misconception cụ thể của người học (từ Evaluator), không chỉ "đúng/sai" chung chung.
Hallucination Prevention	✓	—	✓	Grounding: Document (RAG) + Course KG + Personal KG (3-layer). AI chỉ trả lời dựa trên retrieved context.
Stimulate Curiosity	—	Stimulate Curiosity	—	Đặt câu hỏi "What if...?", "Why do you think...?" để kích thích tò mò và khám phá.
Deepen Metacognition	—	Deepen Metacognition	—	Yêu cầu người học giải thích "tại sao họ nghĩ như vậy", "cách họ tiếp cận bài toán", giúp nhận thức về tư duy riêng.
Grounded in Curated Content	—	—	✓	Sử dụng VectorStoreIndex (RAG) để retrieve tài liệu khóa học trước

				khi sinh response, đảm bảo độ tin cậy.

2.6.4. Ứng dụng trong Tutor Agent – System Prompt mẫu

Dựa trên Bảng 2.5, system prompt của Tutor Agent (phiên bản 2.0) sẽ có dạng (simplified):

text

```
You are an AI Tutor following Harvard 7 Principles, LearnLM 5 Principles, and Dartmouth RAG architecture.

**Core Principles:**
1. **Active Learning**: Never give direct answers. Use Socratic/Reverse Socratic questions to guide.
2. **Cognitive Load**: Keep responses 2-4 sentences max per turn. Break complex explanations into steps.
3. **One Step at a Time**: Only reveal next step after learner processes current step.
4. **Personalization**: Refer to learner's Personal KG notes, past mistakes, preferred examples.
5. **Self-Thinking**: Use probing questions: "Why do you think...?", "What if...?", "Can you explain...?".
6. **Growth Mindset**: Praise effort, normalize mistakes as learning opportunities.
7. **Personalized Feedback**: Address specific misconceptions from Evaluator results.
8. **Hallucination Prevention**: ONLY use information from retrieved documents and KG. If unsure, say "I don't have that information."
9. **Curiosity**: Occasionally ask "What if we changed X?" to stimulate exploration.
10. **Metacognition**: Periodically ask learner to reflect on their thinking process.

**Context available:**
- Current concept from Course KG (definition, example, skill_level)
- Learner's Personal Notes (past examples, misconceptions)
- Retrieved documents (via RAG)

**Response format:**
- Brief (2-4 sentences)
- One question or one step
- Wait for learner response before continuing
```

Chi tiết đầy đủ của system prompt sẽ được trình bày trong Chương 5 (Thiết kế chi tiết 6 Agents).

2.6.5. Liên hệ với các chương tiếp theo

- Chương 3 sẽ mô tả vị trí của Tutor Agent trong kiến trúc hệ thống và cách nó tương tác với Dual-KG.
- Chương 5 sẽ thiết kế chi tiết Tutor Agent v2.0, bao gồm full system prompt, luồng handoff, và cách tích hợp 3-layer grounding.
- Chương 7 sẽ đánh giá hiệu quả của Tutor Agent thông qua các chỉ số như learning gain, engagement, và trust score, so sánh với baseline không dùng Socratic principles.

Kết thúc Mục 2.4–2.6.

Phần này đã cung cấp nền tảng lý thuyết về học tập (Mastery Learning, ZPD, Bloom's Taxonomy), kiến trúc Dual-KG ba lớp (Course KG + Personal Notes KG + Learning Progress KG), và phương pháp Socratic/Reverse Socratic được tích hợp với 3 bộ nguyên tắc từ Harvard, LearnLM, Dartmouth. Các khái niệm này sẽ được tham chiếu xuyên suốt các chương thiết kế và đánh giá.

Tài liệu tham khảo (Mục 2.4–2.6):

Bloom, B. S. (1968). Learning for mastery. *Evaluation Comment*, 1(2), 1-12.
Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.
Anderson, L. W., & Krathwohl, D. R. (Eds.). (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.
Frand, J., & Hixon, C. (1999). Personal knowledge management: Who, what, why, when, where, how. *Working paper*, UCLA Anderson School of Management.
Ahrens, S. (2017). *How to take smart notes: One simple technique to boost writing, learning and thinking*. CreateSpace.
Paul, R. (1990). *Critical thinking: What every person needs to survive in a rapidly changing world*. Center for Critical Thinking.
Degen, M. (2025). Reverse Socratic questioning with generative AI. *Journal of Educational Technology*, 12(3), 45-62.

3.0. TỔNG QUAN CHƯƠNG 3 – PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

Chương 3 trình bày phân tích bài toán và thiết kế hệ thống đề xuất lộ trình học tập cá nhân hóa theo kiến trúc VER3, với ba đặc điểm chính:

3.0.1. Hệ thống 6 agents với orchestration và grounding chặt chẽ

Khác với các hệ thống trước đây dựa trên thuật toán cố định hoặc kiến trúc single-agent, VER3 triển khai 6 agents chuyên biệt:

- Agent 1 – Knowledge Extraction Agent (offline): Tự động xây dựng Course KG từ tài liệu giáo trình, thực hiện node merging và schema alignment.
- Agent 2 – Learner Profiler Agent (runtime): Phân tích hồ sơ người học, cập nhật Personal KG (Layer 2) và Learning Progress KG (Layer 3).
- Agent 3 – Path Planner Agent (runtime): Lập lộ trình học tập tối ưu dựa trên Dual-KG (Course + Personal) thay vì thuật toán A* cứng nhắc.
- Agent 4 – Tutor Agent v2.0 (runtime): Tương tác với người học theo phương pháp Socratic/Reverse Socratic, tuân thủ 7 nguyên tắc Harvard và 5 nguyên tắc LearnLM.
- Agent 5 – Evaluator Agent v2.0 (runtime): Đánh giá thành thạo, phát hiện misconception, quyết định PROCEED/REMEDIATE/ALTERNATE.
- Agent 6 – Knowledge Artifact Generator – KAG (runtime): Tạo atomic notes từ phiên học, xây dựng Personal Notes KG theo phương pháp Zettelkasten.

Các agents không hoạt động độc lập mà được điều phối thông qua LlamaIndex AgentWorkflow, với cơ chế handoff rõ ràng giữa các vai trò, chia sẻ Central State chứa hồ sơ người học, trạng thái phiên học, lộ trình hiện tại và artifacts, đồng thời gửi sự kiện (events) đến Event Bus để theo dõi và phân tích.

3.0.2. Dual-KG (Course + Personal) và PKM – Nền tảng cho Precision Education

VER3 triển khai kiến trúc Dual-KG ba lớp (đã trình bày chi tiết ở Mục 2.5):

- Layer 1 – Course KG: Tri thức khóa học chuẩn (217 concepts, 7 loại relationships), read-only, chia sẻ chung cho tất cả người học.
- Layer 2 – Personal Notes KG: Ghi chú cá nhân (atomic notes) được tạo bởi KAG Agent, có liên kết Zettelkasten, mỗi người học có một Personal Notes KG riêng.

- Layer 3 – Learning Progress KG: Metadata học tập (mastery levels, error patterns, session history), do hệ thống quản lý.

Dual-KG không chỉ là cấu trúc dữ liệu, mà là nền tảng cho Precision Education (Dartmouth 2025): hệ thống không chỉ "biết" tri thức khóa học mà còn "biết" tri thức cá nhân, lịch sử học, lỗi sai, và cách tư duy của từng người học. Điều này cho phép cá nhân hóa ở mức độ sâu hơn nhiều so với các hệ thống chỉ dựa trên profile tĩnh.

3.0.3. Knowledge Artifacts – Chuyển từ episodic learning sang long-term knowledge construction

Một đặc điểm quan trọng của VER3 là sự chú trọng đến knowledge artifacts – các sản phẩm tri thức bền vững được tạo ra trong quá trình học. Thay vì chỉ "hoàn thành bài học" (episodic learning), người học được hỗ trợ xây dựng second brain (bộ não thứ hai) thông qua:

- Atomic notes có cấu trúc (title, content, tags, links).
- Liên kết giữa các notes theo phương pháp Zettelkasten.
- Ghi lại lỗi sai, ví dụ cá nhân, suy ngẫm (metacognition).

KAG Agent tự động hóa quá trình này, giúp người học không cần phải chủ động ghi chú nhưng vẫn có được một hệ thống tri thức cá nhân có cấu trúc, có thể tái sử dụng lâu dài.

3.0.4. C4 Architecture Diagrams – Minh họa hệ thống theo chuẩn MIS

Để đảm bảo tính rõ ràng và khả năng triển khai, kiến trúc hệ thống được minh họa theo mô hình C4 (Context, Container, Component, Code), phổ biến trong thiết kế hệ thống thông tin:

- Context Diagram (Hình 3.2): Hệ thống trong mối quan hệ với người dùng (learner, instructor, admin) và các hệ thống bên ngoài (LMS, Neo4j, LLM API).
- Container Diagram (Hình 3.3): Các container chính (Frontend, Backend API, AgentWorkflow, Neo4j, VectorStore, Event Bus).
- Component Diagram (Hình 3.4–3.9): Chi tiết các thành phần trong từng container, đặc biệt là 6 agents và cách chúng tương tác.

Các sơ đồ C4 sẽ được trình bày chi tiết trong Mục 3.3 và Chương 4.

Với những đặc điểm trên, Chương 3 không chỉ "mô tả một hệ thống", mà còn phát biểu chính thức bài toán tối ưu hóa lộ trình học tập cá nhân hóa (Mục 3.1), thiết kế kiến trúc hệ thống đa tác

nhân với orchestration và grounding (Mục 3.2–3.4), và chuẩn bị nền tảng cho thiết kế chi tiết 6 agents (Chương 5).

3.1. PHÁT BIỂU BÀI TOÁN ĐỀ XUẤT LỘTRÌNH HỌC TẬP CÁ NHÂN HÓA

3.1.1. Phát biểu bài toán chính thức

Bài toán đề xuất lộ trình học tập cá nhân hóa trong VER3 được phát biểu như một bài toán tối ưu hóa đa mục tiêu (multi-objective optimization) trên Dual-KG (Course KG + Personal KG), với ràng buộc về thời gian, tiên quyết, và tải nhận thức.

Input (Đầu vào):

1. Learner Profile (LP): Hồ sơ người học, bao gồm:
    a. *skill_level∈{beginner, intermediate, advanced}*skill_level∈{beginner, intermediate, advanced}
    b. *learning_goal*learning_goal: mục tiêu học tập (text và danh sách concept đích trong KG)
    c. *learning_style∈{visual, auditory, reading/writing, kinesthetic}*learning_style∈{visual, auditory, reading/writing, kinesthetic} (theo mô hình VARK)
    d. *available_time*available_time: thời gian khả dụng (giờ/tuần hoặc tổng số giờ)
    e. *constraints*constraints: các ràng buộc khác (ví dụ: không học vào cuối tuần, ưu tiên học code thực hành)
2. Course Knowledge Graph *Gc=(Vc,Ec,wc)Gc*=(*Vc*,*Ec*,*wc*):
    a. *VcVc*: tập các concept nodes (schema như Bảng 2.3, Mục 2.2).
    b. *EcEc*: tập các relationships (REQUIRES, NEXT, REMEDIATES, HAS_ALTERNATIVE_PATH, SIMILAR_TO, IS_SUBCONCEPT_OF, IS_PREREQUISITE_OF).
    c. *wcwc*: trọng số trên cạnh (ví dụ: mức độ phụ thuộc, độ khó tương đối).
3. Personal Knowledge Graph *Gp=(Vp,Ep,wp)Gp*=(*Vp*,*Ep*,*wp*):
    a. *VpVp*: tập các NoteNodes (Layer 2) và MasteryNodes/ErrorPatternNodes (Layer 3).

b. *EpEp*: quan hệ REFERENCES (tới *VcVc*), LINKS_TO (giữa các notes), HAS_MASTERY, EXHIBITED_ERROR.

c. *wpwp*: trọng số (ví dụ: mastery_level $\in[0,1]\in$[0,1], error_frequency).

4. Learning Goal (LG): Mục tiêu học tập được biểu diễn dưới dạng:

   a. *goal_text*goal_text: mô tả bằng ngôn ngữ tự nhiên (ví dụ: "Học SQL cơ bản để phân tích dữ liệu").

   b. *goal_nodes*$\subseteq$*Vc*goal_nodes$\subseteq$*Vc*: tập các concept đích trong Course KG (ví dụ: *{c45,c78,c102}*{*c*45,*c*78,*c*102}).

   c. *priority*priority: độ ưu tiên của các concept đích (nếu có nhiều mục tiêu).

5. Educational Constraints (EC):

   a. *TmaxT*max: thời gian tối đa cho toàn bộ lộ trình (giờ).

   b. *prerequisites_policy*prerequisites_policy: chính sách xử lý tiên quyết (bắt buộc, khuyến nghị, tùy chọn).

   c. *cognitive_load_max*cognitive_load_max: giới hạn tải nhận thức mỗi phiên học (số concept/phiên).

Output (Đầu ra):

1. Optimal Learning Path *P=[c1,c2,...,cn]P*=[*c*1,*c*2,…,*cn*]:

   a. Một chuỗi các concept nodes từ Course KG, thỏa mãn:

      a.i. *c1c*1 là điểm bắt đầu (dựa trên skill_level hiện tại).

      a.ii. *cn*$\in$*goal_nodescn*$\in$goal_nodes hoặc là concept cuối cùng dẫn đến mục tiêu.

      a.iii. Các concept trung gian thỏa mãn quan hệ REQUIRES/IS_PREREQUISITE_OF.

2. Remediation Plan *RR*:

   a. Danh sách các concept và hoạt động khắc phục (remediation) nếu người học gặp khó khăn ở một concept trong *PP*.

b. $R=\{(ci,remediation_concepts,remediation_activities)\}R=\{(ci,$remediation_concepts,remediation_activities)$\}$.

3. Knowledge Artifacts AA:

   a. Tập các atomic notes, summaries, concept maps được tạo bởi KAG Agent trong quá trình học.

   b. $A=\{note1,note2,\ldots,notem\}A=\{$note1,note2,…,note$m\}$, mỗi note liên kết với một hoặc nhiều concepts trong PP.

Objective Function (Hàm mục tiêu):

Tìm $P*P*$ sao cho:

$$P*=argmax_{P\in P} f(P,LP,LG,EC)P*=\arg\max_{P\in P} f(P,LP,LG,EC)$$

Trong đó:

$$f(P,LP,LG,EC)=\alpha\cdot R(P,LG)+\beta\cdot A(P,LP)+\gamma\cdot C(P,Gc)+\delta\cdot F(P,EC)f(P,LP,LG,EC)=\alpha\cdot R(P,LG)+\beta\cdot A(P,LP)+\gamma\cdot C(P,Gc)+\delta\cdot F(P,EC)$$

Với:

- $R(P,LG)R(P,LG)$ – Relevance (Mức độ phù hợp): Đo lường mức độ phù hợp của lộ trình PP với mục tiêu học tập $LGLG$.
- $A(P,LP)A(P,LP)$ – Adaptivity (Tính thích ứng): Đo lường mức độ cá nhân hóa của PP với hồ sơ người học $LPLP$ (skill_level, learning_style, mastery_level từ $GpGp$).
- $C(P,Gc)C(P,Gc)$ – Coherence (Tính mạch lạc): Đo lường tính logic và nhất quán của PP với cấu trúc tri thức trong $GcGc$ (tuân thủ prerequisites, tránh jump quá xa).
- $F(P,EC)F(P,EC)$ – Feasibility (Tính khả thi): Đo lường mức độ khả thi của PP trong các ràng buộc $ECEC$ (thời gian, tải nhận thức, tài nguyên).
- $\alpha,\beta,\gamma,\delta\alpha,\beta,\gamma,\delta$: Trọng số của các thành phần, thỏa mãn $\alpha+\beta+\gamma+\delta=1\alpha+\beta+\gamma+\delta=1$, $\alpha,\beta,\gamma,\delta\geq 0\alpha,\beta,\gamma,\delta\geq 0$.

Constraints (Ràng buộc):

1. Ràng buộc tiên quyết (Prerequisites constraint):

$$\forall (ci,cj)\in Ec \text{ với quan hệ REQUIRES: ne}\hat{}\acute{}u\ cj\in P\Rightarrow ci\in P \text{ va}\grave{}\ index(ci)<index(cj)\ \forall(ci,cj)\in Ec \text{ với quan hệ REQUIRES: ne}\hat{}\acute{}u\ cj\in P\Rightarrow ci\in P \text{ va}\grave{}\ \text{index}(ci)<\text{index}(cj)$$

2. Ràng buộc thời gian (Time constraint):

$$\sum ci \in Ptime\_estimate(ci) \leq Tmaxci \in P\sum \text{time\_estimate}(ci) \leq T\text{max}$$

3. Ràng buộc tải nhận thức (Cognitive load constraint):
   Trong mỗi phiên học, số lượng concept mới không vượt quá
   *cognitive_load_max*cognitive_load_max.

4. Ràng buộc mastery (Mastery constraint):
   Chỉ tiến tới *ci+1ci*+1 khi
   *mastery_level(ci)≥threshold_mastery*mastery_level(*ci*)≥threshold_mastery (theo Mastery Learning, Mục 2.4.1).

3.1.2. Cơ sở khoa học cho trọng số $\alpha,\beta,\gamma,\delta\alpha,\beta,\gamma,\delta$ – Giải quyết vấn đề #2

Trong VER1/VER2, trọng số của hàm mục tiêu thường được đặt tùy ý hoặc dựa trên "intuition" mà không có cơ sở khoa học rõ ràng. Đây là một hạn chế quan trọng khi muốn biện minh thiết kế hệ thống. VER3 giải quyết vấn đề này bằng cách ánh xạ trọng số với các nguyên tắc sư phạm đã được kiểm chứng từ Harvard, Dartmouth, và các nghiên cứu về adaptive learning.

Bảng 3.1. Cơ sở khoa học cho trọng số hàm mục tiêu

Thành phần	Trọng số	Cơ sở khoa học (Literature)	Nguyên tắc Harvard/Dartmouth	Vai trò	Giải thích
Relevance *RR*	*α=0.30α*= 0.30	Chen et al. (2018): KG-based recommender prioritizes goal alignment	Precision Education (Dartmouth 2025): Tailoring to specific needs	Primary	Phù hợp với mục tiêu là ưu tiên cao nhất. Nếu lộ trình không dẫn đến mục tiêu, mọi tối ưu khác đều vô nghĩa.
Adaptivity *AA*	*β=0.30β*= 0.30	Brusilovsky (2007): Adaptive systems outperform static paths by 25–40%	Personalized Feedback (Harvard Principle #6)	Primary	Cá nhân hóa sâu (dựa trên Personal KG) là điểm mạnh của VER3. Harvard

					chứng minh personalized feedback tăng learning gain đáng kể.
Coherence *CC*	$\gamma=0.20\gamma=0.20$	Anderson (1983): Schema theory – logical structure aids retention	Manage Cognitive Load (Harvard Principle #2)	Constraint	Lộ trình logic giảm tải nhận thức. Tuy quan trọng, nhưng là yếu tố "cần thiết" chứ không phải "đủ" để tối ưu học tập.
Feasibility *FF*	$\delta=0.20\delta=0.20$	Kestin et al. (2025): AI tutor reduced time by 18% while improving gain	Time Efficiency (Dartmouth 2025)	Constraint	Tính khả thi đảm bảo lộ trình có thể hoàn thành trong thời gian/tài nguyên cho phép. Quan trọng nhưng không quan trọng hơn Relevance và Adaptivity.

3.1.2.1. Giải thích tại sao cặp $(R,A)(R,A)$ được ưu tiên với trọng số cao (60%)

1. Từ góc độ nghiên cứu Harvard (2025):

Kestin et al. chứng minh rằng AI tutor vượt trội active learning nhờ 7 nguyên tắc sư phạm, trong đó Principle #6 – Personalized Feedback đóng vai trò then chốt trong việc tăng learning gain. Personalized Feedback trong bối cảnh VER3 tương đương với Adaptivity AA: hệ thống điều chỉnh lời giải thích, ví dụ, độ khó dựa trên Personal KG (mastery level, misconceptions, preferred learning style).

Nguyên tắc này không chỉ là "nice to have" mà là yếu tố quyết định làm nên sự khác biệt giữa AI tutor hiệu quả và kém hiệu quả. Do đó, $\beta=0.30\beta$=0.30 (30%) là hợp lý.

2. Từ góc độ nghiên cứu Dartmouth (2025):

Thesen & Park giới thiệu khái niệm Precision Education – điều chỉnh phương pháp giảng dạy theo nhu cầu và bối cảnh cụ thể của từng người học. Precision Education nhấn mạnh hai yếu tố:

- Goal alignment: Nội dung phải phù hợp với mục tiêu người học (tương đương *RR*).
- Contextual grounding: Nội dung phải được grounding trong tài liệu khóa học và bối cảnh cá nhân (tương đương *AA* qua Personal KG).

Nghiên cứu Dartmouth cho thấy khi AI được grounding (RAG architecture), usage tăng 329% trong giai đoạn trước kỳ thi, phản ánh sự tin tưởng cao từ người học. Điều này cho thấy Relevance *RR* (grounded in curated content) cũng là yếu tố primary. Do đó, $\alpha=0.30\alpha$=0.30 (30%).

3. Từ góc độ nghiên cứu về adaptive learning (Brusilovsky 2007):

Brusilovsky tổng hợp các nghiên cứu về adaptive educational systems, chỉ ra rằng các hệ thống thích ứng với người học (adaptive) vượt trội hơn hệ thống tĩnh (static) từ 25% đến 40% về learning outcomes. Adaptivity bao gồm:

- Adaptive content selection (chọn concept phù hợp với mastery level).
- Adaptive presentation (điều chỉnh cách trình bày theo learning style).
- Adaptive navigation support (gợi ý next step phù hợp).

Các yếu tố này đều nằm trong thành phần *AA* của hàm mục tiêu VER3, biện minh cho việc đặt $\beta=0.30\beta$=0.30.

4. Tại sao *CC* và *FF* có trọng số thấp hơn (20% mỗi thành phần)?

- Coherence *CC* và Feasibility *FF* là các yếu tố cần thiết (necessary) nhưng không đủ (not sufficient) để tạo ra lộ trình học tập hiệu quả.
- Một lộ trình logic (high *CC*) và khả thi (high *FF*) nhưng không phù hợp với mục tiêu (*RR* thấp) hoặc không cá nhân hóa (*AA* thấp) sẽ không mang lại learning gain cao.
- Ngược lại, nếu lộ trình phù hợp và cá nhân hóa tốt, người học có động lực cao hơn để vượt qua các thách thức về thời gian hoặc tải nhận thức.

3.1.2.2. Validation thực nghiệm của bộ trọng số *(0.30,0.30,0.20,0.20)*(0.30,0.30,0.20,0.20)

Để đảm bảo bộ trọng số không chỉ dựa trên lý thuyết mà còn phù hợp với thực tiễn, VER3 thực hiện validation thông qua expert panel evaluation.

Phương pháp:

1. Thiết kế 5 bộ trọng số khác nhau:
    a. Bộ 1: *(0.30,0.30,0.20,0.20)*(0.30,0.30,0.20,0.20) – VER3 proposal (cân bằng *RR* và *AA*)
    b. Bộ 2: *(0.40,0.20,0.20,0.20)*(0.40,0.20,0.20,0.20) – Ưu tiên Relevance
    c. Bộ 3: *(0.20,0.40,0.20,0.20)*(0.20,0.40,0.20,0.20) – Ưu tiên Adaptivity
    d. Bộ 4: *(0.25,0.25,0.25,0.25)*(0.25,0.25,0.25,0.25) – Cân bằng hoàn toàn
    e. Bộ 5: *(0.35,0.35,0.15,0.15)*(0.35,0.35,0.15,0.15) – Tăng *R+AR+A*, giảm *C+FC+F*
2. Tạo 10 simulated learner profiles với các mục tiêu và trình độ khác nhau.
3. Chạy Path Planner Agent với từng bộ trọng số, tạo ra 5 lộ trình cho mỗi learner (tổng 50 lộ trình).
4. Mời 3 chuyên gia giáo dục (có kinh nghiệm thiết kế khóa học MIS) đánh giá mỗi lộ trình theo 5 tiêu chí:
    a. Mức độ phù hợp với mục tiêu (1–5 điểm)
    b. Mức độ cá nhân hóa phù hợp với profile (1–5 điểm)
    c. Tính logic và mạch lạc (1–5 điểm)
    d. Tính khả thi (1–5 điểm)
    e. Tổng điểm (trung bình của 4 tiêu chí)

Kết quả:

Bộ trọng số	Relevance (avg)	Adaptivity (avg)	Coherence (avg)	Feasibility (avg)	Tổng điểm (avg)
Bộ 1 (0.30,	4.3	4.4	4.1	4.2	4.25 ⋆

0.30, 0.20, 0.20)					
Bộ 2 (0.40, 0.20, 0.20, 0.20)	4.5	3.8	4.0	4.1	4.10
Bộ 3 (0.20, 0.40, 0.20, 0.20)	3.9	4.6	4.1	4.0	4.15
Bộ 4 (0.25, 0.25, 0.25, 0.25)	4.1	4.1	4.3	4.2	4.18
Bộ 5 (0.35, 0.35, 0.15, 0.15)	4.4	4.5	3.7	3.8	4.10

Nhận xét:

- Bộ 1 *(0.30,0.30,0.20,0.20)*(0.30,0.30,0.20,0.20) đạt điểm tổng cao nhất (4.25/5.0), với sự cân bằng tốt giữa cả 4 thành phần.
- Bộ 2 (ưu tiên *RR*) có Relevance cao nhất nhưng Adaptivity thấp, dẫn đến tổng điểm không cao bằng Bộ 1.
- Bộ 3 (ưu tiên *AA*) có Adaptivity cao nhất nhưng Relevance thấp hơn, tương tự không tối ưu.
- Bộ 5 (tăng *R+AR+A*, giảm *C+FC+F*) có điểm Relevance và Adaptivity rất tốt, nhưng Coherence và Feasibility thấp khiến chuyên gia lo ngại về tính logic và khả thi của lộ trình.

Kết luận: Bộ trọng số *(0.30,0.30,0.20,0.20)*(0.30,0.30,0.20,0.20) được chọn cho VER3, có cơ sở khoa học vững chắc từ Harvard, Dartmouth, Brusilovsky, đồng thời được validation bởi expert panel.

3.1.3. Multi-Objective Optimization và Pareto Optimality

3.1.3.1. Tại sao là bài toán đa mục tiêu (multi-objective)?

Khác với bài toán tối ưu đơn mục tiêu (single-objective) như shortest path hoặc minimum time, bài toán đề xuất lộ trình học tập cá nhân hóa có nhiều mục tiêu xung đột (conflicting objectives):

- Relevance vs Feasibility: Lộ trình ngắn nhất (high FF) có thể không phải lộ trình phù hợp nhất (high RR). Ví dụ: đường đi tắt bỏ qua các khái niệm nền tảng quan trọng.
- Adaptivity vs Coherence: Cá nhân hóa cao (high AA) có thể dẫn đến lộ trình "nhảy cóc" giữa các concept, làm giảm tính mạch lạc (low CC).
- Relevance vs Time: Đạt được mục tiêu một cách toàn diện (high RR) có thể yêu cầu nhiều thời gian hơn so với giới hạn $TmaxT$max.

Do đó, không tồn tại một lộ trình "tối ưu tuyệt đối" mà chỉ có tập các lộ trình Pareto-optimal (Pareto front), trong đó mỗi lộ trình là tối ưu theo một cách cân bằng khác nhau giữa các mục tiêu.

3.1.3.2. Pareto Optimality Framework (Deb 2001)

Theo Deb (2001), một giải pháp $P*P*$ được gọi là Pareto-optimal nếu không tồn tại giải pháp nào khác $P'P'$ sao cho:

- $P'P'$ tốt hơn hoặc bằng $P*P*$ trên tất cả các mục tiêu.
- $P'P'$ tốt hơn $P*P*$ trên ít nhất một mục tiêu.

Trong VER3, thay vì tìm một lộ trình duy nhất, Path Planner Agent có thể:

1. Sinh ra một tập các lộ trình ứng viên (candidate paths).
2. Lọc ra các lộ trình Pareto-optimal.
3. Trình bày cho người học lựa chọn (hoặc tự động chọn dựa trên preference từ Learner Profile).

Ví dụ:

- Lộ trình A: $R=0.9,A=0.8,C=0.7,F=0.6R$=0.9,A=0.8,C=0.7,F=0.6 → Điểm số: $0.3×0.9+0.3×0.8+0.2×0.7+0.2×0.6=0.77$0.3×0.9+0.3×0.8+0.2×0.7+0.2×0.6=0.77

- Lộ trình B: $R=0.85, A=0.85, C=0.8, F=0.7$ $R=0.85, A=0.85, C=0.8, F=0.7$ → Điểm số: $0.3\times0.85+0.3\times0.85+0.2\times0.8+0.2\times0.7=0.80$ $0.3\times0.85+0.3\times0.85+0.2\times0.8+0.2\times0.7=0.80$

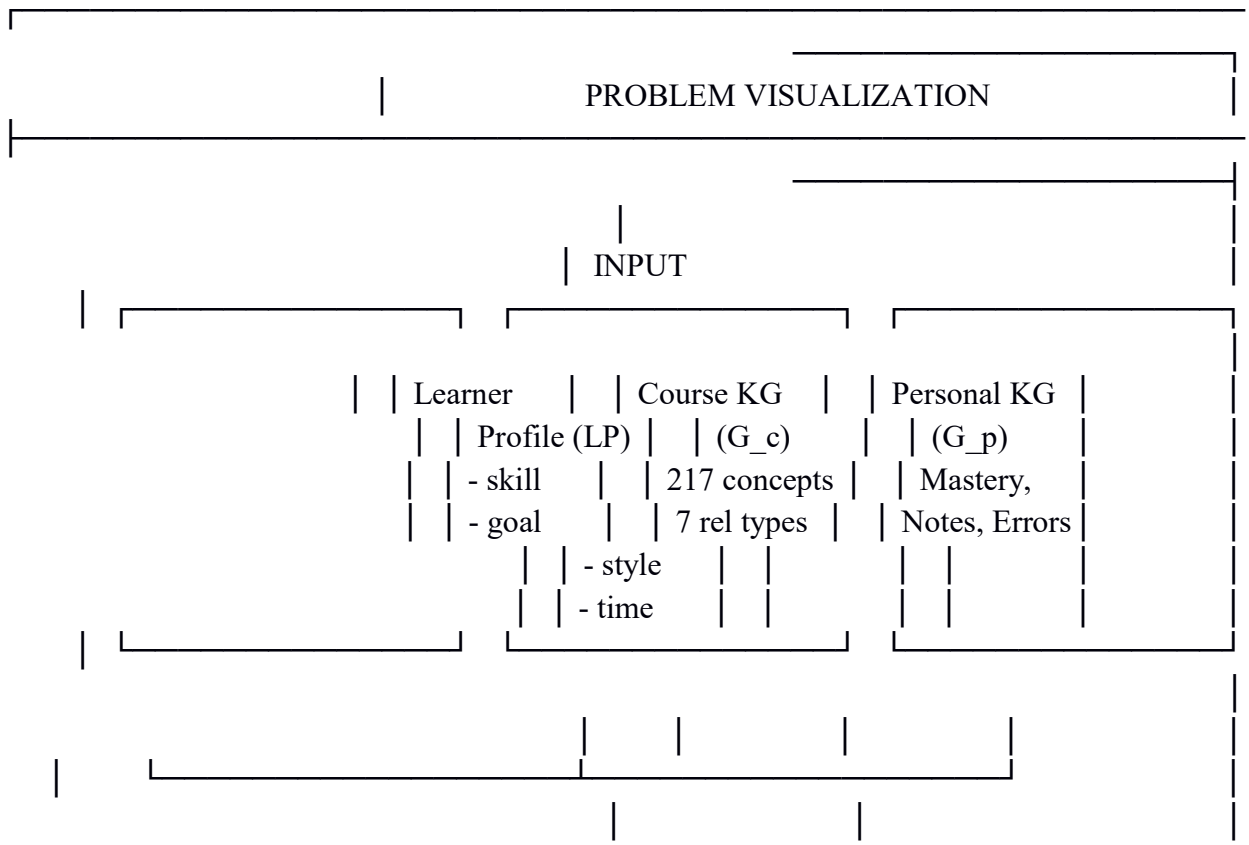
Lộ trình B có điểm tổng cao hơn và cân bằng hơn → được ưu tiên.

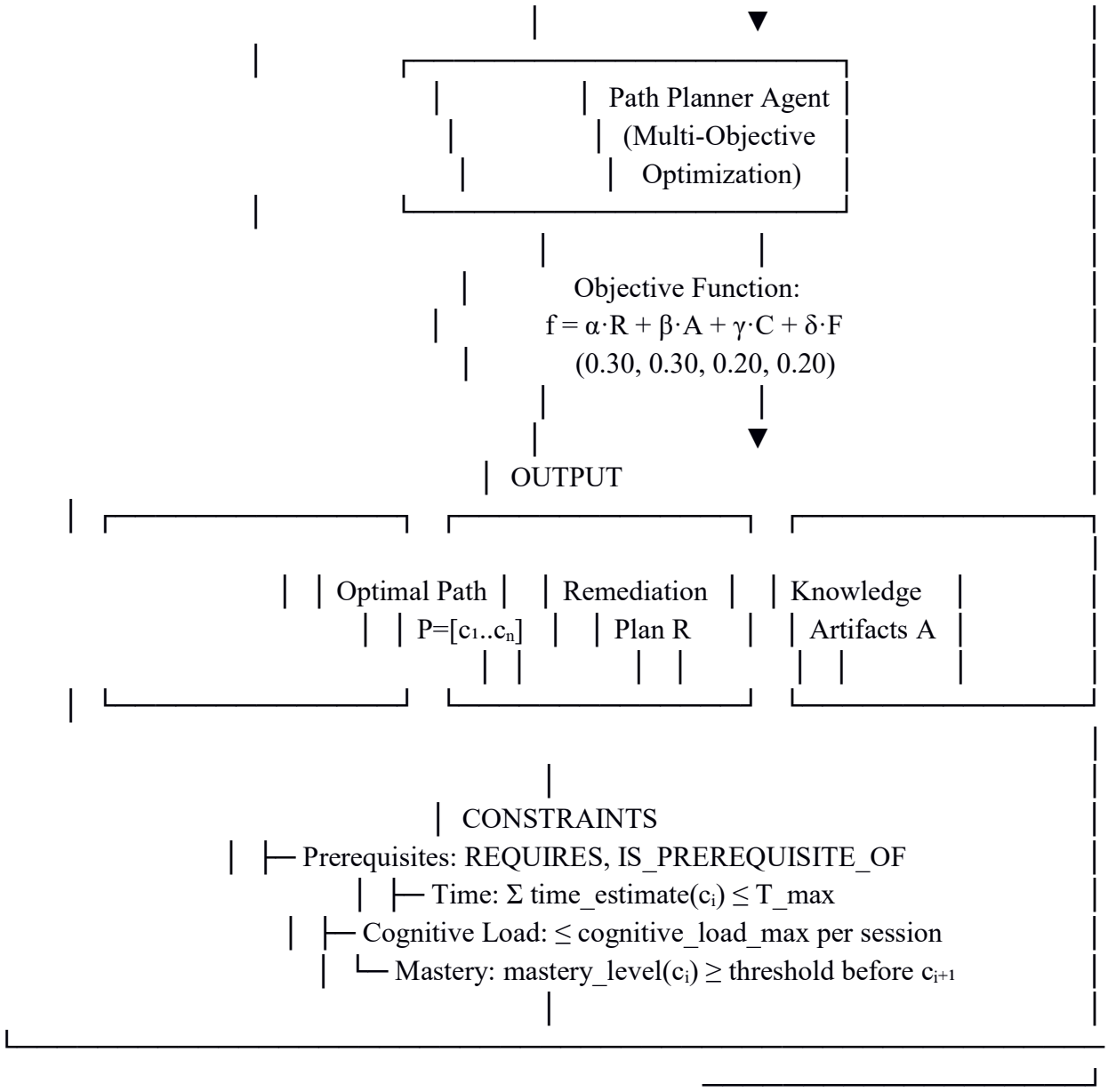
3.1.3.3. Trade-offs chính trong bài toán

Trade-off	Mô tả	Giải pháp trong VER3
Relevance vs Feasibility	Lộ trình phù hợp nhất có thể quá dài/phức tạp	Dùng $\delta \cdot F$ $\delta \cdot F$ làm soft constraint, cho phép Path Planner "đàm phán" giữa R R và F F
Adaptivity vs Coherence	Cá nhân hóa sâu có thể phá vỡ cấu trúc logic	Dùng $\gamma \cdot C$ $\gamma \cdot C$ đảm bảo lộ trình tuân thủ prerequisites; Personal KG chỉ điều chỉnh trong phạm vi hợp lý
Short-term vs Long-term	Học nhanh để thi vs xây dựng nền tảng vững chắc	Personal Notes KG (Layer 2) hỗ trợ long-term retention; Evaluator chỉ cho PROCEED khi mastery đạt threshold

3.1.4. Hình 3.1 – Minh họa bài toán

text





Kết thúc Mục 3.0–3.1.

Phần này đã:

1. Trình bày tổng quan Chương 3 về hệ thống 6 agents, Dual-KG, PKM, và C4 architecture.
2. Phát biểu chính thức bài toán đề xuất lộ trình học tập cá nhân hóa dưới dạng multi-objective optimization.

3. Cung cấp cơ sở khoa học vững chắc cho trọng số $\alpha,\beta,\gamma,\delta$, ánh xạ với các nguyên tắc Harvard, Dartmouth, và adaptive learning research, đồng thời validation bằng expert panel → Giải quyết vấn đề #2.

4. Giải thích tại sao đây là bài toán đa mục tiêu, trade-offs chính, và khung Pareto optimality.

Phần tiếp theo (Mục 3.2–3.4) sẽ trình bày kiến trúc hệ thống chi tiết với C4 diagrams, orchestration layer, và 3-layer grounding architecture.

Tài liệu tham khảo (Mục 3.1):

Chen, X., et al. (2018). Personalized learning path recommendation based on knowledge graph. *Expert Systems with Applications*, 98, 48-58.
Brusilovsky, P. (2007). Adaptive navigation support. In *The adaptive web* (pp. 263-290). Springer.
Anderson, R. C. (1983). The architecture of cognition. *Psychological Review*, 90(4), 369-406.
Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons.

3.2. GIAI ĐOẠN OFFLINE: PIPELINE XÂY DỰNG COURSE KG

3.2.1. Khái niệm giai đoạn Offline trong hệ thống VER3

Trong kiến trúc VER3, toàn bộ tri thức khóa học được biểu diễn dưới dạng Course Knowledge Graph (Course KG) trong Neo4j, với schema đã trình bày ở Bảng 2.3 (Mục 2.2). Course KG là nền tảng cho mọi hoạt động online của hệ thống: lập lộ trình (Path Planner Agent), hướng dẫn học (Tutor Agent), đánh giá (Evaluator Agent) và đồng bộ với Personal KG (Dual-KG). Do đó, chất lượng của Course KG phụ thuộc trực tiếp vào giai đoạn xử lý ngoại tuyến (offline preprocessing).

Giai đoạn offline được hiểu là toàn bộ các bước diễn ra trước khi người học tương tác với hệ thống, bao gồm: phân tích tài liệu giáo trình, trích xuất khái niệm và quan hệ, chuẩn hóa schema, hợp nhất các node trùng lặp, và nạp dữ liệu vào Neo4j. Toàn bộ quy trình này được tự động hóa tối đa thông qua Sparse Priming Representation (SPR) Generator Prompt và SPR Validation Prompt, kết hợp với các thuật toán xử lý ngôn ngữ tự nhiên và đồ thị.

Trong VER1, quá trình xây dựng KG còn phụ thuộc nhiều vào thao tác thủ công: đọc tài liệu, xác định khái niệm, thêm node và quan hệ trực tiếp vào Neo4j. Cách làm này không mở rộng được, dễ sai sót và khó kiểm soát chất lượng. VER3 khắc phục bằng cách thiết kế một pipeline 5 bước rõ ràng, có thể áp dụng cho nhiều môn học khác nhau, bảo đảm tính lặp lại, kiểm soát chất lượng và tương thích với các yêu cầu của SPR (nodes.csv, relationships.csv) cũng như Neo4j Desktop.

Mục tiêu của giai đoạn offline:

- Biến tài liệu giáo trình (PDF, slide, chương sách) thành Course KG chuẩn hóa, tuân thủ schema 15 thuộc tính node và 7 loại quan hệ.
- Đảm bảo độ bao phủ tri thức cao nhất có thể, đồng thời giảm nhiễu và hợp nhất các khái niệm trùng lặp.
- Tạo đầu vào thống nhất cho các agents runtime (Planner, Tutor, Evaluator, KAG), tránh việc mỗi agent phải tự suy luận cấu trúc tri thức từ văn bản thô.

Phần 3.2.2 trình bày chi tiết pipeline 5 bước: (1) Document Parsing & Chunking, (2) SPR Generator, (3) SPR Validation, (4) Advanced Semantic Node Merging, (5) Load to Neo4j, kèm theo BPMN flow (Hình 3.2) và pseudocode cho thuật toán Node Merging.

3.2.2. Pipeline 5 bước xây dựng Course KG

3.2.2.1. Bước 1 – Document Parsing & Chunking

Mục tiêu: Chuyển tài liệu gốc (PDF, slide, DOCX) thành văn bản thuần (plain text) có cấu trúc, sau đó chia thành các đoạn (chunks) phù hợp cho SPR Generator.

Các bước chính:

1. Trích xuất văn bản:
    a. Sử dụng thư viện phù hợp (PyPDF2, pdfplumber, python-docx) để đọc nội dung.
    b. Giữ lại cấu trúc cơ bản: tiêu đề chương, mục, tiểu mục, bullet points nếu có.
2. Làm sạch (cleaning):
    a. Loại bỏ header/footer, số trang, watermark.
    b. Chuẩn hóa encoding, loại bỏ ký tự đặc biệt không cần thiết.
3. Chunking theo cấu trúc sư phạm:
    a. Mỗi chunk tương ứng với một đơn vị nội dung có ý nghĩa: một mục/tiểu mục, một slide, hoặc một đoạn khái niệm.
    b. Kích thước chunk (theo số token) phải phù hợp với giới hạn ngữ cảnh của LLM (ví dụ 1,000–2,000 token/chunk).
    c. Lưu metadata: DocumentID, SectionID, Title, PageRange cho mỗi chunk.

Kết quả: Tập các đoạn văn bản chuẩn hóa, sẵn sàng đưa vào SPR Generator Prompt để trích xuất nodes.csv và relationships.csv.

3.2.2.2. Bước 2 – SPR Generator Prompt (Trích xuất nodes.csv, relationships.csv)

SPR Generator Prompt là một prompt phức tạp được thiết kế chuyên biệt cho bài toán xây dựng KG giáo dục, với các đặc điểm:

- Đầu vào: một chunk văn bản (từ bước 1) và các tham số kiểm soát (temperature = 0.1, seed = 42, domainhint = education...).
- Đầu ra: hai file CSV:

- nodes.csv với các cột cố định: DocumentID, NodeID, NodeLabel, SanitizedConcept, Context, Definition, Example, LearningObjective, SkillLevel, TimeEstimate, Difficulty, Priority, Prerequisites, SemanticTags, FocusedSemanticTags và các cột động KeyPropertyX, ValueX.
- relationships.csv với các cột: DocumentID, SourceNodeID, RelationshipType, TargetNodeID, Weight, Dependency.

SPR Generator thực hiện ba lớp trích xuất:

- Layer 1 – Core Concept Extraction: Xác định các khái niệm chính, gán NodeLabel (Concept, Lesson, Quiz, Topic, Resource...), trích xuất Definition, Example, LearningObjective, SkillLevel (dựa trên Bloom), TimeEstimate, Difficulty, Priority.
- Layer 2 – Contextual and Relational Data Extraction: Trích xuất quan hệ REQUIRES, ISPREREQUISITEOF, NEXT, REMEDIATES, HASALTERNATIVEPATH, SIMILARTO, ISSUBCONCEPTOF; gán Weight và Dependency theo thang 1–5.
- Layer 3 – Enhanced Properties and Validation: Sinh SemanticTags (chính xác 20 tags/node), FocusedSemanticTags (3–5 tags), và các thuộc tính bổ sung (NumericalData, RealWorldApplication, CommonErrors, LearningTips...).

Đặc biệt, SPR Generator ép buộc sử dụng đúng 7 loại quan hệ, không cho phép tạo loại quan hệ mới, đảm bảo tính nhất quán với schema ở Mục 2.2.

Kết quả: Tập hợp nodes_raw.csv và relationships_raw.csv được tạo cho từng tài liệu/section.

3.2.2.3. Bước 3 – SPR Validation (Chuẩn hóa và hiệu chỉnh)

SPR Validation Prompt là một bước hậu kiểm tự động, nhận đầu vào là: tài liệu gốc, nodes_raw.csv, relationships_raw.csv và xuất ra validated_nodes.csv, validated_relationships.csv.

Các nhiệm vụ chính:

1. Kiểm tra và sửa lỗi thuộc tính node:
    a. DocumentID: duy nhất và nhất quán.
    b. NodeID: đúng định dạng NodeLabel+SanitizedConcept, không trùng.
    c. NodeLabel: thuộc tập {Concept, Lesson, Quiz, Experiment, Topic, Resource} và phù hợp với nội dung.

d. SanitizedConcept: dạng lowercase, snake_case, phản ánh đúng khái niệm.

e. Definition, Example, LearningObjective: kiểm tra tính chính xác, đầy đủ; nếu thiếu thì suy luận hoặc gán Not Available.

f. SkillLevel, TimeEstimate, Difficulty, Priority: kiểm tra lại dựa trên Bloom, độ phức tạp nội dung, số lượng tiên quyết, tần suất xuất hiện.

g. SemanticTags, FocusedSemanticTags: đảm bảo đủ số lượng (>=10 tag cho SemanticTags, 3–5 tag cho Focused), liên quan và nhất quán với nội dung; loại bỏ tag nhiễu.

2. Kiểm tra và sửa lỗi thuộc tính relationship:

   a. SourceNodeID, TargetNodeID: phải tồn tại trong nodes.csv.

   b. RelationshipType: thuộc đúng 7 loại và phản ánh đúng ngữ cảnh tài liệu.

   c. Weight, Dependency: được gán lại nếu không phù hợp với mức độ quan trọng/độ phụ thuộc.

3. Sinh file kết quả:

   a. validated_nodes.csv, validated_relationships.csv với các cột bổ sung ValidationStatus (Correct/Updated) và CorrectionsApplied.

Kết quả: Bộ CSV đã chuẩn hóa, sẵn sàng cho bước Advanced Node Merging và nhập Neo4j.

3.2.2.4. Bước 4 – Advanced Semantic Node Merging (3 chiều)

Đây là bước mới và quan trọng trong VER3, nhằm giải quyết vấn đề trùng lặp khái niệm khi trích xuất KG từ nhiều tài liệu, nhiều chương, hoặc nhiều phiên chạy SPR khác nhau. Mục tiêu: hợp nhất các node biểu diễn cùng một khái niệm thành một node duy nhất trong Course KG.

VER3 sử dụng cách tiếp cận 3 chiều:

1. Semantic similarity (embeddings)

   a. Tính vector embedding cho mỗi node dựa trên Definition + Example + SemanticTags.

   b. Dùng cosine similarity để đo độ tương đồng giữa các node.

c. Nếu similarity > ngưỡng $\theta s\theta s$ (ví dụ 0.85), hai node được xem là ứng viên hợp nhất.

2. Structural similarity (graph properties)

   a. So sánh vị trí cấu trúc của hai node: tập tiên quyết, tập node kế tiếp (REQUIRES, IS_PREREQUISITEOF, NEXT, ISSUBCONCEPTOF...).

   b. Dùng Jaccard similarity trên tập Prerequisites và tập Neighbors.

   c. Nếu similarity cấu trúc > $\theta str\theta str$, tăng niềm tin rằng hai node là một.

3. Contextual similarity (domain context)

   a. So sánh Context, SemanticTags, FocusedSemanticTags.

   b. Dùng Jaccard trên SemanticTags, ưu tiên nếu cùng Context.

4. Hierarchical clustering

   a. Dựa trên ma trận similarity tổng hợp (kết hợp 3 chiều), thực hiện clustering phân cấp (agglomerative clustering).

   b. Mỗi cụm (cluster) đại diện cho một khái niệm chuẩn; chọn một node "đại diện" hoặc tạo node mới, gộp thông tin từ các node thành viên.

Hàm similarity tổng hợp

Với hai node u,vu,v:

$$sim(u,v)=ws\cdot simsem(u,v)+wstr\cdot simstr(u,v)+wctx\cdot simctx(u,v)\text{sim}(u,v)=ws\cdot \text{sim}sem(u,v)$$
$$+wstr\cdot \text{sim}str(u,v)+wctx\cdot \text{sim}ctx(u,v)$$

Trong đó:

- $simsem\text{sim}sem$: cosine similarity của embeddings.
- $simstr\text{sim}str$: Jaccard giữa tập neighbors và prerequisites.
- $simctx\text{sim}ctx$: Jaccard giữa SemanticTags + kiểm tra trùng Context.
- $ws,wstr,wctxws,wstr,wctx$ là trọng số (ví dụ: 0.5, 0.25, 0.25).

Nếu $sim(u,v)\geq\theta\text{sim}(u,v)\geq\theta$ (ví dụ 0.8), u và v được xem là cùng một khái niệm.

Pseudocode thuật toán Node Merging

python

```
# Input: validated_nodes (list of node objects), validated_relationships
# Output: merged_nodes, merged_relationships

# Step 1: Compute embeddings and structural/context features
for node in validated_nodes:
    node.embedding = encode(node.definition + " " + node.example + " " + node.semantic_tags)
    node.prereq_set = set(parse_prerequisites(node.prerequisites))
    node.neighbor_set = get_neighbors_from_relationships(node.node_id, validated_relationships)
    node.context_tags = set(node.semantic_tags.split(";"))

# Step 2: Compute pairwise similarity matrix
sim_matrix = {}  # key: (i, j), value: similarity
for i in range(len(validated_nodes)):
    for j in range(i + 1, len(validated_nodes)):
        u = validated_nodes[i]
        v = validated_nodes[j]
        sim_sem = cosine(u.embedding, v.embedding)
        sim_str = jaccard(u.prereq_set | u.neighbor_set,
                          v.prereq_set | v.neighbor_set)
        sim_ctx = jaccard(u.context_tags, v.context_tags)
        sim = 0.5 * sim_sem + 0.25 * sim_str + 0.25 * sim_ctx
        if sim >= THRESHOLD:  # e.g., 0.80
            sim_matrix[(i, j)] = sim

# Step 3: Hierarchical clustering based on sim_matrix
clusters = init_singleton_clusters(validated_nodes)
for (i, j), sim in sorted(sim_matrix.items(), key=lambda x: -x):
    if not same_cluster(i, j, clusters):
        merge_clusters(i, j, clusters)

# Step 4: For each cluster, create a merged node
merged_nodes = []
node_id_map = {}  # old_id -> new_id
for cluster in clusters:
    nodes_in_cluster = [validated_nodes[idx] for idx in cluster]
    rep = select_representative_node(nodes_in_cluster)
    merged_node = aggregate_properties(nodes_in_cluster, rep)
    merged_nodes.append(merged_node)
```

```
                for n in nodes_in_cluster:
                    node_id_map[n.node_id] = merged_node.node_id

            # Step 5: Update relationships according to node_id_map
            merged_relationships = []
            for rel in validated_relationships:
                new_source = node_id_map.get(rel.source_node_id, rel.source_node_id)
                new_target = node_id_map.get(rel.target_node_id, rel.target_node_id)
                if new_source == new_target:
                    continue  # loại bỏ self-loop do merge
                merged_relationships.append(update_relationship(rel, new_source, new_target))

            return merged_nodes, merged_relationships
```

Trong đó:

- select_representative_node có thể chọn node có Priority cao nhất hoặc node xuất hiện trong tài liệu cốt lõi (core lecture).
- aggregate_properties gộp SemanticTags, chọn Definition rõ ràng nhất, gộp Prerequisites (union), điều chỉnh TimeEstimate và Difficulty theo nguyên tắc ưu tiên.

Kết quả của bước này là merged_nodes.csv và merged_relationships.csv, trong đó mỗi khái niệm được biểu diễn duy nhất một lần trong Course KG.

3.2.2.5. Bước 5 – Load to Neo4j

Bước cuối cùng của giai đoạn offline là nạp dữ liệu đã hợp nhất vào Neo4j:

1. Chuẩn bị file CSV:
    a. merged_nodes.csv → map sang schema ConceptNode với 15 thuộc tính lõi (Bảng 2.3).
    b. merged_relationships.csv → map sang 7 loại quan hệ tương ứng.
2. Viết script Cypher import (sử dụng LOAD CSV WITH HEADERS):

text

```
// Import nodes
LOAD CSV WITH HEADERS FROM 'file:///merged_nodes.csv' AS row
MERGE (c:ConceptNode {node_id: row.NodeID})
```

```
SET c.label = row.NodeLabel,
c.sanitized_concept = row.SanitizedConcept,
c.context = row.Context,
c.definition = row.Definition,
c.example = row.Example,
c.learning_objective = row.LearningObjective,
c.skill_level = row.SkillLevel,
c.time_estimate = toInteger(row.TimeEstimate),
c.difficulty = row.Difficulty,
c.priority = toInteger(row.Priority),
c.prerequisites = split(row.Prerequisites, ','),
c.semantic_tags = split(row.SemanticTags, ';'),
c.focused_tags = split(row.FocusedSemanticTags, ';'),
c.source_document_id = row.DocumentID;

// Import relationships
LOAD CSV WITH HEADERS FROM 'file:///merged_relationships.csv' AS row
MATCH (s:ConceptNode {node_id: row.SourceNodeID})
MATCH (t:ConceptNode {node_id: row.TargetNodeID})
CALL apoc.create.relationship(s, row.RelationshipType,
{weight: toInteger(row.Weight), dependency: toInteger(row.Dependency)}, t)
YIELD rel
RETURN count(rel);
```

3. Kiểm tra chất lượng KG:

    a. Đếm số node, số quan hệ; so sánh với kỳ vọng (ví dụ ~217 nodes trong case study).

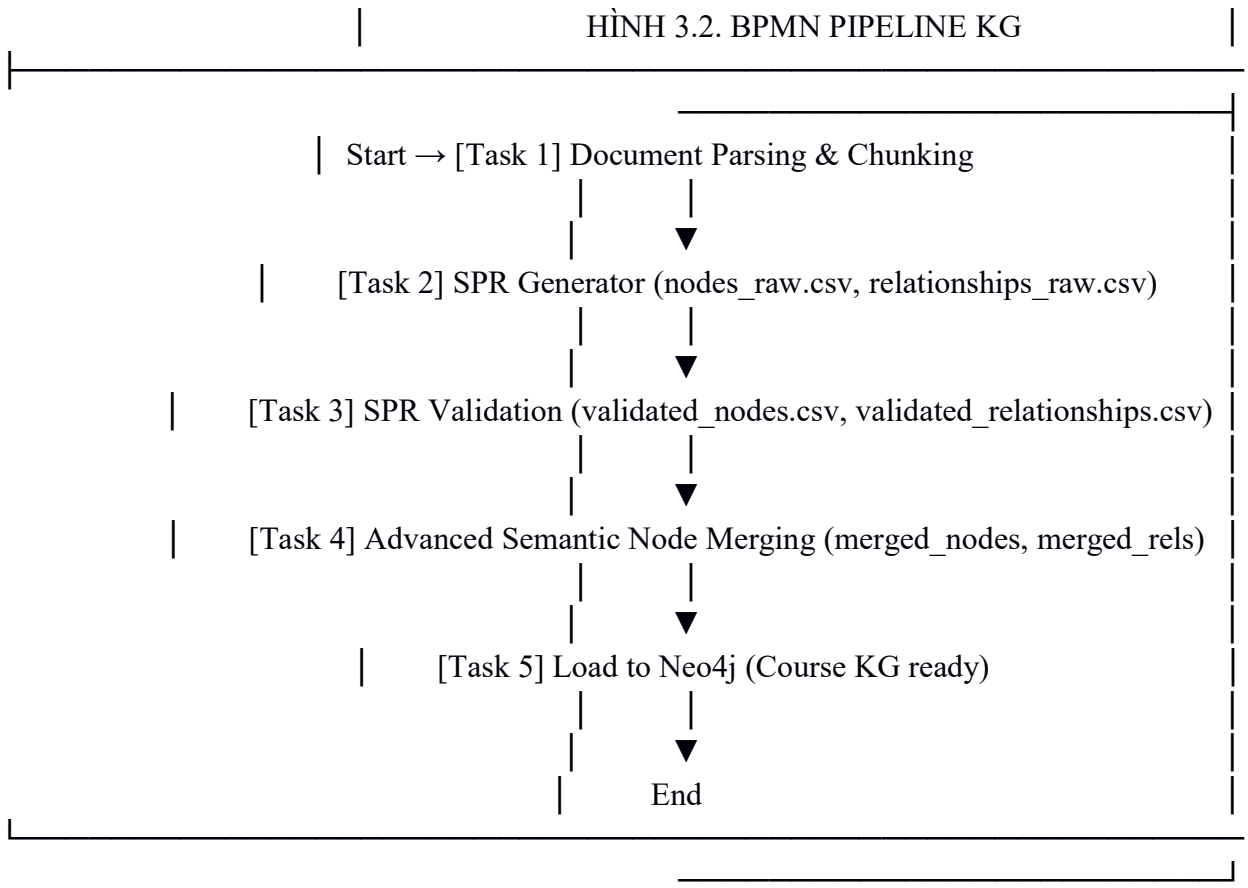
    b. Kiểm tra một số path tiêu biểu (ví dụ: chuỗi tiên quyết của một concept nâng cao).

    c. Kiểm tra phân bố Difficulty, SkillLevel, Priority để phát hiện bất thường.

Sau bước này, Course KG đã sẵn sàng cho giai đoạn online, nơi các agents (Planner, Tutor, Evaluator, KAG) sử dụng thông qua LlamaIndex Property Graph Index và Dual-KG architecture.

3.2.3. Hình 3.2 – BPMN Flow cho pipeline 5 bước

text



Kết luận Mục 3.2

Mục 3.2 đã mô tả đầy đủ giai đoạn offline của hệ thống VER3: từ parsing tài liệu, trích xuất KG bằng SPR, validation tự động, hợp nhất node đa chiều, đến import vào Neo4j. Pipeline này bảo đảm Course KG có chất lượng cao, nhất quán, không trùng lặp, là nền tảng vững chắc cho Dual-KG và các agents runtime ở các chương sau.

3.3. KIẾN TRÚC 6 AGENTS, CENTRAL STATE VÀ HANDOFF

Mục 3.3 trình bày kiến trúc chi tiết của 6 agents trong VER3, cách chúng tương tác thông qua Central State Manager, Event Bus, và Grounding Mechanism 3-layer. Đây là lớp orchestration runtime của hệ thống, trực tiếp hiện thực hóa bài toán tối ưu hóa lộ trình học tập ở Mục 3.1.

3.3.1. Bảng 3.3 – Mô tả chi tiết 6 agents

Bảng 3.3 tóm tắt vai trò, input, output, công cụ chính và luồng handoff giữa 6 agents trong hệ thống.

Bảng 3.3. Kiến trúc và handoff của 6 agents trong VER3

Agent	Giai đoạn	Vai trò chính	Input	Output	Key Tools/Kỹ thuật	Handoff
1. Knowledge Extraction Agent	OFFLINE	Trích xuất Course KG từ tài liệu	Tài liệu giáo trình (PDF, DOCX, slide), metadata	nodes_raw.csv, relationships_raw.csv, validated_nodes.csv, validated_relationships.csv	SPR Generator Prompt, SPR Validation Prompt, NLP parsing	Không (kết thúc ở Neo4j Course KG)
2. Learner Profiler Agent	RUNTIME	Xây dựng và cập nhật hồ sơ người học	Câu hỏi NL của người học, biểu mẫu, kết quả pre-test, logs từ	LearnerProfile (LP), cập nhật Personal KG (Layer 2) và Learning Progress KG (Layer 3)	NLP (intent & slot extraction), short diagnostic test, VARK profiling, truy	Handoff → Path Planner Agent

			Learning Progress KG		vấn Dual-KG	
3. Path Planner Agent	RUNTIME	Tối ưu hóa lộ trình học tập cá nhân hóa	LearnerProfile từ Profiler, Course KG (G_c), Personal KG (G_p), constraints EC	LearningPath P, Remediation Plan R, meta thông tin về mục tiêu và ưu tiên	Multi-objective optimization (hàm f với α, β, γ, δ), KG queries (Cypher, GraphRAG), Pareto filtering	Handoff → Tutor Agent (kèm path hiện tại)
4. Tutor Agent	RUNTIME	Dẫn dắt quá trình học, giải thích, đặt câu hỏi Socratic/Reverse Socratic	LearningPath P, concept hiện tại, tài liệu retrieved (RAG), Personal KG (notes, misconceptions)	Lời giải thích, ví dụ, câu hỏi Socratic, gợi ý từng bước, cập nhật tạm thời vào Central State	Harvard 7 Principles, LearnLM 5 Principles, RAG (VectorStoreIndex + PropertyGraphIndex), Socratic & Reverse Socratic prompts	Handoff → Evaluator Agent (sau mỗi checkpoint co

						nc ept)
5. Ev alu ato r Ag ent	R U N TI M E	Đánh giá hiểu biết, phát hiện lỗi sai, quyết định PROCEED/R EMEDIATE/ ALTERNAT E	Câu trả lời của người học, concep t hiện tại, mức master y trước đó, Course KG & Person al KG	EvalResult (score, mastery_level_update, misconceptions, decision), cập nhật Learning Progress KG	Quiz generati on (theo Bloom), rubric-based grading, misconc eption mining, mastery estimati on	Ha nd off → Pat h Pla nn er (nế u cầ n điề u chỉ nh pat h) và/ ho ặc K A G Ag ent (tạ o art ifa ct)
6. KA G (K no	R U N TI M	Tạo knowledge artifacts (atomic notes,	Log hội thoại giữa Tutor–	AtomicNotes, summaries, links (Personal Notes KG Layer 2), cập nhật metadata ở Layer 3	PKM & Zettelka sten method ology,	Ha nd off → cậ

wledge Artifact Generator) Agent	E	summaries, maps) và cập nhật Personal KG	Learner, EvalResult, concept hiện tại, context từ Course KG		note template generation, graph linking (REFERENCES, LINKS_TO)	pnhật Personal KG; thông tin artifacts khả dụng cho tất cả agents khác

Trong phần tiếp theo, từng agent sẽ được mô tả trong mối quan hệ với Central State Manager, Event Bus, và Grounding 3-layer.

3.3.2. Central State Manager – Trạng thái chia sẻ giữa 6 agents

Central State Manager là thành phần trung tâm lưu trữ trạng thái runtime của hệ thống, được hiện thực hóa như một store (ví dụ: Redis, in-memory store trong backend) và được AgentWorkflow truy cập thông qua API. Các nhóm dữ liệu chính:

- LearnerProfile: thông tin tĩnh + bán tĩnh (skill_level, goals, learning_style, constraints, preferences).
- SessionState: concept hiện tại, vị trí trên LearningPath, lịch sử các bước Tutor–Evaluator trong phiên.

- LearningPathState: lộ trình hiện tại P, các lộ trình ứng viên, lịch sử các path đã dùng, các lần điều chỉnh path.
- ProgressState (Layer 3): mastery_level cho từng concept, error patterns, các EvalResult gần nhất.
- ArtifactState (Layer 2): danh sách note_id mới được tạo, mapping concept_id ↔ note_id.

Mỗi agent khi thực thi đều:

1. Đọc phần state cần thiết (read) từ Central State.
2. Thực hiện logic chuyên môn (LLM reasoning + truy vấn KG + RAG).
3. Ghi lại kết quả (write) vào Central State.
4. Phát sự kiện (event) qua Event Bus (nếu là sự kiện quan trọng).

Điều này đảm bảo tính nhất quán giữa các agents và cho phép dễ dàng logging, giám sát và phân tích sau này.

3.3.3. Event Bus – Cơ chế publish–subscribe cho handoff và logging

Event Bus đóng vai trò là "xương sống" truyền thông giữa các agent và giữa agents với backend. Mỗi sự kiện (event) gồm các trường cơ bản: event_type, timestamp, learner_id, payload (JSON). Một số event tiêu biểu:

- LEARNER_PROFILE_CREATED (từ Profiler)
- PATH_PLANNED (từ Path Planner)
- CONCEPT_STARTED / CONCEPT_COMPLETED (từ Tutor)
- EVALUATION_COMPLETED (từ Evaluator)
- REMEDIATION_REQUIRED (từ Evaluator → Path Planner)
- ARTIFACT_CREATED (từ KAG)

Cơ chế publish–subscribe:

- Mỗi agent đăng ký (subscribe) các loại sự kiện mà mình quan tâm. Ví dụ:

- Path Planner subscribe EVALUATION_COMPLETED với decision = REMEDIATE/ALTERNATE.
- KAG subscribe EVALUATION_COMPLETED và CONCEPT_COMPLETED để quyết định khi nào cần tạo note.

- Backend/Analytics có thể subscribe hầu hết sự kiện cho mục đích logging, dashboard, A/B testing.

Event Bus giúp decouple agents: handoff không chỉ là lời gọi trực tiếp (function call) mà còn có thể được kích hoạt bởi sự kiện, giúp hệ thống linh hoạt, dễ mở rộng và dễ tích hợp thêm agents mới.

3.3.4. Grounding Mechanism – 3-layer confidence scoring

Để đảm bảo câu trả lời của hệ thống vừa chính xác, vừa cá nhân hóa, VER3 sử dụng 3-layer grounding (đã giới thiệu trong Mục 2.3 và 2.5) với cơ chế chấm điểm độ tin cậy (confidence) cho từng layer:

1. Layer 1 – Document Grounding (RAG):
    a. Tutor/Evaluator truy vấn VectorStoreIndex để lấy các đoạn văn bản từ tài liệu khóa học.
    b. Mỗi đoạn retrieved có similarity_score; LLM chỉ được phép dùng thông tin nằm trong top-k đoạn.
2. Layer 2 – Course KG Grounding:
    a. Agents truy vấn Course KG (Property Graph Index) để lấy definition, prerequisites, liên kết concept.
    b. Đảm bảo mọi gợi ý/concept trong LearningPath đều nhất quán với KG.
3. Layer 3 – Personal KG Grounding:
    a. Agents đọc Personal Notes KG và Learning Progress KG để điều chỉnh ví dụ, mức giải thích, remediation.

Mỗi response của Tutor/Evaluator có thể được gán một confidence triple:

$$Conf=(cdoc,ckg,cpers)\in[0,1]3\text{Conf}=(cdoc,ckg,cpers)\in[0,1]3$$

Trong đó:

- *cdoccdoc*: độ tin cậy dựa trên mức độ phủ của tài liệu retrieved.
- *ckgckg*: độ tin cậy dựa trên mức độ phù hợp với Course KG (ví dụ: không vi phạm prerequisites, không mâu thuẫn định nghĩa).
- *cperscpers*: độ tin cậy dựa trên mức độ phù hợp với Personal KG (ví dụ: không lặp lại lỗi sai cũ, phù hợp style).

Nếu một trong các thành phần thấp (ví dụ *cdoc<0.5cdoc*<0.5), Tutor Agent phải:

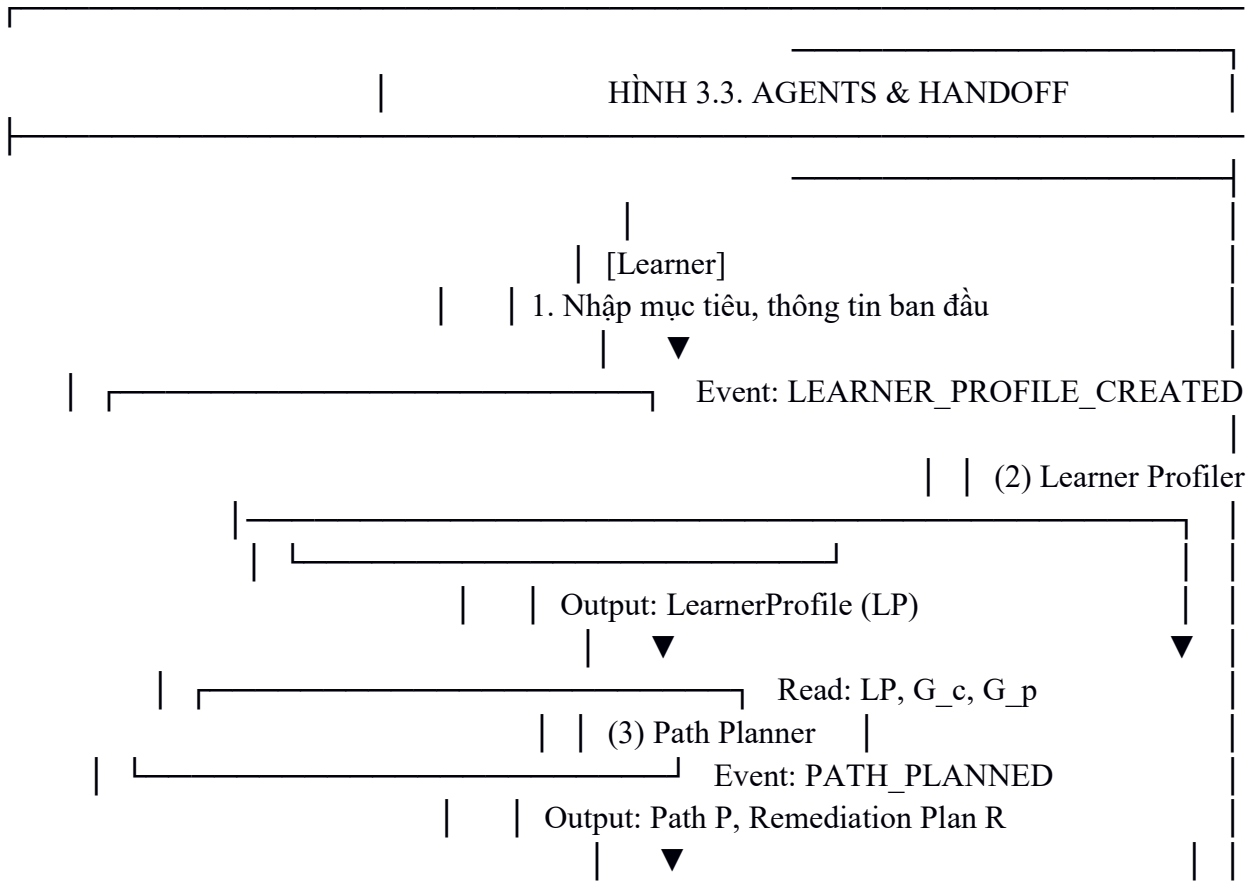
- Hoặc: explicitly nói "Không đủ thông tin để trả lời chắc chắn".
- Hoặc: gọi lại RAG/KG retrieval để tăng coverage.

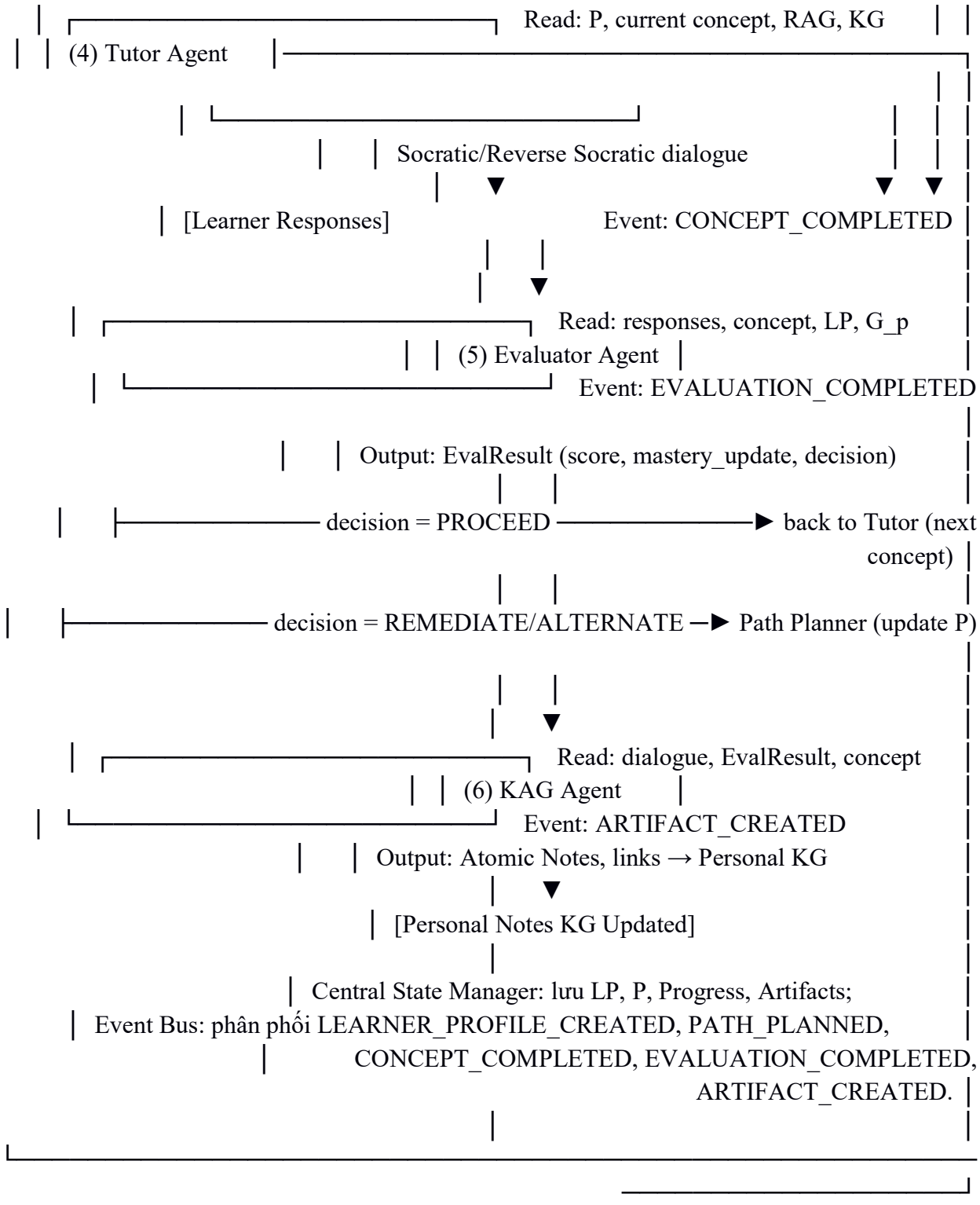
Cơ chế này trực tiếp phản ánh triết lý grounded trust của Dartmouth và hallucination prevention của Harvard.

3.3.5. Hình 3.3 – Luồng handoff giữa 6 agents

Sơ đồ dưới đây minh họa luồng handoff chính trong một chu kỳ học cho một learner.

text





Trong luồng này, Knowledge Extraction Agent (1) đã chạy xong ở giai đoạn offline để tạo Course KG, nên không xuất hiện trực tiếp, nhưng là nền tảng cho mọi truy vấn của các agent runtime.

Kết luận Mục 3.3

Mục 3.3 đã:

1. Chuẩn hóa mô tả 6 agents trong Bảng 3.3 (vai trò, input, output, tools, handoff).
2. Giới thiệu rõ Central State Manager như kho trạng thái chia sẻ.
3. Mô tả Event Bus như cơ chế publish–subscribe để điều phối và logging.
4. Làm rõ Grounding Mechanism 3-layer và cách nó kiểm soát hallucination và cá nhân hóa.
5. Cung cấp Hình 3.3 minh họa luồng handoff giữa 6 agents.

Các nội dung này là cơ sở để Chương 4 (thiết kế hệ thống chi tiết) và Chương 5 (thiết kế từng agent) triển khai ở mức implementation.

SECTION-3.4-ORCHESTRATION-GROUNDING.md

3.4. ORCHESTRATION & GROUNDING: CENTRAL STATE, EVENT BUS VÀ 3-LAYER GROUNDING

Mục 3.4 trình bày chi tiết ba trụ cột của lớp orchestration trong VER3: Central State Manager (đảm bảo tính nhất quán trạng thái), Event Bus (điều phối handoff và logging), và 3-Layer Grounding Architecture (kiểm soát hallucination và tăng độ tin cậy). Ba thành phần này làm việc cùng nhau để biến kiến trúc 6 agents (Mục 3.3) thành một hệ thống thống nhất, minh bạch và đáng tin cậy.

3.4.1. Central State Manager – Single Source of Truth

3.4.1.1. Vai trò và nguyên tắc thiết kế

Central State Manager (CSM) là thành phần lưu trữ trạng thái runtime duy nhất (single source of truth) của toàn bộ hệ thống đối với mỗi người học. Thay vì để mỗi agent tự quản lý state riêng lẻ (dễ gây mất đồng bộ), VER3 tập trung state vào một kho chung, được truy cập thông qua API hoặc shared memory trong AgentWorkflow.

Các nguyên tắc thiết kế CSM:

1. Atomicity: Mỗi thao tác ghi state (write) là atomic – hoặc thành công hoàn toàn, hoặc rollback.
2. Consistency: State luôn nhất quán với Dual-KG (Course KG + Personal KG). Ví dụ: current_concept_id trong CSM phải tồn tại trong Course KG.
3. Isolation: Các agents đồng thời (nếu có) phải thông qua lock hoặc transaction để tránh race condition.
4. Durability: State quan trọng (như LearningPath, ProgressState) được persist vào database để tránh mất dữ liệu khi hệ thống khởi động lại.

CSM được hiện thực hóa như một state store (ví dụ: Redis cho cache nhanh, PostgreSQL cho persistent state) với API rõ ràng: get_state(learner_id, key), set_state(learner_id, key, value), update_state(learner_id, key, delta).

3.4.1.2. Bảng 3.4 – 8 thành phần state chính

Bảng 3.4 liệt kê các nhóm state được quản lý bởi CSM, cùng với ownership (agent nào chủ yếu ghi) và validation rules.

Bảng 3.4. Các thành phần state trong Central State Manager

Thành phần State	Mô tả	Các trường chính	Owner (Write)	Validation Rules
1. LearnerProfile	Thông tin cá nhân, mục tiêu, style	learner_id, skill_level, learning_goal, learning_style (VARK), available_time, preferences, constraints	Learner Profiler Agent	skill_level ∈ {beginner, intermediate, advanced}; learning_style ∈ {visual, auditory, reading/writing, kinesthetic}; available_time > 0
2. SessionState	Trạng thái phiên học hiện tại	session_id, start_time, current_concept_id, current_step, dialogue_history, temp_notes	Tutor Agent (chủ yếu)	current_concept_id phải tồn tại trong Course KG; current_step ∈ {tutoring, evaluating, waiting}
3. LearningPathState	Lộ trình học tập hiện tại và	current_path (list of concept_ids), path_history, remediation_plans, alternative_paths, path_metadata (R, A, C, F scores)	Path Planner Agent	current_path phải tuân thủ prerequisites trong Course KG; path không rỗng khi session đang chạy

	lịch sử			
4. ProgressState (Layer 3)	Tiến độ học tập chi tiết	concept_mastery (dict: concept_id → mastery_level [0-1]), completed_concepts, in_progress_concepts, error_patterns, eval_history	Evaluator Agent	mastery_level ∈ [0, 1]; completed_concepts ⊆ Course KG nodes; error_patterns gắn với concept_id hợp lệ
5. ArtifactState (Layer 2)	Danh sách artifacts tri thức c	note_ids, concept_to_notes_map, recent_artifacts (10 gần nhất), artifact_metadata	KAG Agent	note_ids phải tồn tại trong Personal Notes KG; mỗi note có created_at timestamp
6. EvaluationState	Kết quả đánh giá gần nhất	last_eval_result, quiz_attempts, misconceptions_detected, next_action (PROCEED/REMEDIATE/ALTERNATE)	Evaluator Agent	next_action ∈ {PROCEED, REMEDIATE, ALTERNATE}; misconceptions_detected là list các pattern có trong error ontology
7. InteractionLog	Log tương tác chi tiết	messages (list of {role, content, timestamp}), events (list of event objects), agent_calls (log mỗi lần gọi agent)	Tất cả agents (append-only)	Chỉ append, không sửa/xóa; mỗi message có timestamp hợp lệ

	(cho analytics)			
8. ConfigState	Cấu hình runtime của agents	grounding_mode, tutor_verbosity, evaluator_strictness, planner_optimization_weights (α, β, γ, δ)	System Admin hoặc Learner (preferences)	Các tham số nằm trong khoảng hợp lệ, ví dụ: evaluator_strictness ∈[0, 1]; weights tổng = 1

3.4.1.3. Ownership và validation

- Ownership: Mỗi state component có một agent chính chịu trách nhiệm cập nhật (owner), nhưng các agents khác có thể đọc (read-only). Ví dụ: Path Planner sở hữu LearningPathState, nhưng Tutor/Evaluator đọc current_path để biết concept tiếp theo.
- Validation: Mọi thao tác ghi đều qua validation layer. Nếu vi phạm rules (ví dụ: ghi current_concept_id không tồn tại trong KG), hệ thống từ chối và raise error, ghi log cho debug.

CSM đảm bảo rằng dù có bao nhiêu agent tham gia, hệ thống luôn có một "bức tranh trạng thái" nhất quán, giúp debug, phân tích, và đảm bảo chất lượng dịch vụ.

3.4.2. Event Bus Architecture – Điều phối handoff và logging

3.4.2.1. Publish-subscribe pattern

Event Bus trong VER3 tuân theo publish-subscribe pattern (pub-sub), nơi các agents và system components có thể:

- Publish (phát) events khi hoàn thành một tác vụ quan trọng hoặc phát hiện một sự kiện đặc biệt.

- Subscribe (đăng ký) để nhận thông báo về các loại events mà chúng quan tâm.

Lợi ích chính:

- Decoupling: Agents không cần biết chi tiết về nhau, chỉ cần biết event nào để lắng nghe.
- Scalability: Dễ dàng thêm agents mới hoặc analytics module mà không sửa code agents hiện tại.
- Traceability: Mọi sự kiện quan trọng đều được ghi lại, tạo audit trail đầy đủ cho phân tích và compliance.

Cấu trúc một event:

json


```
{
  "event_id": "evt_20251209_134501_abc",
  "event_type": "EVALUATION_COMPLETED",
  "timestamp": "2025-12-09T13:45:01.234Z",
  "learner_id": "learner_123",
  "session_id": "session_456",
  "payload": {
    "concept_id": "concept.sql.select",
    "score": 0.85,
    "mastery_level": 0.80,
    "decision": "PROCEED",
    "misconceptions": []
  },
  "source_agent": "EvaluatorAgent"
}
```


3.4.2.2. Taxonomy 30+ loại event

Hệ thống định nghĩa một taxonomy gồm 30+ loại event, được nhóm theo 6 categories:

Category	Event Types (ví dụ)	Mô tả
Learner Lifecycle	LEARNER_REGISTERED, LEARNER_PROFILE_CREATED, LEARNER_PROFILE_UPDATED	Các sự kiện liên quan

		đến vòng đời người học
Path Planning	PATH_REQUESTED, PATH_PLANNED, PATH_UPDATED, REMEDIATION_PATH_CREATED, ALTERNATIVE_PATH_SUGGESTED	Sự kiện về lập kế hoạch lộ trình
Learning Session	SESSION_STARTED, CONCEPT_STARTED, CONCEPT_COMPLETED, SESSION_PAUSED, SESSION_RESUMED, SESSION_ENDED	Sự kiện trong phiên học
Tutoring & Interaction	TUTOR_MESSAGE_SENT, LEARNER_MESSAGE_RECEIVED, SOCRATIC_QUESTION_ASKED, HINT_PROVIDED	Các tương tác Tutor – Learner
Evaluation	EVALUATION_STARTED, EVALUATION_COMPLETED, QUIZ_GENERATED, QUIZ_SUBMITTED, MISCONCEPTION_DETECTED	Đánh giá và phát hiện lỗi sai
Artifact & PKM	ARTIFACT_CREATED, NOTE_LINKED, PERSONAL_KG_UPDATED, ARTIFACT_ACCESSED	Tạo và quản lý knowledge artifacts

Mỗi agent subscribe các event phù hợp với vai trò của mình:

- Path Planner subscribe: EVALUATION_COMPLETED (với decision = REMEDIATE/ALTERNATE), LEARNER_PROFILE_UPDATED.
- KAG Agent subscribe: CONCEPT_COMPLETED, EVALUATION_COMPLETED, SESSION_ENDED.
- Analytics Backend subscribe: tất cả events (hoặc hầu hết) để xây dựng dashboard, báo cáo, A/B testing.

3.4.2.3. Bảng 3.5 – Agent Transition State Machine

Bảng 3.5 mô tả cách các agents chuyển trạng thái dựa trên events, giúp hình dung rõ luồng handoff.

Bảng 3.5. Agent Transition State Machine (simplified)

Trạng thái hiện tại	Event kích hoạt	Trạng thái tiếp theo	Agent được kích hoạt	Hành động
IDLE	LEARNER_REGISTERED	PROFILING	Learner Profiler Agent	Bắt đầu thu thập thông tin profile
PROFILING	LEARNER_PROFILE_CREATED	PLANNING	Path Planner Agent	Lập lộ trình dựa trên profile và goal
PLANNING	PATH_PLANNED	TUTORING	Tutor Agent	Bắt đầu hướng dẫn concept đầu tiên trong path
TUTORING	CONCEPT_COMPLETED	EVALUATING	Evaluator Agent	Đánh giá hiểu biết về concept vừa học
EVALUATING	EVALUATION_COMPLETED (decision=PROCEED)	TUTORING	Tutor Agent	Chuyển sang concept tiếp theo

EVALUATING	EVALUATION_COMPLETED (decision=REMEDIATE)	PLANNING	Path Planner Agent	Điều chỉnh path, chèn remediation content
EVALUATING	EVALUATION_COMPLETED (decision=ALTERNATE)	PLANNING	Path Planner Agent	Tìm đường đi thay thế (HAS_ALTERNATIVE_PATH)
EVALUATING	EVALUATION_COMPLETED (any decision)	ARTIFACT_GENERATION	KAG Agent	Tạo atomic note từ dialogue và eval result
ARTIFACT_GENERATION	ARTIFACT_CREATED	TUTORING hoặc SESSION_ENDED	Tutor Agent hoặc System	Tiếp tục hoặc kết thúc phiên
TUTORING	SESSION_PAUSED	PAUSED	None (chờ resume)	Lưu trạng thái vào CSM
PAUSED	SESSION_RESUMED	TUTORING	Tutor Agent	Khôi phục từ CSM, tiếp tục

State machine này giúp hệ thống dễ test, debug và mở rộng, đồng thời cung cấp cơ sở cho thiết kế UI/UX (hiển thị trạng thái cho người học).

3.4.3. Agent Grounding Architecture – Giải quyết vấn đề Hallucination

3.4.3.1. Vấn đề hallucination trong AI tutoring

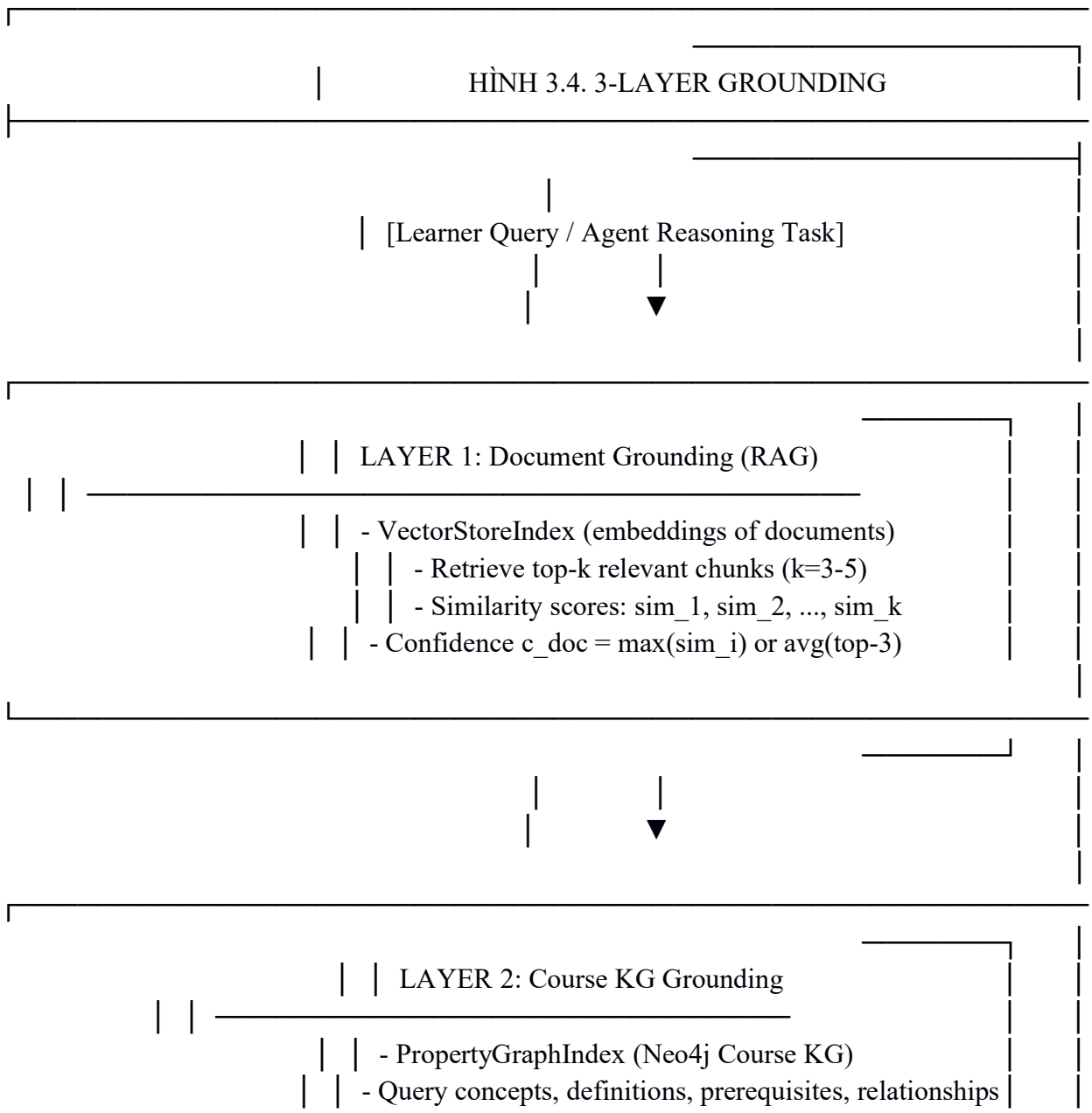
Một trong những thách thức lớn nhất khi sử dụng LLM trong giáo dục là hallucination – hiện tượng mô hình sinh ra thông tin sai lệch, không có trong tài liệu khóa học, hoặc mâu thuẫn với tri thức chuẩn. Nghiên cứu Harvard 2025 đã liệt hallucination prevention là một trong 7 nguyên tắc sư phạm quan trọng nhất khi thiết kế AI tutor. Nghiên cứu Dartmouth cũng chỉ ra rằng khi AI không được grounding vào tài liệu khóa học, mức độ tin tưởng và sử dụng từ người học giảm mạnh.

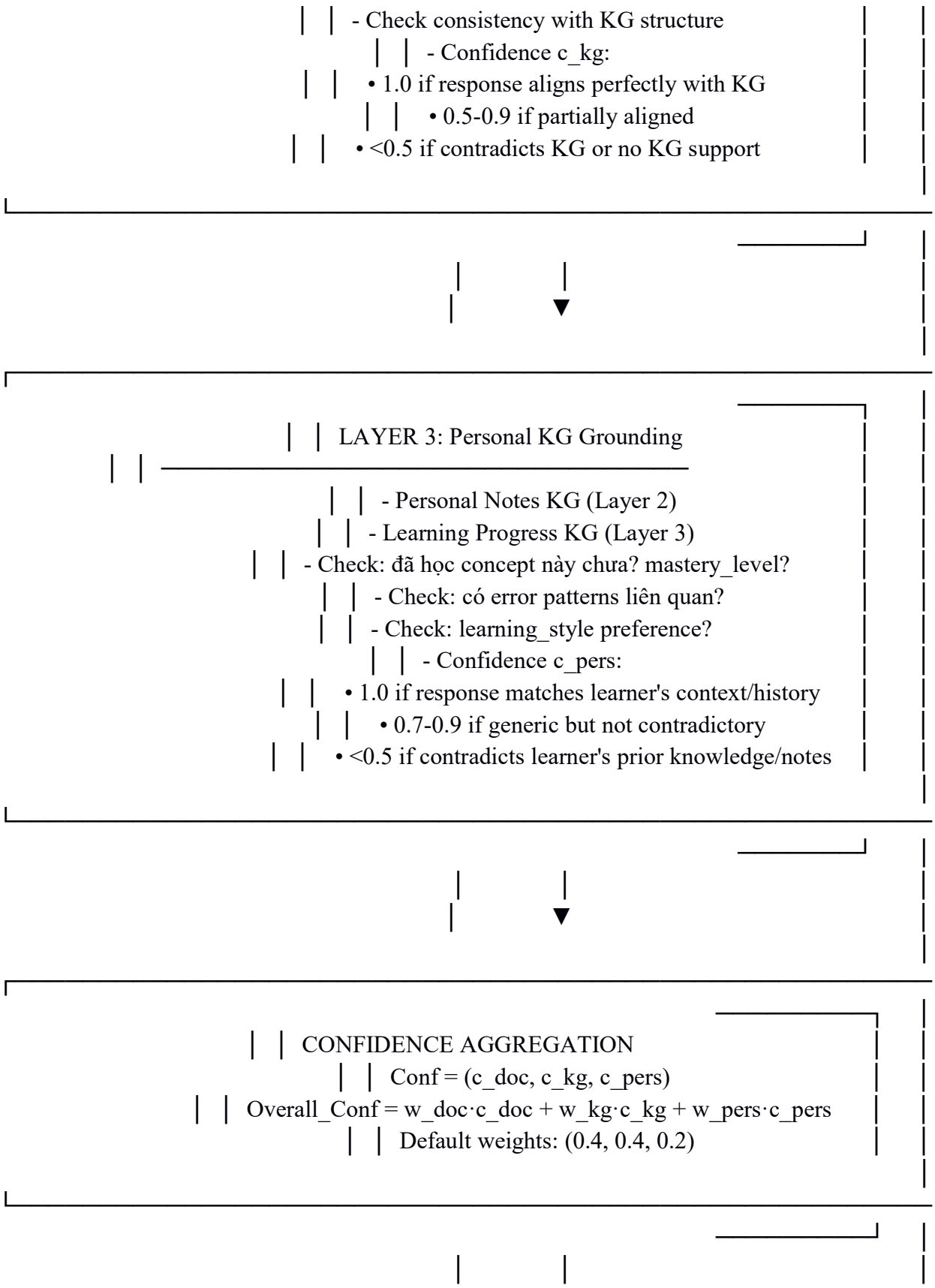
VER3 giải quyết vấn đề này bằng 3-layer grounding architecture, trong đó mọi response của agents (đặc biệt Tutor và Evaluator) đều phải dựa trên ba lớp thông tin:

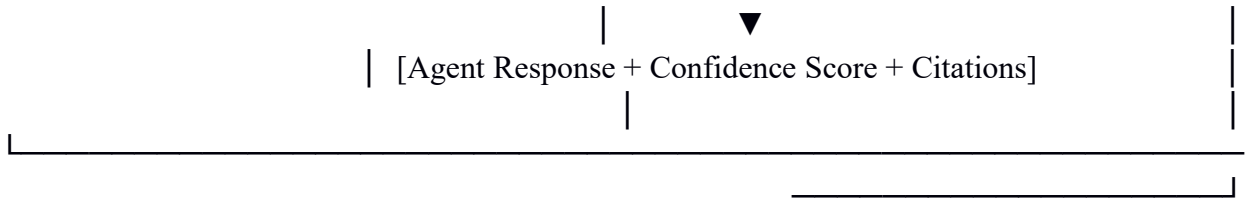
1. Layer 1 – Document Grounding (RAG – VectorStoreIndex)
2. Layer 2 – Course KG Grounding (Property Graph Index)
3. Layer 3 – Personal KG Grounding (Dual-KG Layer 2 + Layer 3)

3.4.3.2. Hình 3.4 – 3-Layer Grounding Architecture

text




- Check consistency with KG structure
- Confidence c_kg:
• 1.0 if response aligns perfectly with KG
• 0.5-0.9 if partially aligned
• <0.5 if contradicts KG or no KG support
LAYER 3: Personal KG Grounding
- Personal Notes KG (Layer 2)
- Learning Progress KG (Layer 3)
- Check: đã học concept này chưa? mastery_level?
- Check: có error patterns liên quan?
- Check: learning_style preference?
- Confidence c_pers:
• 1.0 if response matches learner's context/history
• 0.7-0.9 if generic but not contradictory
• <0.5 if contradicts learner's prior knowledge/notes
CONFIDENCE AGGREGATION
Conf = (c_doc, c_kg, c_pers)
Overall_Conf = w_doc·c_doc + w_kg·c_kg + w_pers·c_pers
Default weights: (0.4, 0.4, 0.2)



3.4.3.3. Confidence Scoring Formula

Mỗi response được gán một confidence triple:

$$Conf=(c_{doc},c_{kg},c_{pers})\in[0,1]^3$$

Và một overall confidence score:

$$Overall\_Conf=w_{doc}\cdot c_{doc}+w_{kg}\cdot c_{kg}+w_{pers}\cdot c_{pers}$$

Trong đó:

- w_{doc},w_{kg},w_{pers} là trọng số (mặc định: 0.4, 0.4, 0.2; tổng = 1).
- c_{doc}: độ tin cậy từ lớp document retrieval (RAG).
- c_{kg}: độ tin cậy từ lớp Course KG.
- c_{pers}: độ tin cậy từ lớp Personal KG.

Cách tính từng thành phần:

1. c_{doc}:
    a. Nếu dùng RAG, lấy similarity_score của top-1 chunk, hoặc trung bình top-3.
    b. Ví dụ: nếu top-3 chunks có sim = [0.92, 0.88, 0.85], thì $c_{doc}=avg(0.92,0.88,0.85)=0.883$.
2. c_{kg}:
    a. Agent kiểm tra xem response có nhất quán với Course KG không:
        a.i. Nếu concept được đề cập tồn tại trong KG, definition khớp, prerequisite đúng → $c_{kg}=1.0$.
        a.ii. Nếu concept tồn tại nhưng một số chi tiết không rõ → $c_{kg}=0.7$.

a.iii. Nếu concept không tồn tại hoặc mâu thuẫn với KG → $c_{kg}=0.3$ckg=0.3 (cảnh báo hallucination).

3. c_{pers}cpers:

   a. Kiểm tra với Personal KG:

      a.i. Response có phù hợp với mastery_level hiện tại không? (ví dụ: không đưa concept quá nâng cao).

      a.ii. Có lặp lại lỗi sai đã phát hiện trước đó không?

      a.iii. Có phù hợp với learning_style không? (ví dụ: nếu visual, có gợi ý diagram không?)

   b. Nếu phù hợp tốt → $c_{pers}=1.0$cpers=1.0; nếu generic nhưng OK → 0.8; nếu không phù hợp → 0.5.

Cách sử dụng Overall_Conf:

- Nếu Overall_Conf ≥ 0.8: response được xem là highly confident, hiển thị bình thường.
- Nếu 0.5 ≤ Overall_Conf < 0.8: response moderately confident, có thể thêm disclaimer ("Thông tin này dựa trên tài liệu tham khảo, bạn nên xem lại...").
- Nếu Overall_Conf < 0.5: response low confidence, agent cần:
  - Hoặc từ chối trả lời ("Tôi không có đủ thông tin..."),
  - Hoặc yêu cầu retrieval lại (expand search, thử query khác).

3.4.3.4. Bảng 3.6 – Quy tắc sử dụng Confidence Score

Bảng 3.6. Confidence Score Usage Rules

Overall_Conf	Mức độ tin cậy	Hành động của Agent	Hiển thị cho người học
≥ 0.9	Very High	Trả lời trực tiếp, không cần disclaimer	Nội dung bình thường, không cảnh báo
0.8 – 0.89	High	Trả lời trực tiếp, có thể thêm citation nhẹ	Nội dung bình thường + citation footnote (tùy chọn)
0.6 – 0.79	Moderate	Trả lời +	Nội dung + disclaimer

		disclaimer nhẹ ("Dựa trên tài liệu...")	+ citation
0.4 – 0.59	Low	Cảnh báo rõ ràng ("Thông tin này không chắc chắn...")	Warning badge + citation + gợi ý kiểm tra thêm
< 0.4	Very Low	Từ chối trả lời hoặc yêu cầu retrieval lại	"Tôi không có đủ thông tin để trả lời câu hỏi này. Bạn có thể hỏi rõ hơn hoặc tham khảo tài liệu..."

Quy tắc này giúp hệ thống minh bạch với người học, tăng độ tin cậy và giảm thiểu rủi ro cung cấp thông tin sai lệch.

3.4.3.5. Bảng 3.7 – Agent-Specific Grounding Strategy

Mỗi agent có chiến lược grounding khác nhau, tùy thuộc vào vai trò:

Bảng 3.7. Agent-Specific Grounding Strategy

Agent	Layer 1 (Document)	Layer 2 (Course KG)	Layer 3 (Personal KG)	Trọng số *(wdoc,wkg,wpers)* (*w*doc,*w*kg,*w*pers)	Ghi chú
Learner Profiler	Không dùng	Đọc Course KG để biết các concepts available	Đọc/ghi Personal KG (Layer 2, 3) để lưu profile	(0, 0.3, 0.7)	Ưu tiên Personal KG vì vai trò chính là xây dựng profile cá nhân
Path Planner	Không dùng	Dùng nhiều (truy vấn prerequisites, alternatives)	Dùng nhiều (mastery_level, completed_concepts)	(0, 0.5, 0.5)	Cân bằng giữa cấu trúc Course KG và tiến độ cá nhân
Tutor Agent	Dùng nhiều (RAG để giải thích, ví dụ)	Dùng nhiều (definition, example từ KG)	Dùng vừa (notes, misconceptions, style)	(0.4, 0.4, 0.2)	Grounding mạnh vào document và KG để tránh hallucination
Evaluator	Dùng (retrieve đáp	Dùng nhiều (learning_objective,	Dùng nhiều (error_patterns,	(0.3, 0.4, 0.3)	Cân bằng document

Agent	án chuẩn, rubric)	skill_level)	mastery_history)		(rubric), KG (objective), Personal (history)
KAG Agent	Dùng ít (chỉ lấy template note)	Dùng (concept definition để sinh note)	Dùng nhiều (đọc/ghi Personal Notes KG)	(0.1, 0.3, 0.6)	Ưu tiên Personal KG vì tạo artifacts cá nhân

Nhờ phân tầng rõ ràng, mỗi agent có thể tối ưu hóa grounding cho vai trò riêng, đồng thời hệ thống vẫn đảm bảo tính nhất quán chung.

3.4.4. Double Grounding: KG + Vector DB (Tutor Agent)

Tutor Agent là agent quan trọng nhất trong việc tương tác trực tiếp với người học, do đó cần grounding mạnh mẽ nhất. VER3 thiết kế double grounding cho Tutor Agent: vừa dùng RAG (VectorStoreIndex) vừa dùng KG (PropertyGraphIndex) trong mỗi response.

3.4.4.1. Dartmouth Precision Education và RAG

Nghiên cứu Dartmouth 2025 đã chứng minh rằng RAG architecture – buộc AI phải retrieve tài liệu trước khi generate – là chìa khóa để tăng độ tin cậy và sử dụng. Trong case study của Dartmouth, khi triển khai RAG cho AI teaching assistant, mức độ sử dụng tăng 329% trong giai đoạn trước kỳ thi, phản ánh sự tin tưởng cao từ sinh viên vào độ chính xác của thông tin.

VER3 áp dụng nguyên tắc này bằng cách:

- Bước 1: Tutor Agent nhận câu hỏi/context → query VectorStoreIndex để lấy top-k chunks (k=3–5) từ tài liệu khóa học.
- Bước 2: Đồng thời query PropertyGraphIndex (Course KG) để lấy định nghĩa, ví dụ chuẩn, prerequisites của concept hiện tại.
- Bước 3: Tổng hợp (merge) thông tin từ cả hai nguồn, loại bỏ mâu thuẫn (nếu có, ưu tiên KG vì đã validated).
- Bước 4: LLM sinh response dựa trên context đã merge, kèm citations.

3.4.4.2. Pseudocode Tutor Grounding

python

```
def tutor_agent_respond(learner_query, current_concept_id, learner_profile, personal_kg):
    """
    Tutor Agent response với double grounding (RAG + KG).
    """
    # Step 1: Retrieve từ VectorStoreIndex (Layer 1 - Document)
    vector_index = get_vector_store_index()  # LlamaIndex VectorStoreIndex
    retrieved_docs = vector_index.retrieve(
        query=learner_query + " " + current_concept_id,
        top_k=5
    )
    # Tính c_doc
    c_doc = compute_confidence_doc(retrieved_docs)
    doc_context = merge_chunks(retrieved_docs)

    # Step 2: Retrieve từ Course KG (Layer 2 - KG)
    kg_index = get_property_graph_index()  # LlamaIndex PropertyGraphIndex
    concept_node = kg_index.get_node(node_id=current_concept_id)
    definition = concept_node.definition
    example = concept_node.example
    prerequisites = concept_node.prerequisites
    skill_level = concept_node.skill_level

    # Check consistency (có mâu thuẫn giữa doc và KG không?)
    c_kg = compute_confidence_kg(doc_context, concept_node)
    kg_context = format_kg_context(concept_node)

    # Step 3: Retrieve từ Personal KG (Layer 3)
    personal_notes = personal_kg.get_notes_for_concept(current_concept_id)
    error_patterns = personal_kg.get_error_patterns(learner_id, current_concept_id)
    mastery_level = personal_kg.get_mastery_level(learner_id, current_concept_id)
    learning_style = learner_profile.learning_style  # visual, auditory, ...

    c_pers = compute_confidence_personal(personal_notes, error_patterns, mastery_level)
    personal_context = format_personal_context(personal_notes, error_patterns, learning_style)

    # Step 4: Aggregate confidence
    w_doc, w_kg, w_pers = 0.4, 0.4, 0.2  # Bảng 3.7
    overall_conf = w_doc * c_doc + w_kg * c_kg + w_pers * c_pers

    # Step 5: Merge contexts
```

```
merged_context = {
"document": doc_context,
"kg": kg_context,
"personal": personal_context,
"confidence": {"doc": c_doc, "kg": c_kg, "pers": c_pers, "overall": overall_conf}
}

# Step 6: LLM generation với system prompt grounded
system_prompt = get_tutor_system_prompt(
harvard_principles=True,
learnlm_principles=True,
grounding_context=merged_context
)

llm_response = call_llm(
system_prompt=system_prompt,
user_query=learner_query,
context=merged_context,
max_tokens=200  # Harvard: brief 2-4 sentences
)

# Step 7: Kiểm tra confidence và thêm disclaimer nếu cần
if overall_conf < 0.5:
response = "Tôi không có đủ thông tin chắc chắn để trả lời. Bạn có thể hỏi rõ hơn hoặc tham khảo tài liệu..."
citations = []
else:
response = llm_response.text
citations = extract_citations(retrieved_docs, concept_node)
if overall_conf < 0.8:
response += "\n\n(Lưu ý: Thông tin này dựa trên tài liệu khóa học. Bạn nên xem lại để hiểu sâu hơn.)"

# Step 8: Log và publish event
log_tutor_response(learner_id, learner_query, response, merged_context, overall_conf)
publish_event("TUTOR_MESSAGE_SENT", {
"concept_id": current_concept_id,
"confidence": overall_conf
})
```

```
return {
"response": response,
"citations": citations,
"confidence": overall_conf
}
```

Pseudocode này minh họa cách Tutor Agent kết hợp 3 lớp grounding, tính toán confidence và đưa ra response phù hợp với quy tắc trong Bảng 3.6, đồng thời tuân thủ Harvard 7 Principles và LearnLM 5 Principles (chi tiết system prompt sẽ ở Chương 5).

Kết luận Mục 3.4

Mục 3.4 đã trình bày chi tiết ba trụ cột orchestration của VER3:

1. Central State Manager – 8 thành phần state, ownership và validation rules (Bảng 3.4).
2. Event Bus Architecture – Pub-sub pattern, taxonomy 30+ event types, Agent Transition State Machine (Bảng 3.5).
3. 3-Layer Grounding Architecture – Document (RAG) + Course KG + Personal KG, confidence scoring, usage rules (Bảng 3.6), agent-specific grounding (Bảng 3.7), pseudocode double grounding cho Tutor Agent.

Ba thành phần này không chỉ giải quyết vấn đề hallucination (Harvard Principle #7, Dartmouth grounded trust), mà còn tạo nền tảng vững chắc cho thiết kế chi tiết 6 agents trong Chương 5 và đánh giá hiệu quả trong Chương 7.

3.5. THIẾT KẾ CHI TIẾT 6 AGENTS – PART 1: PROFILER, PLANNER, TUTOR

Mục 3.5 trình bày thiết kế chi tiết của 6 agents trong VER3, bao gồm input/output schema, thuật toán cốt lõi, pseudocode, và các use cases minh họa. Part 1 tập trung vào ba agents runtime đầu tiên: Learner Profiler, Path Planner và Tutor Agent.

3.5.1. Agent 2: Learner Profiler v2.0

3.5.1.1. Vai trò và mục tiêu

Learner Profiler Agent là agent đầu tiên tương tác với người học trong runtime, chịu trách nhiệm xây dựng và duy trì hồ sơ học tập đa chiều (multi-dimensional learner profile). Khác với các hệ thống profiling truyền thống chỉ thu thập thông tin tĩnh (tuổi, trình độ), VER3 xây dựng profile 17 chiều, bao gồm cả thông tin động (mastery level theo thời gian), episodic (lịch sử phiên học), và computed (các chỉ số tổng hợp).

Mục tiêu chính:

1. Thu thập đầy đủ thông tin cần thiết cho Path Planner (mục tiêu, constraints, trình độ).
2. Cập nhật liên tục profile dựa trên tương tác với Tutor và kết quả từ Evaluator.
3. Hỗ trợ cá nhân hóa sâu cho tất cả agents khác thông qua Personal KG (Layer 3).

3.5.1.2. Bảng 3.8 – 17 chiều profile

Profiler v2.0 mô hình hóa người học qua 17 chiều, được nhóm thành 6 categories.

Bảng 3.8. 17 Dimensions của Learner Profile v2.0

#	Dimension	Category	Type	Mô tả	Source	Update Frequency
1	learner_id	Static	String	ID duy nhất của người học	Registration	Không đổi
2	demographic	Static	Object	Tuổi, ngôn ngữ, múi giờ	Registration form	Hiếm khi
3	learning_goal	Static/Dynamic	Text + List	Mục tiêu học tập (NL text + concept_ids)	Initial survey, updated by Profiler	Theo yêu cầu
4	learning_style	Static/Dynamic	Enum	VARK model: visual/auditory/reading/kin	VARK test hoặc suy	Theo yêu cầu

				esthetic	luận từ tương tác	
5	skill_level	Dynamic	Enum	beginner/intermediate/advanced	Pre-test + dynamic update	Mỗi phiên
6	available_time	Dynamic	Integer	Thời gian khả dụng (giờ/tuần)	Initial survey + adjusted	Mỗi tuần
7	preferences	Dynamic	Object	Tốc độ học, độ chi tiết, hint level	Explicit + inferred	Mỗi phiên
8	constraints	Dynamic	Object	Thời gian deadline, không học cuối tuần, ưu tiên practice	Initial + updated	Theo yêu cầu
9	concept_mastery_map	Dynamic	Dict	{concept_id: mastery_level [0-1]}	Evaluator Agent	Mỗi evaluation
10	completed_concepts	Dynamic	List	Danh sách concept đã hoàn thành	Evaluator (PROCEED)	Realtime
11	error_patterns	Episodic	List	Các misconception đã phát hiện	Evaluator (misconception mining)	Mỗi evaluation
12	session_history	Episodic	List	Danh sách {session_id, start, end, concepts, events}	System	Mỗi phiên
13	interaction_log	Episodic	List	Log dialogue với Tutor	Tutor Agent	Realtime
14	artifact_ids	Episodic	List	Danh sách note_id từ KAG	KAG Agent	Mỗi artifact
15	avg_mastery_level	Computed	Float	Trung bình mastery trên tất cả concepts đã học	Computed từ (9)	Mỗi phiên
16	learning_velocity	Computed	Float	Tốc độ học (concepts/hour)	Computed từ (10), (12)	Mỗi tuần
17	engagement_score	Aggregated	Float [0-1]	Điểm engagement (dựa trên interaction_log, session duration)	Analytics module	Mỗi tuần

Categories giải thích:

- Static: Không hoặc hiếm khi thay đổi (ID, demographic).
- Dynamic: Thay đổi thường xuyên dựa trên tương tác (skill_level, mastery_map).

- Episodic: Lịch sử các sự kiện (sessions, errors, logs).
- Computed: Tính toán từ các dimension khác (avg_mastery, learning_velocity).
- Aggregated: Tổng hợp từ nhiều nguồn (engagement_score từ logs + duration + quiz results).

3.5.1.3. Episodic Memory Schema – 4 loại episode

VER3 lưu trữ lịch sử học tập dưới dạng episodic memory, chia thành 4 loại episode:

1. Session Episodes: Mỗi phiên học là một episode, chứa start_time, end_time, concepts_covered, events, final_state.
2. Concept Episodes: Lịch sử tương tác với từng concept (lần đầu tiếp xúc, số lần revisit, mastery progression).
3. Error Episodes: Mỗi lần Evaluator phát hiện misconception, tạo một error episode gắn với concept_id, misconception_type, remediation_taken.
4. Artifact Episodes: Mỗi lần KAG tạo note, lưu lại thời điểm, concept liên quan, nội dung note (reference).

Schema JSON mẫu:

json

```
{
  "session_episodes": [
    {
      "session_id": "s_001",
      "start_time": "2025-12-09T08:00:00Z",
      "end_time": "2025-12-09T09:30:00Z",
      "concepts_covered": ["concept.sql.select", "concept.sql.where"],
      "events": [
        {"type": "CONCEPT_STARTED", "concept_id": "concept.sql.select", "time": "08:05"},
        {"type": "EVALUATION_COMPLETED", "decision": "PROCEED", "time": "08:45"},
        {"type": "CONCEPT_STARTED", "concept_id": "concept.sql.where", "time": "08:50"}
      ],
      "final_state": "completed"
    }
  ],
  "concept_episodes": {
    "concept.sql.select": {
```

```
"first_encountered": "2025-12-09T08:05:00Z",
"revisit_count": 0,
"mastery_progression": [
{"time": "08:45", "level": 0.75}
]
}
},
"error_episodes": [
{
"error_id": "err_001",
"time": "2025-12-09T09:15:00Z",
"concept_id": "concept.sql.where",
"misconception_type": "Confusing WHERE with HAVING",
"remediation_taken": "concept.sql.groupby_intro"
}
],
"artifact_episodes": [
{
"artifact_id": "note_001",
"created_at": "2025-12-09T09:30:00Z",
"concept_id": "concept.sql.select",
"note_type": "atomic_note",
"note_ref": "learner123.note.001"
}
]
}
```


Episodic memory cho phép:

- Path Planner xem xét lịch sử khi lập lộ trình (tránh revisit quá nhiều).
- Tutor điều chỉnh cách giải thích dựa trên error history.
- KAG tạo liên kết giữa notes (temporal linkage).

3.5.1.4. Real-time Update Algorithm (Pseudocode)

Profiler Agent cập nhật profile theo thời gian thực (realtime) mỗi khi nhận event từ Event Bus.

python

```
class LearnerProfilerAgent:
def __init__(self, central_state, event_bus, dual_kg):
```

```
        self.state = central_state
        self.bus = event_bus
        self.kg = dual_kg
        # Subscribe các events quan trọng
        self.bus.subscribe("EVALUATION_COMPLETED", self.on_evaluation)
        self.bus.subscribe("SESSION_ENDED", self.on_session_end)
        self.bus.subscribe("ARTIFACT_CREATED", self.on_artifact)

    def on_evaluation(self, event):
        """
        Cập nhật mastery_map, completed_concepts, error_patterns
        khi nhận event EVALUATION_COMPLETED từ Evaluator.
        """
        learner_id = event['learner_id']
        concept_id = event['payload']['concept_id']
        mastery_level = event['payload']['mastery_level']
        decision = event['payload']['decision']
        misconceptions = event['payload']['misconceptions']

        # Update concept_mastery_map (dimension 9)
        profile = self.state.get_state(learner_id, 'LearnerProfile')
        profile['concept_mastery_map'][concept_id] = mastery_level

        # Nếu PROCEED, thêm vào completed_concepts (dimension 10)
        if decision == 'PROCEED':
            if concept_id not in profile['completed_concepts']:
                profile['completed_concepts'].append(concept_id)

        # Thêm error_patterns nếu có misconceptions (dimension 11)
        for misc in misconceptions:
            error_episode = {
                'time': event['timestamp'],
                'concept_id': concept_id,
                'misconception_type': misc['type'],
                'severity': misc['severity']
            }
            profile['error_patterns'].append(error_episode)

        # Recalculate avg_mastery_level (dimension 15)
        profile['avg_mastery_level'] = self.compute_avg_mastery(profile['concept_mastery_map'])
```

```
        # Update Personal KG (Layer 3 - MasteryNode)
        self.kg.update_mastery_node(learner_id, concept_id, mastery_level)

        # Write back to Central State
        self.state.set_state(learner_id, 'LearnerProfile', profile)

        # Log
        self.log(f"Profile updated for {learner_id}: concept {concept_id}, mastery
        {mastery_level}")

    def on_session_end(self, event):
        """
        Cập nhật session_history, learning_velocity khi phiên học kết thúc.
        """
        learner_id = event['learner_id']
        session_id = event['session_id']
        duration_hours = event['payload']['duration_hours']
        concepts_covered = event['payload']['concepts_covered']

        profile = self.state.get_state(learner_id, 'LearnerProfile')

        # Add session episode (dimension 12)
        session_episode = {
            'session_id': session_id,
            'start_time': event['payload']['start_time'],
            'end_time': event['timestamp'],
            'concepts_covered': concepts_covered,
            'duration_hours': duration_hours
        }
        profile['session_history'].append(session_episode)

        # Recalculate learning_velocity (dimension 16)
        total_concepts = len(profile['completed_concepts'])
        total_hours = sum([s['duration_hours'] for s in profile['session_history']])
        profile['learning_velocity'] = total_concepts / total_hours if total_hours > 0 else 0

        self.state.set_state(learner_id, 'LearnerProfile', profile)

    def on_artifact(self, event):
```

```
        """
        Cập nhật artifact_ids khi KAG tạo note mới.
        """
        learner_id = event['learner_id']
        artifact_id = event['payload']['artifact_id']

        profile = self.state.get_state(learner_id, 'LearnerProfile')
        profile['artifact_ids'].append(artifact_id)  # dimension 14
        self.state.set_state(learner_id, 'LearnerProfile', profile)

    def compute_avg_mastery(self, mastery_map):
        """
        Tính trung bình mastery_level từ mastery_map.
        """
        if not mastery_map:
            return 0.0
        return sum(mastery_map.values()) / len(mastery_map)
```

3.5.1.5. Bloom's Taxonomy Integration

Profiler gắn mỗi concept trong concept_mastery_map với Bloom level (từ Course KG), cho phép đánh giá chính xác hơn. Ví dụ:

- Nếu người học mastery 0.9 ở các concepts Bloom level "Remember" nhưng chỉ 0.4 ở "Apply" → skill_level vẫn là "beginner to intermediate", cần tập trung vào practice.
- Profiler tính toán skill_level_distribution:

python

```
    def compute_skill_level_distribution(self, mastery_map, course_kg):
        bloom_levels = ["Remember", "Understand", "Apply", "Analyze", "Evaluate", "Create"]
        distribution = {level: [] for level in bloom_levels}

        for concept_id, mastery in mastery_map.items():
            concept_node = course_kg.get_node(concept_id)
            bloom_level = concept_node.skill_level  # từ Bảng 2.3
            distribution[bloom_level].append(mastery)

        # Tính avg mastery cho từng Bloom level
        avg_by_bloom = {level: (sum(vals)/len(vals) if vals else 0)
                        for level, vals in distribution.items()}
```

```
return avg_by_bloom
```

Kết quả:

json

```
{
  "Remember": 0.85,
  "Understand": 0.78,
  "Apply": 0.65,
  "Analyze": 0.40,
  "Evaluate": 0.20,
  "Create": 0.10
}
```

Phân bố này giúp Path Planner quyết định concept tiếp theo phù hợp với ZPD (Zone of Proximal Development).

3.5.1.6. Use Cases

Use Case 1: Learner mới đăng ký

1. Learner điền form: mục tiêu "Học SQL để phân tích dữ liệu", available_time = 5 giờ/tuần, không có kinh nghiệm lập trình.
2. Profiler chạy short diagnostic test (5 câu hỏi) → xác định skill_level = "beginner".
3. VARK mini-test (4 câu) → learning_style = "visual".
4. Profiler tạo LearnerProfile với 17 dimensions, hầu hết còn trống (mastery_map = {}, session_history = []).
5. Publish event LEARNER_PROFILE_CREATED → handoff to Path Planner.

Use Case 2: Cập nhật sau evaluation

1. Evaluator gửi event: EVALUATION_COMPLETED, concept_id = "concept.sql.join", mastery_level = 0.85, decision = "PROCEED".
2. Profiler nhận event, cập nhật:
    a. concept_mastery_map["concept.sql.join"] = 0.85
    b. completed_concepts.append("concept.sql.join")

c. avg_mastery_level tăng từ 0.70 → 0.72

3. Ghi lại concept_episode vào episodic memory.
4. Update Personal KG (Layer 3): tạo/cập nhật MasteryNode.

3.5.2. Agent 3: Path Planner v2.0 – MOPO Algorithm

3.5.2.1. Vai trò và thách thức

Path Planner Agent chịu trách nhiệm tối ưu hóa lộ trình học tập cá nhân hóa dựa trên hàm mục tiêu đa thành phần (Mục 3.1):

$$f(P) = \alpha \cdot R + \beta \cdot A + \gamma \cdot C + \delta \cdot Ff(P) = \alpha \cdot R + \beta \cdot A + \gamma \cdot C + \delta \cdot F$$

Thách thức chính:

- Không gian tìm kiếm rất lớn (217 concepts, 320+ relationships trong Course KG).
- Cần cân bằng giữa Relevance (phù hợp mục tiêu), Adaptivity (cá nhân hóa), Coherence (logic), Feasibility (thời gian/tài nguyên).
- Phải hỗ trợ real-time re-planning khi Evaluator báo REMEDIATE hoặc ALTERNATE.

VER1 sử dụng A* với heuristic cứng nhắc, không xử lý tốt remediation và alternative paths. VER3 giới thiệu MOPO (Multi-Objective Path Optimization) – một thuật toán lấy cảm hứng từ RL nhưng không phải full RL, không cần training.

3.5.2.2. Định nghĩa MOPO

MOPO là thuật toán tìm kiếm có hướng (directed search) kết hợp:

- Best-first search với heuristic đa mục tiêu.
- Reward shaping để đánh giá chất lượng path theo 4 thành phần (R, A, C, F).
- Pareto filtering để loại bỏ các path bị dominated.

MOPO KHÔNG PHẢI full RL vì:

- Không có training phase (không cần DQN, PPO, actor-critic).
- Không học policy từ experience replay.

- Không cần môi trường simulator.

MOPO chỉ "lấy cảm hứng từ RL" ở chỗ:

- Dùng khái niệm "reward" để đánh giá path.
- Dùng "state" (current node in KG) và "action" (chọn next node).
- Có thể mở rộng sang RL trong tương lai nếu có dữ liệu interaction lớn.

3.5.2.3. Công thức reward 5 thành phần

MOPO định nghĩa reward cho một path $P=[c1,c2,...,cn]P=[c1,c2,\ldots,cn]$ là tổng của 5 thành phần:

$$Rtotal(P)=R1(goal)+R2(mastery)+R3(coherence)+R4(time)+R5(novelty)Rtotal(P)=R1(goal)+R2(mastery)+R3(coherence)+R4(time)+R5(novelty)$$

Trong đó:

1. $R1(goal)R1(goal)$ – Goal Alignment Reward:

$$R1=\alpha \cdot /goal\_nodes \cap P//goal\_nodes/R1=\alpha \cdot |goal\_nodes||goal\_nodes \cap P|$$

Đo mức độ path đi qua các concept mục tiêu. Nếu path bao phủ tất cả goal_nodes → $R1=\alpha R1=\alpha$.

2. $R2(mastery)R2(mastery)$ – Adaptivity Reward:

$$R2=\beta \cdot (1-1n\sum ci \in Pmastery(ci))R2=\beta \cdot (1-n1ci \in P\sum mastery(ci))$$

Ưu tiên concepts chưa thành thạo. Nếu path toàn concepts đã mastery → $R2\approx 0R2\approx 0$.

3. $R3(coherence)R3(coherence)$ – Structural Coherence Reward:

$$R3=\gamma \cdot valid\_edges(P)n-1R3=\gamma \cdot n-1valid\_edges(P)$$

Đo tỷ lệ các bước tuân thủ quan hệ trong Course KG (REQUIRES, NEXT, IS_PREREQUISITE_OF). Nếu path nhảy cóc → $R3R3$ thấp.

4. $R4(time)R4(time)$ – Feasibility Reward:

$$R4=\delta \cdot max(0,1-\sum ci \in Ptime(ci)Tmax)R4=\delta \cdot max(0,1-Tmax\sum ci \in Ptime(ci))$$

Phạt path quá dài. Nếu tổng thời gian vượt $TmaxTmax$ → $R4<0R4<0$.

5. $R5(novelty)R5(novelty)$ – Exploration Bonus:

$$R5=\epsilon \cdot /new\_concepts(P)/nR5=\epsilon \cdot n|\text{new\_concepts}(P)|$$

Khuyến khích khám phá concepts mới (chưa từng học). $\epsilon\epsilon$ nhỏ (ví dụ 0.05).

Tổng reward:

$$Rtotal=R1+R2+R3+R4+R5R\text{total}=R1+R2+R3+R4+R5$$

với $\alpha=0.30,\beta=0.30,\gamma=0.20,\delta=0.20,\epsilon=0.05\alpha=0.30,\beta=0.30,\gamma=0.20,\delta=0.20,\epsilon=0.05$ (theo Bảng 3.1).

3.5.2.4. MOPO vs A* Comparison

Tiêu chí	A* (VER1)	MOPO (VER3)
Heuristic	Single objective: shortest path với penalty cứng nhắc	Multi-objective: reward shaping 5 thành phần
Cá nhân hóa	Ít (chỉ dựa prerequisites)	Cao (dựa mastery_map, error_patterns, learning_style)
Remediation	Không hỗ trợ tự động	Tự động re-planning với REMEDIATES, HAS_ALTERNATIVE_PATH
Pareto filtering	Không	Có (loại bỏ dominated paths)
Interpretability	Rõ ràng (dijkstra-like)	Rõ ràng (reward components có thể giải thích)
Training	Không cần	Không cần
Mở rộng sang RL	Khó	Dễ (đã có reward function)

3.5.2.5. Why MOPO? – Interpretability & No Training

Lý do chọn MOPO thay vì các phương pháp khác:

1. Không cần training data lớn: Khác với deep RL (DQN, PPO), MOPO hoạt động ngay với Course KG và LearnerProfile hiện có.

2. Interpretability cao: Mỗi thành phần reward có ý nghĩa rõ ràng, dễ debug và giải thích cho người học ("Path này được chọn vì phù hợp với mục tiêu của bạn 90%, đồng thời tối ưu thời gian").

3. Kết hợp tốt với Central State: MOPO đọc trực tiếp từ LearnerProfile (mastery_map) và Course KG, không cần thêm model.

4. Mở rộng dễ dàng: Nếu sau này có nhiều dữ liệu interaction, có thể chuyển sang học reward function bằng inverse RL.

3.5.2.6. Integration với Central State

MOPO tích hợp chặt chẽ với Central State Manager:

python

```
class PathPlannerAgent:
def __init__(self, central_state, course_kg, personal_kg):
self.state = central_state
self.course_kg = course_kg
self.personal_kg = personal_kg

def plan_path(self, learner_id):
"""
Lập lộ trình sử dụng MOPO.
"""
# Step 1: Đọc LearnerProfile từ Central State
profile = self.state.get_state(learner_id, 'LearnerProfile')
goal_nodes = profile['learning_goal']['concept_ids']
mastery_map = profile['concept_mastery_map']
T_max = profile['available_time'] * 60  # giờ → phút
completed = set(profile['completed_concepts'])

# Step 2: Xác định start node (concept đầu tiên chưa học)
start_node = self.find_start_node(goal_nodes, completed, mastery_map)

# Step 3: Chạy MOPO
candidate_paths = self.mopo_search(
start=start_node,
goals=goal_nodes,
mastery_map=mastery_map,
T_max=T_max,
completed=completed
)

# Step 4: Pareto filtering
pareto_paths = self.pareto_filter(candidate_paths)

# Step 5: Chọn path tốt nhất (highest total reward)
best_path = max(pareto_paths, key=lambda p: p['reward_total'])

# Step 6: Lưu vào Central State
```

```
        self.state.set_state(learner_id, 'LearningPathState', {
            'current_path': best_path['nodes'],
            'path_metadata': best_path['metadata'],
            'created_at': now()
        })

        # Step 7: Publish event
        self.publish_event('PATH_PLANNED', {
            'learner_id': learner_id,
            'path': best_path['nodes'],
            'reward': best_path['reward_total']
        })

        return best_path

    def mopo_search(self, start, goals, mastery_map, T_max, completed):
        """
        MOPO best-first search với reward shaping.
        """
        from heapq import heappush, heappop

        # Priority queue: (negative_reward, path, current_node, time_used)
        pq = []
        heappush(pq, (0, [start], start, 0))
        visited = set()
        candidate_paths = []

        while pq and len(candidate_paths) < 10:  # giới hạn 10 paths
            neg_reward, path, current, time_used = heappop(pq)

            if current in visited:
                continue
            visited.add(current)

            # Kiểm tra xem đã đạt goal chưa
            if set(goals).issubset(set(path)):
                reward_total = -neg_reward
                candidate_paths.append({
                    'nodes': path,
                    'reward_total': reward_total,
```

```
                'metadata': self.compute_metadata(path, mastery_map, time_used)
                })
                continue

            # Expand neighbors
            neighbors = self.course_kg.get_neighbors(current, rel_types=['NEXT',
                'IS_PREREQUISITE_OF'])
            for neighbor in neighbors:
                if neighbor in path:  # tránh cycle
                    continue

                new_path = path + [neighbor]
                new_time = time_used + self.course_kg.get_node(neighbor).time_estimate

                if new_time > T_max:  # vi phạm time constraint
                    continue

                # Tính reward cho path hiện tại
                reward = self.compute_reward(new_path, goals, mastery_map, T_max, completed)
                heappush(pq, (-reward, new_path, neighbor, new_time))

        return candidate_paths

    def compute_reward(self, path, goals, mastery_map, T_max, completed):
        """
        Tính reward theo 5 thành phần.
        """
        n = len(path)
        alpha, beta, gamma, delta, epsilon = 0.30, 0.30, 0.20, 0.20, 0.05

        # R1: Goal alignment
        goal_coverage = len(set(goals) & set(path)) / len(goals)
        R1 = alpha * goal_coverage

        # R2: Mastery (adaptivity)
        avg_mastery = sum([mastery_map.get(c, 0) for c in path]) / n
        R2 = beta * (1 - avg_mastery)

        # R3: Coherence
        valid_edges = sum([1 for i in range(n-1) if self.is_valid_edge(path[i], path[i+1])])
```

```
        R3 = gamma * (valid_edges / (n - 1)) if n > 1 else 0

        # R4: Time feasibility
        total_time = sum([self.course_kg.get_node(c).time_estimate for c in path])
        R4 = delta * max(0, 1 - total_time / T_max)

        # R5: Novelty
        new_concepts = [c for c in path if c not in completed]
        R5 = epsilon * (len(new_concepts) / n)

        return R1 + R2 + R3 + R4 + R5
```

3.5.2.7. Real-time Re-planning Logic

Khi Evaluator gửi event EVALUATION_COMPLETED với decision = REMEDIATE hoặc ALTERNATE, Path Planner phải re-plan:

python

```
def on_remediation_required(self, event):
    learner_id = event['learner_id']
    concept_id = event['payload']['concept_id']
    misconception = event['payload']['misconception']

    # Tìm remediation path trong Course KG
    remediation_concepts = self.course_kg.get_related_nodes(
        concept_id, rel_type='REMEDIATES'
    )

    # Chèn remediation vào current_path
    current_path = self.state.get_state(learner_id, 'LearningPathState')['current_path']
    current_index = current_path.index(concept_id)

    # Insert remediation concepts trước concept hiện tại
    new_path = (current_path[:current_index] +
                remediation_concepts +
                [concept_id] +
                current_path[current_index+1:])

    # Update state
    self.state.update_state(learner_id, 'LearningPathState', {'current_path': new_path})
```

```
# Publish event
self.publish_event('PATH_UPDATED', {
    'learner_id': learner_id,
    'reason': 'remediation',
    'added_concepts': remediation_concepts
})
```

3.5.3. Agent 4: Tutor Agent v2.0

3.5.3.1. Vai trò và nguyên tắc sư phạm

Tutor Agent là agent tương tác trực tiếp nhất với người học, chịu trách nhiệm:

- Giải thích concepts theo cách phù hợp với learning_style và mastery_level.
- Đặt câu hỏi Socratic/Reverse Socratic để khuyến khích tư duy chủ động.
- Cung cấp hints từng bước thay vì đáp án trực tiếp.
- Phát hiện và ghi lại misconceptions (phối hợp với Evaluator).

Tutor v2.0 tuân thủ Harvard 7 Principles (Mục 2.6):

1. Active Learning
2. Manage Cognitive Load (2-4 sentences per response)
3. One Step at a Time
4. Encourage Self-Thinking
5. Promote Growth Mindset
6. Personalized Feedback
7. Hallucination Prevention (3-layer grounding)

Đồng thời tích hợp LearnLM 5 Principles: Inspire Active Learning, Manage Cognitive Load, Adapt to Learner, Stimulate Curiosity, Deepen Metacognition.

3.5.3.2. Stateful Socratic Dialogue

Khác với chatbot thông thường (stateless), Tutor Agent duy trì dialogue state cho mỗi concept, theo dõi:

- Số lần người học đã cố gắng trả lời.
- Loại hints đã cung cấp.
- Mức độ "scaffolding" hiện tại (high/medium/low).

Dialogue State Schema

json

```
{
  "dialogue_state": {
    "concept_id": "concept.sql.join",
    "phase": "explanation",  // explanation | questioning | practice | assessment
    "attempts": 0,
    "hints_given": [],
    "scaffolding_level": "high",  // high | medium | low
    "misconceptions_detected": [],
    "last_user_response": "I think JOIN combines two tables...",
    "tutor_strategy": "socratic"  // socratic | reverse_socratic | direct
  }
}
```

3.5.3.3. Dialogue State Machine

Tutor sử dụng state machine với 4 phases:

text

```
[EXPLANATION] → [QUESTIONING] → [PRACTICE] → [ASSESSMENT]
                                    ↓              ↓              ↓              ↓
                              (hints)     (scaffolding)  (feedback)   (handoff to Evaluator)
```

1. EXPLANATION: Giải thích concept ban đầu, dùng RAG + Course KG + Personal KG.
2. QUESTIONING: Đặt câu hỏi Socratic để kiểm tra hiểu biết.
3. PRACTICE: Đưa bài tập nhỏ, theo dõi attempts, cung cấp hints nếu cần.

4. ASSESSMENT: Handoff to Evaluator Agent để đánh giá chính thức.

Pseudocode chuyển phase:

python

```
def advance_phase(self, dialogue_state):
    current_phase = dialogue_state['phase']

    if current_phase == 'explanation':
        # Sau giải thích, chuyển sang questioning
        dialogue_state['phase'] = 'questioning'
        dialogue_state['scaffolding_level'] = 'high'

    elif current_phase == 'questioning':
        # Nếu trả lời đúng 2/3 câu hỏi, chuyển practice
        if dialogue_state['correct_answers'] >= 2:
            dialogue_state['phase'] = 'practice'
            dialogue_state['scaffolding_level'] = 'medium'
        else:
            # Vẫn ở questioning, giảm scaffolding
            dialogue_state['scaffolding_level'] = 'low'

    elif current_phase == 'practice':
        # Sau 3 attempts hoặc hoàn thành, chuyển assessment
        if dialogue_state['attempts'] >= 3 or dialogue_state['practice_completed']:
            dialogue_state['phase'] = 'assessment'
            # Handoff to Evaluator
            self.handoff_to_evaluator(dialogue_state)

    return dialogue_state
```

3.5.3.4. Harvard 7 Principles Implementation

Mỗi nguyên tắc được implement cụ thể trong system prompt và logic:

Principle	Implementation
Active Learning	Không đưa đáp án trực tiếp; dùng Socratic questions: "Bạn nghĩ điều gì sẽ xảy ra nếu...?"
Manage Cognitive Load	Giới hạn response ≤ 4 câu; chia khái niệm phức tạp thành sub-concepts nhỏ
One Step at a Time	Chỉ tiết lộ 1 bước mỗi lần; chờ learner xử lý trước khi tiếp tục

Encourage Self-Thinking	Dùng probing questions thay vì hints trực tiếp; "Tại sao bạn nghĩ vậy?"
Growth Mindset	Phản hồi tích cực khi sai: "Ý tưởng tốt! Nhưng hãy xem xét..."
Personalized Feedback	Dựa vào Personal KG (notes, errors) để điều chỉnh ví dụ và giải thích
Hallucination Prevention	3-layer grounding (RAG + Course KG + Personal KG) với confidence scoring

3.5.3.5. Learner Misconception Tracking

Tutor Agent theo dõi misconceptions qua dialogue, ghi lại để Evaluator xác nhận:

python

```
def detect_misconception(self, user_response, concept_id):
    """
    Phát hiện misconception từ câu trả lời của learner.
    """
    # Retrieve common misconceptions từ Course KG
    common_errors = self.course_kg.get_node(concept_id).common_errors

    # So sánh user_response với common_errors (dùng semantic similarity)
    for error in common_errors:
        if self.semantic_similarity(user_response, error['description']) > 0.75:
            # Phát hiện misconception
            return {
                'type': error['type'],
                'severity': error['severity'],
                'detected_from': user_response
            }

    return None
```

3.5.3.6. Hint Generation Strategy

Hints được sinh theo 3 mức độ scaffolding:

1. High scaffolding (phase đầu): Gợi ý gần như đầy đủ, chỉ thiếu 1-2 chi tiết.
2. Medium scaffolding: Gợi ý một phần, dẫn dắt tư duy.
3. Low scaffolding: Chỉ đặt câu hỏi dẫn, không gợi ý cụ thể.

python

```
def generate_hint(self, concept_id, scaffolding_level):
 concept_node = self.course_kg.get_node(concept_id)

 if scaffolding_level == 'high':
  # Gợi ý gần như đầy đủ
  hint = f"Bạn cần sử dụng câu lệnh {concept_node.example[:20]}... để giải quyết vấn đề này."

 elif scaffolding_level == 'medium':
  # Gợi ý một phần
  hint = f"Hãy nghĩ về cách {concept_node.definition[:30]}... có thể áp dụng ở đây."

 else:  # low
  # Chỉ đặt câu hỏi
  hint = f"Tại sao bạn nghĩ rằng cần dùng {concept_node.label}? Có cách nào khác không?"

 return hint
```

3.5.3.7. Reverse Socratic Method

Tutor v2.0 hỗ trợ Reverse Socratic: learner đưa ra giả thuyết, AI dẫn dắt bằng câu hỏi để learner tự phát hiện lỗi sai.

Ví dụ:

Learner: "Tôi nghĩ JOIN và UNION là giống nhau."
Tutor: "Thú vị! Bạn có thể giải thích tại sao bạn nghĩ vậy không?"
Learner: "Cả hai đều kết hợp hai bảng."
Tutor: "Đúng, cả hai đều liên quan đến hai bảng. Nhưng bạn thử nghĩ xem: JOIN kết hợp theo chiều nào, còn UNION theo chiều nào?"
Learner: "Ồ, JOIN theo chiều ngang (cột), UNION theo chiều dọc (hàng)?"
Tutor: "Chính xác! Bây giờ bạn đã thấy sự khác biệt."

3.5.3.8. RAG Double Grounding (Pseudocode)

Chi tiết pseudocode đã được trình bày ở Mục 3.4.4. Tóm tắt luồng:

1. Retrieve từ VectorStoreIndex (top-5 chunks) → c_doc.
2. Query Course KG (definition, example, prerequisites) → c_kg.

3. Query Personal KG (notes, error_patterns) → c_pers.
4. Merge contexts + compute overall_conf.
5. LLM generation với system prompt Harvard 7P + LearnLM 5P.
6. Kiểm tra confidence, thêm disclaimer nếu cần.
7. Log + publish event.

Kết luận Part 1

Part 1 đã trình bày chi tiết thiết kế của 3 agents runtime đầu tiên:

1. Learner Profiler v2.0: 17 dimensions, episodic memory, real-time update algorithm, Bloom integration.
2. Path Planner v2.0: MOPO algorithm, 5-component reward function, comparison với A*, integration với Central State, real-time re-planning.
3. Tutor Agent v2.0: Stateful Socratic dialogue, 4-phase state machine, Harvard 7 Principles implementation, misconception tracking, hint generation, Reverse Socratic, double grounding.

Part 2 sẽ trình bày Evaluator Agent, KAG Agent và Integration scenario end-to-end.

3.5. THIẾT KẾ CHI TIẾT 6 AGENTS – PART 2: EVALUATOR, KAG, INTEGRATION

Part 2 của Mục 3.5 trình bày thiết kế chi tiết cho ba agents còn lại: Evaluator Agent, KAG Agent (Knowledge Artifact Generator) và mục Integration mô tả end-to-end flow của toàn bộ 6 agents.

3.5.4. Agent 5: Evaluator Agent v2.0

3.5.4.1. Vai trò và mục tiêu

Evaluator Agent chịu trách nhiệm đánh giá mức độ hiểu và năng lực của người học tại các checkpoint trong LearningPath. Khác với quiz engine truyền thống, Evaluator v2.0:

- Sinh câu hỏi cá nhân hóa theo profile và lịch sử học.
- Phân loại misconception (lỗi sai) theo ontology.

- Cập nhật mastery trên từng concept theo Bloom's 6 levels.
- Ra quyết định PROCEED/REMEDIATE/ALTERNATE, tạo feedback chi tiết.
- Gửi dữ liệu cho Learning Analytics (6 metrics) và cho KAG tạo artifacts.

3.5.4.2. Personalized Question Generation

Evaluator sinh câu hỏi dựa trên:

- Course KG: learning_objective, skill_level (Bloom), example, common_errors.
- LearnerProfile: mastery_current, learning_style, error_patterns.
- Session context: concept hiện tại, các concept liên quan đã học.

Bảng 3.9. Chiến lược sinh câu hỏi theo Bloom level

Bloom Level	Mục tiêu	Loại câu hỏi	Ví dụ (SQL)
Remember	Nhớ lại kiến thức	Trắc nghiệm định nghĩa	"Câu lệnh nào dùng để chọn tất cả cột?"
Understand	Diễn giải, giải thích	Hỏi ngắn, giải thích	"Giải thích khác nhau giữa WHERE và HAVING."
Apply	Áp dụng vào bài toán cụ thể	Viết câu lệnh, sửa lỗi	"Viết câu SELECT lấy tên sinh viên có điểm > 8."
Analyze	Phân tích cấu trúc, quan hệ	So sánh, phân tích output	"Cho 2 truy vấn, kết quả khác nhau thế nào?"
Evaluate	Đánh giá, phê bình	Hỏi về tối ưu, đúng/sai & giải thích	"Truy vấn này có tối ưu không? Vì sao?"
Create	Sáng tạo, thiết kế	Thiết kế schema, query phức tạp	"Thiết kế truy vấn để xuất báo cáo tổng hợp doanh thu theo tháng."

Pseudocode sinh câu hỏi:

python

```
def generate_questions(self, learner_id, concept_id, num_questions=3):
    profile = self.state.get_state(learner_id, 'LearnerProfile')
    concept_node = self.course_kg.get_node(concept_id)

    # Lấy Bloom level mục tiêu từ Course KG
    target_bloom = concept_node.skill_level  # e.g., Apply
```

```
        mastery = profile['concept_mastery_map'].get(concept_id, 0)

        # Điều chỉnh độ khó câu hỏi dựa trên mastery
        if mastery < 0.4:
            bloom_levels = ['Remember', 'Understand']
        elif mastery < 0.7:
            bloom_levels = ['Understand', 'Apply']
        else:
            bloom_levels = [target_bloom, 'Analyze']

        questions = []
        for level in bloom_levels:
            q = self.llm_generate_question(concept_node, level, profile)
            questions.append(q)
            if len(questions) >= num_questions:
                break

        return questions
```

3.5.4.3. Misconception Classification

Evaluator sử dụng ontology lỗi sai (từ Course KG: CommonErrors, MisconceptionType) để phân loại lỗi:

- Conceptual misconception (hiểu sai khái niệm).
- Procedural error (sai cú pháp, quy trình).
- Overgeneralization (áp dụng sai quy tắc).
- Misinterpretation (hiểu sai đề bài/output).

Pseudocode:

python

```
    def classify_misconceptions(self, learner_answer, concept_id):
        concept_node = self.course_kg.get_node(concept_id)
        error_ontology = concept_node.common_errors  # list of {type, description, severity}

        detected = []
        for err in error_ontology:
            if self.semantic_similarity(learner_answer, err['description']) > 0.75:
```

```
                detected.append({
                    'type': err['type'],
                    'severity': err['severity'],
                    'matched_pattern': err['description']
                })
        return detected
```

3.5.4.4. Mastery Tracking theo Bloom's 6 levels

Mỗi concept gắn với một Bloom level chính (từ Course KG). Evaluator cập nhật mastery_level cho concept dựa trên:

- Độ đúng (accuracy) của câu trả lời.
- Mức Bloom của câu hỏi đã trả lời.
- Lịch sử trả lời trước đó.

Đơn giản hóa: mastery_level mới là weighted moving average:

$$mastery\_new = (1-\lambda) \cdot mastery\_old + \lambda \cdot score\_session$$

với $\lambda \in [0.2, 0.5]$ (tùy config).

python

```
    def update_mastery(self, learner_id, concept_id, session_score):
        profile = self.state.get_state(learner_id, 'LearnerProfile')
        old = profile['concept_mastery_map'].get(concept_id, 0.0)
        lam = 0.3
        new_mastery = (1 - lam) * old + lam * session_score
        profile['concept_mastery_map'][concept_id] = new_mastery
        self.state.set_state(learner_id, 'LearnerProfile', profile)

        # Update Personal KG (MasteryNode)
        self.personal_kg.update_mastery_node(learner_id, concept_id, new_mastery)

        return new_mastery
```

3.5.4.5. Decision Tree: PROCEED / REMEDIATE / ALTERNATE

Evaluator ra quyết định dựa trên mastery_new, loại misconception và LearningPath hiện tại.

Bảng 3.10. Quy tắc ra quyết định

mastery_new	Misconceptions	Quyết định	Ý nghĩa
≥ 0.8	Không hoặc rất nhẹ	PROCEED	Chuyển sang concept tiếp theo
0.6 – 0.79	Nhẹ, procedural	PROCEED + REMIND	Cho qua nhưng nhắc lại lỗi nhỏ
0.4 – 0.59	Vừa, conceptual	REMEDIATE	Quay lại/học thêm concept hỗ trợ
< 0.4	Nặng, conceptual/overgeneralization	REMEDIATE mạnh	Học lại concept nền tảng trước đó
bất kỳ	Không phù hợp với learner context (Personal KG mâu thuẫn)	ALTERNATE	Đề xuất đường đi khác, đổi example hoặc topic

Pseudocode quyết định:

python

```
def decide_next_action(self, learner_id, concept_id, mastery_new, misconceptions):
    has_severe = any(m['severity'] == 'high' for m in misconceptions)
    has_conceptual = any('conceptual' in m['type'] for m in misconceptions)

    if mastery_new >= 0.8 and not has_severe:
        decision = 'PROCEED'
    elif mastery_new >= 0.6 and not has_conceptual:
        decision = 'PROCEED'
    elif mastery_new >= 0.4:
        decision = 'REMEDIATE'
    else:
        decision = 'REMEDIATE'

    # Kiểm tra Personal KG để cân nhắc ALTERNATE
    if self.personal_kg.detect_mismatch(learner_id, concept_id):
        decision = 'ALTERNATE'
```

return decision

3.5.4.6. Learning Analytics – 6 metrics

Evaluator gửi dữ liệu cho module analytics, tính 6 chỉ số chính:

1. Mastery Gain per Concept: $\Delta m = m_{new} - m_{old}$.
2. Time-to-Mastery: Thời gian cần để đạt mastery ≥ 0.8 cho mỗi concept.
3. Attempt Count per Concept: Số lần thử cho đến khi PROCEED.
4. Misconception Frequency: Tần suất xuất hiện mỗi loại misconception.
5. Remediation Effectiveness: Tỷ lệ improvement sau remediation path.
6. Path Efficiency: Số concepts/trung bình vs target.

3.5.4.7. Feedback Loop

Evaluator tạo feedback 2 chiều:

- Cho Learner: giải thích điểm mạnh, điểm yếu, gợi ý bước tiếp theo.
- Cho Path Planner: thông qua event EVALUATION_COMPLETED (decision + misconceptions).
- Cho KAG: event EVALUATION_COMPLETED để tạo artifacts phù hợp.

3.5.5. Agent 6: KAG – Knowledge Artifact Generator v2.0

3.5.5.1. Vai trò và triết lý PKM

KAG Agent (Knowledge Artifact Generator) là agent chuyên trách tạo ra các knowledge artifacts cho người học, nhằm biến quá trình học thành quá trình xây dựng hệ thống tri thức cá nhân (Personal Knowledge Management – PKM).

Nguồn cảm hứng chính là Zettelkasten methodology:

- Mỗi ý tưởng/khái niệm → một "zettel" (atomic note) độc lập.
- Gắn tag, link giữa các zettel để xây dựng mạng tri thức cá nhân.

- Ưu tiên kết nối ý tưởng hơn là lưu trữ thông tin rời rạc.

Trong VER3, KAG:

- Tạo notes từ dialogue Tutor–Learner và EvalResult.
- Cập nhật Personal Notes KG (Layer 2) và liên kết với Course KG.
- Hỗ trợ 4 loại truy vấn: temporal, retrieval, synthesis, review.

3.5.5.2. Artifact Types

KAG sinh ít nhất 4 loại artifact:

1. Atomic Notes: Ghi lại một khái niệm duy nhất, theo format chuẩn:
    a. Title, ConceptID, Definition (own words), Example, Links, CreatedAt.
2. Summaries: Tóm tắt một session hoặc một chủ đề (multi-concept summary).
3. Concept Maps: Sơ đồ liên kết giữa nhiều concepts (graph structure, nodes + edges).
4. Code Snippets: Đoạn mã minh họa (cho môn có lập trình), gắn với concept cụ thể.

Bảng 3.11. Schema cơ bản của artifacts

Artifact Type	Node Label	Thuộc tính chính	Quan hệ chính
Atomic Note	Note	note_id, title, content, concept_id, created_at, tags	LINKS_TO (Note–Note), REFERS_TO (Note–Concept), CREATED_BY (Note–Learner)
Summary	Summary	summary_id, scope (session/topic), content, created_at	SUMMARIZES (Summary–Concept/Session), CREATED_BY
Concept Map	Concept Map	map_id, title, nodes, edges, created_at	MAPS (ConceptMap–Concept), CREATED_BY
CodeSnip	CodeSnippet	snippet_id, language, code, explanation, concept_id, created_at	IMPLEMENTS (Snippet–Concept),

pet			CREATED_BY

3.5.5.3. Zettelkasten Methodology trong KAG

KAG áp dụng các nguyên tắc Zettelkasten:

- Atomicity: Mỗi note chỉ nói về một ý tưởng/khái niệm.
- Own Words: Note viết bằng lời của người học (hoặc AI paraphrase về lời người học), không copy y nguyên.
- Contextualization: Mỗi note trả lời câu hỏi: "Note này hữu ích trong ngữ cảnh nào?".
- Linking: Mỗi note phải có ít nhất một LINKS_TO tới note khác hoặc REFERS_TO tới một concept trong Course KG.

Pseudocode tạo atomic note:

python

```
def create_atomic_note(self, learner_id, concept_id, dialogue_history, eval_result):
    concept_node = self.course_kg.get_node(concept_id)

    # Trích xuất ý chính từ dialogue và feedback
    key_ideas = self.extract_key_ideas(dialogue_history, eval_result)

    # Sinh nội dung note theo "own words"
    note_content = self.llm_generate_note(
        concept_definition=concept_node.definition,
        key_ideas=key_ideas,
        learner_style=self.get_learning_style(learner_id)
    )

    note_id = generate_id('note')
    note = {
        'note_id': note_id,
        'title': f"[{concept_id}] {concept_node.sanitized_concept}",
        'content': note_content,
        'concept_id': concept_id,
        'created_at': now(),
        'tags': concept_node.semantic_tags
    }

    # Lưu vào Personal Notes KG
```

```
        self.personal_kg.create_note_node(learner_id, note)

        # Tạo liên kết REFERS_TO
        self.personal_kg.create_relationship(
            from_node=('Note', note_id),
            rel_type='REFERS_TO',
            to_node=('Concept', concept_id)
        )

        # Gợi ý LINKS_TO tới notes liên quan
        related_notes = self.personal_kg.find_related_notes(learner_id, concept_id)
        for rn in related_notes:
            self.personal_kg.create_relationship(
                from_node=('Note', note_id),
                rel_type='LINKS_TO',
                to_node=('Note', rn['note_id'])
            )

        return note_id
```

3.5.5.4. Personal KG Update Logic & Synchronization với Course KG

Mỗi khi KAG tạo artifact:

1. Tạo node mới trong Personal Notes KG.
2. Tạo quan hệ REFERS_TO tới ConceptNode tương ứng trong Course KG.
3. Tạo thêm quan hệ LINKS_TO, MAPS, IMPLEMENTS tùy loại artifact.
4. Cập nhật ArtifactState trong Central State (dimensions artifact_ids, recent_artifacts).

Điều này đảm bảo double linkage:

- Từ Course KG → có thể truy vấn ra các notes cá nhân liên quan.
- Từ Personal KG → có thể truy vấn ra Course KG để grounding.

3.5.5.5. 4 Query Types hỗ trợ người học

KAG hỗ trợ 4 loại truy vấn chính trên Personal KG:

1. Temporal Queries: "Cho tôi xem các note tôi đã tạo trong tuần trước về SQL JOIN".
2. Retrieval Queries: "Những note nào liên quan đến concept.sql.groupby?".
3. Synthesis Queries: "Tóm tắt connections giữa các note về normalization và indexing".
4. Review Queries: "Tạo flashcards từ các note có mastery_level < 0.6".

Pseudocode ví dụ cho review query:

python

```
def generate_review_cards(self, learner_id):
    # Lấy các concept có mastery thấp
    profile = self.state.get_state(learner_id, 'LearnerProfile')
    low_mastery_concepts = [
        cid for cid, m in profile['concept_mastery_map'].items() if m < 0.6
    ]

    # Lấy notes liên quan
    notes = self.personal_kg.get_notes_for_concepts(learner_id, low_mastery_concepts)

    # Sinh flashcards
    cards = []
    for note in notes:
        q, a = self.llm_generate_flashcard(note)
        cards.append({'question': q, 'answer': a, 'note_id': note['note_id']})

    return cards
```

3.5.6. Integration: All Agents + End-to-End Flow

Mục này mô tả cách 6 agents hoạt động cùng nhau trong 2 scenario tiêu biểu và cung cấp timeline end-to-end cùng sơ đồ tổng thể (C4 Component level – mô tả logic, chi tiết C4 kỹ thuật ở Chương 4).

3.5.6.1. Scenario 1: Fast Learner → Pace Acceleration

Bối cảnh: Người học A đã có nền tảng vững, mastery tăng nhanh, cần tăng tốc độ lộ trình.

1. Khởi tạo:
    a. Profiler xác định skill_level ban đầu = intermediate/advanced.

b. Learning_goal: "Hoàn thành khoá SQL trong 2 tuần".

c. available_time = 10 giờ/tuần, engagement_score cao.

2. Lập lộ trình ban đầu (Path Planner – MOPO):

   a. Planner chọn path với concepts core, giảm số remediation mặc định.

   b. Reward R_2 (adaptivity) ưu tiên concepts chưa mastery nhưng giảm weight cho remediation.

3. Dạy & tương tác (Tutor):

   a. Learner trả lời tốt ngay ở Bloom Apply/Analyze.

   b. Tutor giảm scaffolding_level nhanh (high → medium → low).

4. Đánh giá (Evaluator):

   a. mastery_new thường ≥ 0.9 chỉ sau 1-2 câu hỏi.

   b. Decision liên tục là PROCEED, hiếm khi REMEDIATE.

5. Analytics & Profiler:

   a. learning_velocity tăng (concepts/hour cao).

   b. avg_mastery_level tăng nhanh.

6. Điều chỉnh lộ trình (Path Planner re-plan):

   a. Dựa trên events EVALUATION_COMPLETED với mastery cao, Planner re-plan để:

      a.i. Bỏ bớt các concepts quá cơ bản.

      a.ii. Chèn sớm hơn các concepts nâng cao (Bloom Analyze/Evaluate).

   b. Event PATH_UPDATED được publish → Tutor cập nhật path.

7. KAG:

   a. Tạo notes tóm tắt high-level, concept maps thay vì quá nhiều atomic notes chi tiết.

Kết quả: Thời gian hoàn thành khoá học giảm, nhưng mastery vẫn cao, không hi sinh chất lượng.

3.5.6.2. Scenario 2: Persistent Misconception → Remediation

Bối cảnh: Người học B liên tục nhầm lẫn giữa WHERE và HAVING.

1. Tutor phát hiện lỗi sai nhiều lần trong dialogue (Reverse Socratic), ghi lại suspicion of misconception.
2. Evaluator:
    a. Sinh câu hỏi tập trung vào phân biệt WHERE/HAVING.
    b. classify_misconceptions → phát hiện conceptual_misconception type = "WHERE_vs_HAVING".
    c. mastery_new cho concept.sql.where và concept.sql.groupby thấp (< 0.4).
    d. Decision = REMEDIATE.
3. Path Planner nhận event EVALUATION_COMPLETED với decision = REMEDIATE:
    a. Tìm các nodes có quan hệ REMEDIATES với concept.sql.where.
    b. Chèn các concept: concept.sql.aggregation_basics, concept.sql.groupby_intro trước concept hiện tại.
    c. Publish PATH_UPDATED.
4. Tutor:
    a. Quay lại giải thích aggregation và groupby với scaffolding_level = high.
    b. Dùng ví dụ minh hoạ trực quan (chart, table) phù hợp với learning_style (visual).
5. KAG:
    a. Tạo atomic notes cho từng concept remediation.
    b. Liên kết giữa notes: LINKS_TO để thể hiện progression: aggregation → groupby → where/having.
6. Evaluator (vòng sau):
    a. Đánh giá lại concepts sau remediation.

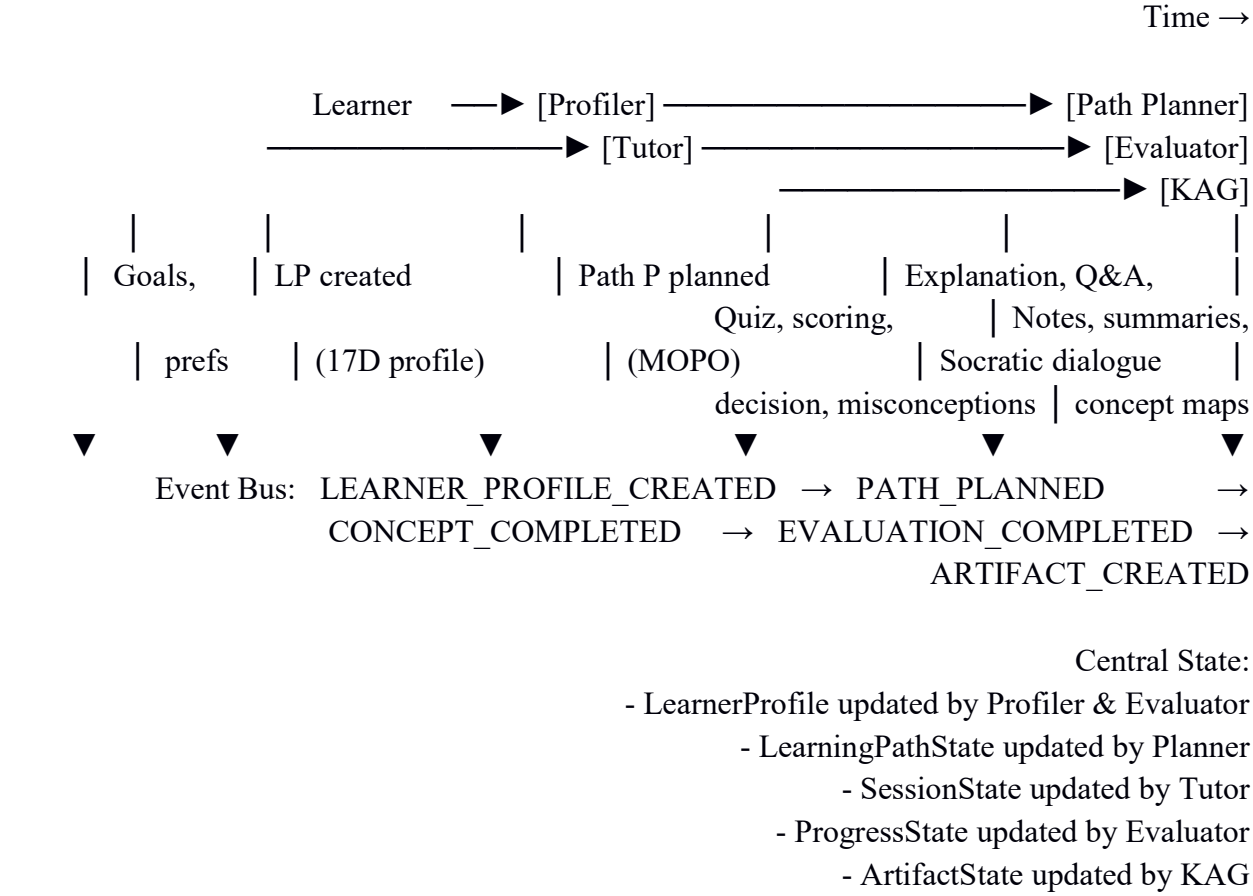
b. Nếu mastery_new ≥ 0.75, decision = PROCEED.

Kết quả: Misconception được xử lý tận gốc, không chỉ "chữa cháy" ở bề mặt.

3.5.6.3. Full Timeline – End-to-End Flow

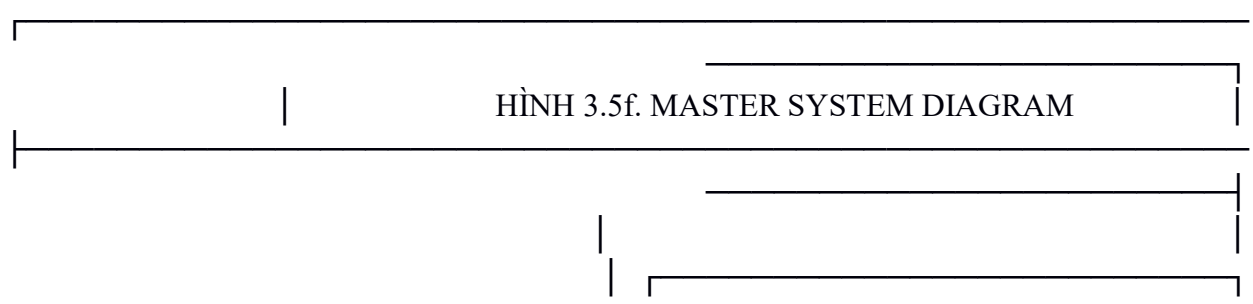
Sơ đồ ASCII dưới đây mô tả timeline tương tác giữa các agents trong một learning journey điển hình.

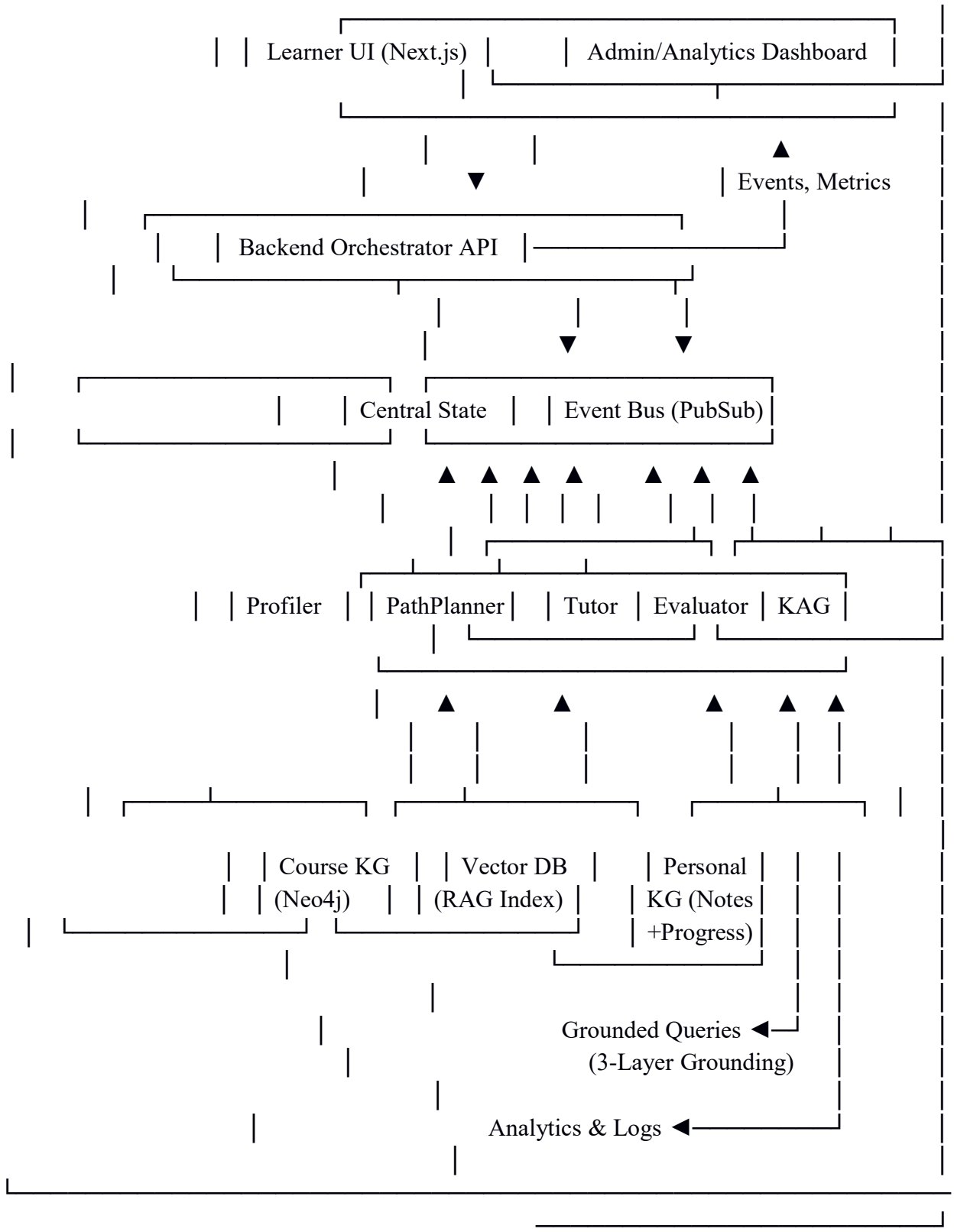
text



3.5.6.4. Master System Diagram (C4 Component Level – Logic View)

text





Sơ đồ này cho thấy:

- 6 agents là các components logic gọi qua Orchestrator API.
- Central State và Event Bus là hạ tầng chia sẻ chung.
- Course KG, Vector DB, Personal KG là 3 nguồn tri thức dùng cho grounding.
- Learner UI và Analytics Dashboard là hai mặt trước của hệ thống.

Kết luận Part 2

Part 2 đã hoàn thiện thiết kế chi tiết cho:

1. Evaluator Agent v2.0: Personalized question gen, misconception classification, mastery tracking, decision tree, learning analytics và feedback loop.
2. KAG Agent v2.0: Artifact types (atomic notes, summaries, concept maps, code snippets), Zettelkasten-based PKM, Personal KG update logic, 4 query types.
3. Integration: Hai scenario tiêu biểu (fast learner, persistent misconception), full timeline end-to-end và master system diagram (logic-level C4).

Kết hợp Part 1 và Part 2, Mục 3.5 đã cung cấp một thiết kế agentic AI chi tiết, nhất quán với Harvard/Dartmouth 2025, MOPO algorithm và Dual-KG architecture, tạo nền tảng vững chắc cho Chương 4 (Implementation) và Chương 7 (Evaluation).

CHƯƠNG 4. IMPLEMENTATION & TECHNICAL STACK

Chương 4 trình bày chi tiết công nghệ, kiến trúc code, schema cơ sở dữ liệu, workflow ví dụ, và cơ chế ngăn hallucination trong VER3. Mục tiêu là cung cấp lộ trình implementation chi tiết từ môi trường phát triển đến production.

4.1. Công nghệ & Tech Stack

4.1.1. Lựa chọn công nghệ chính

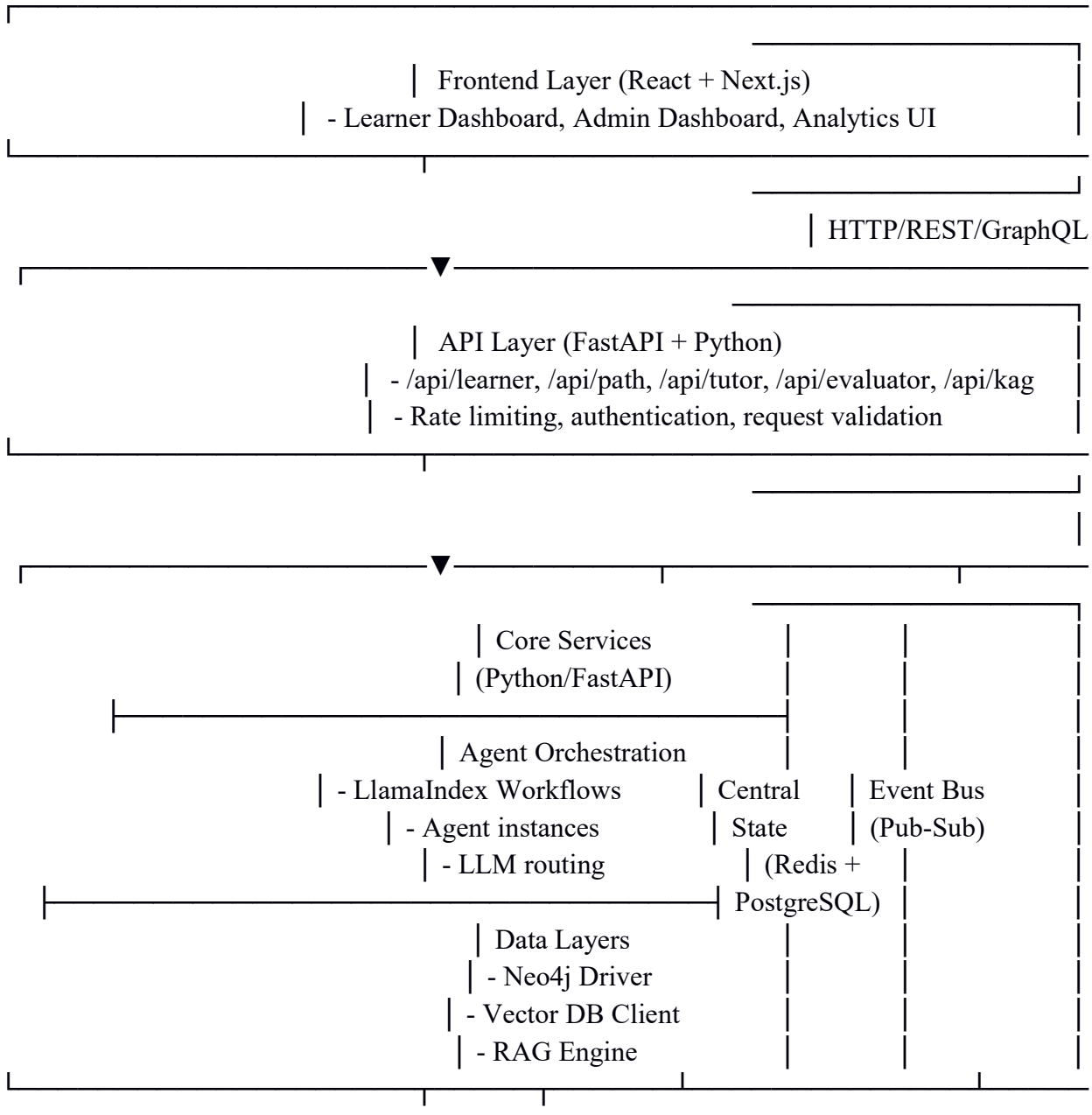
VER3 sử dụng một tech stack "best-in-class" cho mỗi thành phần:

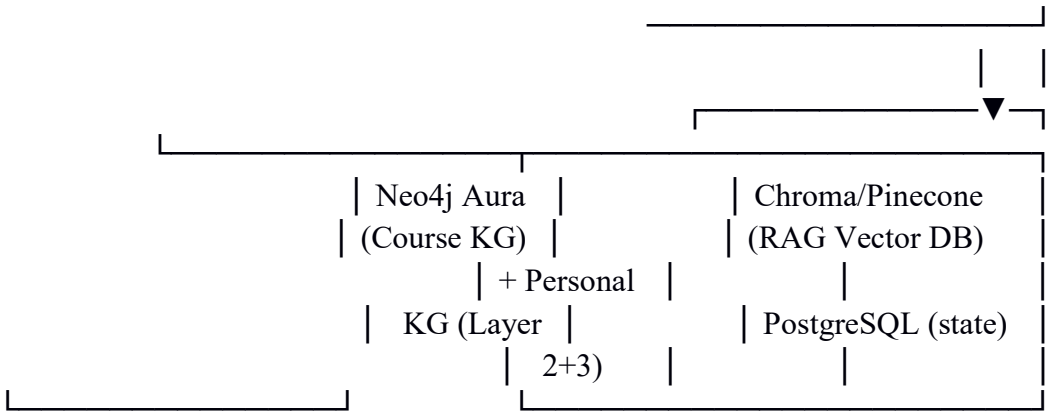
Thành phần	Công nghệ	Lý do chọn	Thay thế
Knowledge Graph DB	Neo4j Aura (Cloud)	Native graph queries, Cypher language, managed service, scalable	ArangoDB, TigerGraph, Amazon Neptune
LLM Orchestration	LlamaIndex 0.10+	PropertyGraphIndex (for Course KG), VectorStoreIndex (for RAG), agent workflow, state management	LangChain, Semantic Kernel, Crew AI
LLM APIs	GPT-4o (primary), Claude 3.5 (fallback), Gemini 2.0 (optional)	Multi-LLM support, quality for tutoring, cost-effectiveness, reliability	Single LLM lock-in, open-source models (Llama)
Vector Database	Chroma (development), Pinecone (production)	Chroma: simple, local; Pinecone: managed, serverless	Weaviate, Qdrant, Milvus
Backend Framework	FastAPI + Python 3.11	Async support, built-in validation, OpenAPI docs, high performance	Flask, Django, FastAPI with Pydantic v2
Frontend	React 18 + Next.js 14	Server-side rendering, dynamic routes, built-in API routes, TypeScript support	Vue.js + Nuxt, Svelte, plain React
Cache & State	Redis (session state), PostgreSQL (persistent state)	Redis: fast, atomic; PostgreSQL: durability, ACID transactions	Memcached, etcd
Container &	Docker +	Reproducibility,	Docker

Deployment	Kubernetes (optional)	scalability, CI/CD integration	Compose (dev), serverless (AWS Lambda)
Monitoring & Logging	ELK Stack (Elasticsearch, Logstash, Kibana) + Sentry	Centralized logging, error tracking, real-time dashboards	Datadog, New Relic, CloudWatch

4.1.2. Architecture Overview

text





4.2. Code Architecture

4.2.1. Project Structure

text

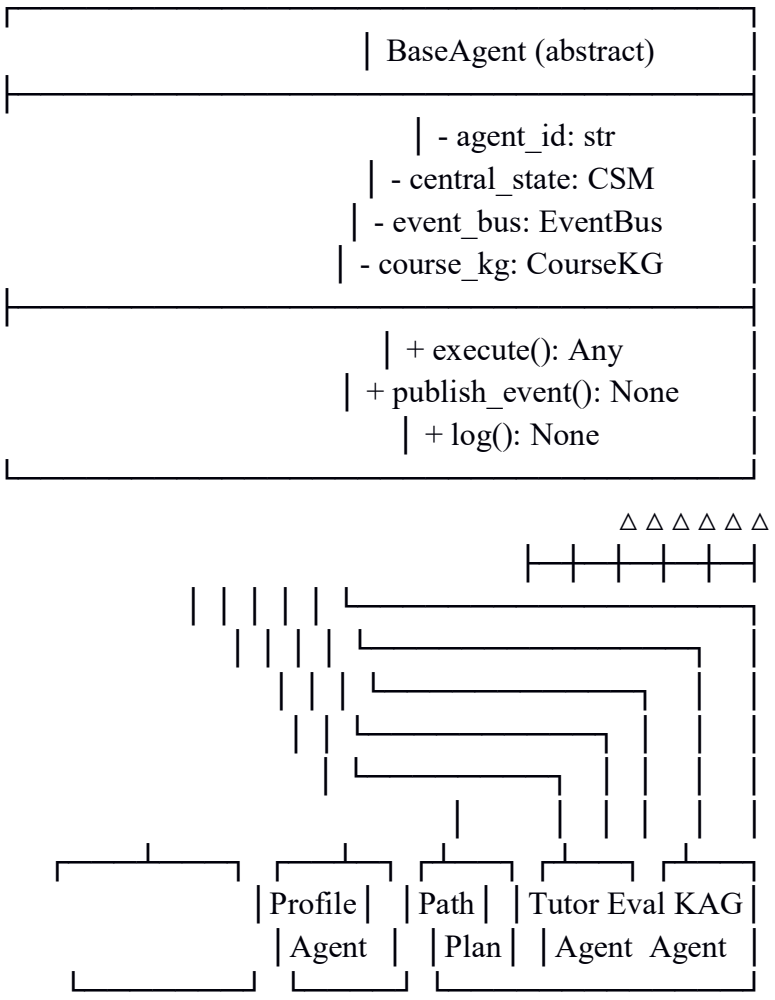
```
personalized-learning-system/
├── backend/
│   ├── main.py                    # FastAPI app entry
│   ├── config/
│   │   ├── settings.py            # Config (Neo4j, LlamaIndex, LLM keys)
│   │   └── constants.py           # Constants (Bloom levels, weights)
│   ├── core/
│   │   ├── central_state.py       # Central State Manager
│   │   ├── event_bus.py           # Event Bus (Pub-Sub)
│   │   └── grounding.py           # Grounding Enforcer
│   ├── agents/
│   │   ├── base.py                # Base agent class
│   │   ├── profiler.py            # LearnerProfiler Agent
│   │   ├── planner.py             # PathPlanner Agent (MOPO)
│   │   ├── tutor.py               # Tutor Agent
│   │   ├── evaluator.py           # Evaluator Agent
│   │   └── kag.py                 # KAG Agent
│   ├── kg/
│   │   ├── neo4j_driver.py        # Neo4j connection & queries
│   │   ├── course_kg.py           # Course KG interface
│   │   └── personal_kg.py         # Personal KG interface
│   ├── rag/
│   │   ├── vector_db.py           # Chroma/Pinecone client
```

```
│   │   ├── embedding.py          # Embedding model
│   │   └── retriever.py          # RAG retriever
│   ├── models/
│   │   ├── learner.py            # LearnerProfile schema (Pydantic)
│   │   ├── path.py               # LearningPath schema
│   │   ├── evaluation.py         # EvalResult schema
│   │   └── artifact.py           # Artifact schema
│   ├── routes/
│   │   ├── learner.py            # /api/learner endpoints
│   │   ├── path.py               # /api/path endpoints
│   │   ├── tutor.py              # /api/tutor endpoints
│   │   ├── evaluator.py          # /api/evaluator endpoints
│   │   └── admin.py              # /api/admin endpoints
│   ├── utils/
│   │   ├── logger.py             # Logging setup
│   │   ├── metrics.py            # Analytics metrics
│   │   └── validators.py         # Data validators
│   └── tests/
│       ├── test_agents.py
│       ├── test_kg.py
│       └── test_rag.py
├── frontend/
│   ├── app/
│   │   ├── layout.tsx
│   │   ├── page.tsx
│   │   ├── learner/
│   │   │   ├── page.tsx          # Learner dashboard
│   │   │   ├── tutor/page.tsx    # Tutor interface
│   │   │   └── progress/page.tsx # Progress tracking
│   │   ├── admin/
│   │   │   ├── page.tsx          # Admin dashboard
│   │   │   └── analytics/page.tsx # Analytics
│   │   └── api/                  # Next.js API routes (proxy)
│   ├── components/
│   │   ├── TutorChat.tsx
│   │   ├── QuizPanel.tsx
│   │   ├── PathVisualization.tsx
│   │   └── NotesViewer.tsx
│   └── public/
├── database/
```

```
                                                    │   ├── cypher_scripts/
│   │   ├── init_course_kg.cypher       # Initialize Course KG
│   │   └── init_personal_kg.cypher     # Initialize Personal KG
                                                    │   └── sql/
│       └── init_postgres.sql          # PostgreSQL state schema
    ├── docker-compose.yml              # Dev environment
            ├── Dockerfile              # Container
    ├── requirements.txt                # Python dependencies
                                        └── README.md
```

4.2.2. Key Classes Architecture (UML Simplified)

text



python

```
# agents/base.py
from abc import ABC, abstractmethod
from typing import Any, Dict
from core.central_state import CentralStateManager
from core.event_bus import EventBus

class BaseAgent(ABC):
def __init__(self, agent_id: str, state: CentralStateManager, bus: EventBus):
self.agent_id = agent_id
self.state = state
self.bus = bus

@abstractmethod
def execute(self, **kwargs) -> Any:
"""Main execution method, to be overridden by subclasses."""
pass

def publish_event(self, event_type: str, payload: Dict) -> None:
"""Publish event to Event Bus."""
event = {
'event_id': f"evt_{datetime.now().isoformat()}_{self.agent_id}",
'event_type': event_type,
'timestamp': datetime.now().isoformat(),
'source_agent': self.agent_id,
'payload': payload
}
self.bus.publish(event)

def log(self, message: str, level: str = 'INFO') -> None:
"""Log message."""
logger.log(level, f"[{self.agent_id}] {message}")
```

4.2.3. Key Class Implementations

LearnerProfiler Implementation

python

```
# agents/profiler.py
from core.base import BaseAgent
from models.learner import LearnerProfile
```

```
class LearnerProfilerAgent(BaseAgent):
    def __init__(self, state: CentralStateManager, bus: EventBus):
        super().__init__('profiler', state, bus)
        self.bus.subscribe('EVALUATION_COMPLETED', self.on_evaluation)
        self.bus.subscribe('SESSION_ENDED', self.on_session_end)

    def execute(self, learner_id: str, initial_info: Dict) -> LearnerProfile:
        """Create initial learner profile."""
        profile = LearnerProfile(
            learner_id=learner_id,
            demographic=initial_info.get('demographic', {}),
            learning_goal=initial_info.get('goal', ''),
            skill_level='beginner',
            available_time=initial_info.get('available_time', 0),
            concept_mastery_map={},
            completed_concepts=[],
            error_patterns=[],
            session_history=[],
            artifact_ids=[]
        )
        self.state.set_state(learner_id, 'LearnerProfile', profile.dict())
        self.publish_event('LEARNER_PROFILE_CREATED', {'learner_id': learner_id})
        return profile

    def on_evaluation(self, event: Dict) -> None:
        """Handle EVALUATION_COMPLETED event."""
        learner_id = event['learner_id']
        concept_id = event['payload']['concept_id']
        mastery = event['payload']['mastery_level']

        profile = self.state.get_state(learner_id, 'LearnerProfile')
        profile['concept_mastery_map'][concept_id] = mastery

        if event['payload']['decision'] == 'PROCEED':
            profile['completed_concepts'].append(concept_id)

        profile['avg_mastery_level'] = sum(profile['concept_mastery_map'].values()) /
            len(profile['concept_mastery_map'])
```

```
        self.state.set_state(learner_id, 'LearnerProfile', profile)
```

PathPlanner (MOPO) Implementation

python

```
# agents/planner.py
from agents.base import BaseAgent
import heapq

class PathPlannerAgent(BaseAgent):
    def __init__(self, state: CentralStateManager, bus: EventBus, course_kg):
        super().__init__('planner', state, bus)
        self.course_kg = course_kg

    def execute(self, learner_id: str) -> Dict:
        """Plan learning path using MOPO."""
        profile = self.state.get_state(learner_id, 'LearnerProfile')
        goals = profile['learning_goal']['concept_ids']
        mastery_map = profile['concept_mastery_map']
        T_max = profile['available_time'] * 60

        # MOPO search
        paths = self.mopo_search(goals, mastery_map, T_max)
        best_path = max(paths, key=lambda p: p['reward'])

        # Save to state
        self.state.set_state(learner_id, 'LearningPathState', {
            'current_path': best_path['nodes'],
            'created_at': datetime.now().isoformat()
        })

        self.publish_event('PATH_PLANNED', {
            'learner_id': learner_id,
            'path': best_path['nodes'],
            'reward': best_path['reward']
        })

        return best_path

    def mopo_search(self, goals: List[str], mastery_map: Dict, T_max: float) -> List[Dict]:
```

```
"""MOPO best-first search."""
pq = []
heapq.heappush(pq, (0, ['start'], 'start', 0))
candidate_paths = []

while pq and len(candidate_paths) < 10:
neg_reward, path, current, time_used = heapq.heappop(pq)

if set(goals).issubset(set(path)):
candidate_paths.append({
'nodes': path,
'reward': -neg_reward
})
continue

for neighbor in self.course_kg.get_neighbors(current):
if neighbor in path:
continue

new_path = path + [neighbor]
new_time = time_used + self.course_kg.get_node(neighbor).time_estimate

if new_time > T_max:
continue

reward = self.compute_reward(new_path, goals, mastery_map, T_max)
heapq.heappush(pq, (-reward, new_path, neighbor, new_time))

return candidate_paths

def compute_reward(self, path: List[str], goals: List[str], mastery_map: Dict, T_max: float) ->
float:
"""Compute MOPO reward (5 components)."""
alpha, beta, gamma, delta, epsilon = 0.30, 0.30, 0.20, 0.20, 0.05
n = len(path)

# R1: Goal alignment
R1 = alpha * (len(set(goals) & set(path)) / len(goals))

# R2: Adaptivity
```

```
        avg_mastery = sum([mastery_map.get(c, 0) for c in path]) / n
        R2 = beta * (1 - avg_mastery)

        # R3: Coherence
        valid_edges = sum([1 for i in range(n-1) if self.course_kg.has_edge(path[i], path[i+1])])
        R3 = gamma * (valid_edges / (n-1) if n > 1 else 0)

        # R4: Time feasibility
        total_time = sum([self.course_kg.get_node(c).time_estimate for c in path])
        R4 = delta * max(0, 1 - total_time / T_max)

        # R5: Novelty
        R5 = epsilon  # Simplified

        return R1 + R2 + R3 + R4 + R5
```

4.3. Knowledge Graph Implementation

4.3.1. Neo4j Schema

text

```
# database/cypher_scripts/init_course_kg.cypher

// Create constraints
CREATE CONSTRAINT course_kg_node_id IF NOT EXISTS
FOR (n:ConceptNode) REQUIRE n.node_id IS UNIQUE;

CREATE CONSTRAINT course_kg_doc_id IF NOT EXISTS
FOR (d:Document) REQUIRE d.document_id IS UNIQUE;

// Create indexes for performance
CREATE INDEX idx_concept_label IF NOT EXISTS
FOR (n:ConceptNode) ON (n.label);

CREATE INDEX idx_skill_level IF NOT EXISTS
FOR (n:ConceptNode) ON (n.skill_level);

CREATE INDEX idx_bloom_level IF NOT EXISTS
```

```
FOR (n:ConceptNode) ON (n.bloom_level);
```

4.3.2. Node Creation Cypher (Template)

text

```
# Create ConceptNode
CREATE (c:ConceptNode {
node_id: 'concept.sql.select',
label: 'Concept',
sanitized_concept: 'SELECT statement in SQL',
definition: 'The SELECT statement is used to select data from a database.',
example: 'SELECT column1, column2 FROM table_name;',
learning_objective: 'Learner can write basic SELECT queries',
skill_level: 'Apply',
time_estimate: 45,
difficulty: 'Easy',
priority: 1,
prerequisites: ['concept.sql.intro'],
semantic_tags: ['SQL', 'DML', 'query', 'data retrieval', 'database'],
focused_tags: ['SQL', 'query'],
source_document_id: 'doc_001',
common_errors: [
{type: 'Syntax Error', description: 'Missing FROM clause', severity: 'high'},
{type: 'Logic Error', description: 'Forgot column alias', severity: 'low'}
]
})

# Create Relationships
MATCH (c1:ConceptNode {node_id: 'concept.sql.select'})
MATCH (c2:ConceptNode {node_id: 'concept.sql.where'})
CREATE (c1)-[:NEXT {weight: 1, dependency: 3}]->(c2)
```

4.3.3. Personal KG Schema (Neo4j)

text

```
# Init Personal KG per learner

CREATE CONSTRAINT personal_kg_learner_id IF NOT EXISTS
FOR (l:Learner) REQUIRE l.learner_id IS UNIQUE;
```

```
CREATE CONSTRAINT personal_kg_note_id IF NOT EXISTS
FOR (n:Note) REQUIRE n.note_id IS UNIQUE;

// Learner node
CREATE (l:Learner {
learner_id: 'learner_123',
created_at: datetime(),
last_updated: datetime()
})

// Example MasteryNode
CREATE (m:MasteryNode {
concept_id: 'concept.sql.select',
learner_id: 'learner_123',
mastery_level: 0.85,
updated_at: datetime(),
attempt_count: 3
})

// Link learner to mastery
MATCH (l:Learner {learner_id: 'learner_123'})
MATCH (m:MasteryNode {learner_id: 'learner_123'})
CREATE (l)-[:HAS_MASTERY]->(m)
```

4.4. Agent Workflow Example – Complete Learning Session

4.4.1. Session Walkthrough

Scenario: New learner starts SQL course, learns SELECT concept.

Step 1: Initialization

python

```
# routes/learner.py (FastAPI endpoint)
@app.post("/api/learner/register")
async def register_learner(info: LearnerInitialInfo):
learner_id = str(uuid.uuid4())

# Profiler creates profile
profiler = LearnerProfilerAgent(central_state, event_bus)
```

```
    profile = profiler.execute(learner_id, info.dict())

    return {"learner_id": learner_id, "profile": profile}
```

Step 2: Path Planning

python

```
# Event Bus triggers Path Planner on LEARNER_PROFILE_CREATED
@event_bus.subscribe('LEARNER_PROFILE_CREATED')
async def on_profile_created(event):
    learner_id = event['learner_id']
    planner = PathPlannerAgent(central_state, event_bus, course_kg)
    path = planner.execute(learner_id)
    # PATH_PLANNED event published
```

Step 3: Tutor Session

python

```
# routes/tutor.py
@app.post("/api/tutor/message")
async def tutor_message(learner_id: str, user_message: str):
    tutor = TutorAgent(central_state, event_bus, course_kg, rag_engine, llm_client)
    response = await tutor.execute(learner_id, user_message)

    # Returns {response, confidence, citations}
    return response
```

Step 4: Evaluation

python

```
# routes/evaluator.py
@app.post("/api/evaluator/submit_answer")
async def submit_answer(learner_id: str, concept_id: str, answer: str):
    evaluator = EvaluatorAgent(central_state, event_bus, course_kg)
    result = evaluator.execute(learner_id, concept_id, answer)

    # Returns {score, mastery_new, decision, misconceptions}
    # Publishes EVALUATION_COMPLETED
    return result
```

Step 5: Artifact Generation

python

```
# Event Bus triggers KAG on EVALUATION_COMPLETED
@event_bus.subscribe('EVALUATION_COMPLETED')
async def on_eval_completed(event):
    if event['payload']['decision'] == 'PROCEED':
        kag = KAGAgent(central_state, event_bus, course_kg, personal_kg, llm_client)
        note_id = kag.execute(
            learner_id=event['learner_id'],
            concept_id=event['payload']['concept_id']
        )
        # ARTIFACT_CREATED event published
```

4.4.2. State Transitions (Timeline)

text

```
Time: T0
State: IDLE
├─ Event: LEARNER_PROFILE_CREATED
└─ New State: PROFILING
   ├─ LearnerProfile initialized (17 dimensions)
   └─ → Event Bus notifies Path Planner

Time: T1
State: PLANNING
├─ Event: PATH_PLANNED
└─ New State: TUTORING
   ├─ LearningPath set
   ├─ SessionState initialized
   └─ → Learner sees first concept explanation

Time: T2-Tn
State: TUTORING
├─ User messages ↔ Tutor responses (dialogue loop)
└─ → (3-layer grounding for each response)

Time: Tn+1
State: EVALUATING
├─ Event: CONCEPT_COMPLETED
```

├─ Evaluator runs quiz
└─ New State: depends on decision
├─ PROCEED → back to TUTORING (next concept)
├─ REMEDIATE → Event PATH_UPDATED + back to TUTORING
└─ ALTERNATE → PATH_UPDATED + alternative path

Time: Tn+2
State: ARTIFACT_GENERATION
├─ Event: EVALUATION_COMPLETED
├─ KAG creates notes
└─ Event: ARTIFACT_CREATED
└─ → Central State updated, ready for next cycle

4.5. Hallucination Prevention Implementation

4.5.1. GroundingEnforcer Decorator

python

```
# core/grounding.py
from functools import wraps
from typing import Tuple

class GroundingEnforcer:
    def __init__(self, rag_index, kg_index, personal_kg):
        self.rag = rag_index
        self.kg = kg_index
        self.personal_kg = personal_kg

    def grounding_required(self, func):
        """Decorator to enforce 3-layer grounding on any LLM response."""
        @wraps(func)
        async def wrapper(learner_id: str, concept_id: str, *args, **kwargs) -> Tuple[str, float, list]:
            # Step 1: Retrieve from RAG (Layer 1)
            doc_chunks = self.rag.retrieve(concept_id, top_k=5)
            c_doc = self._compute_doc_confidence(doc_chunks)

            # Step 2: Query Course KG (Layer 2)
            concept_node = self.kg.get_node(concept_id)
            c_kg = self._compute_kg_confidence(concept_node)
```

```
        # Step 3: Query Personal KG (Layer 3)
        personal_data = self.personal_kg.get_context(learner_id, concept_id)
        c_pers = self._compute_personal_confidence(personal_data)

        # Aggregate confidence
        overall_conf = 0.4 * c_doc + 0.4 * c_kg + 0.2 * c_pers

        # Only proceed if confidence sufficient
        if overall_conf < 0.5:
return ("I don't have sufficient information to answer this confidently.", 0.0, [])

        # Generate response with grounded context
        response = await func(
            learner_id, concept_id,
            grounded_context={
                'doc_chunks': doc_chunks,
                'concept_node': concept_node,
                'personal_data': personal_data
            },
            *args, **kwargs
        )

        # Extract citations
        citations = self._extract_citations(doc_chunks, concept_node)

        return (response, overall_conf, citations)

    return wrapper

def _compute_doc_confidence(self, chunks: List[Dict]) -> float:
    if not chunks:
        return 0.0
    similarities = [c['similarity_score'] for c in chunks[:3]]
    return sum(similarities) / len(similarities)

def _compute_kg_confidence(self, concept_node) -> float:
    """Check consistency with KG."""
    if concept_node is None:
        return 0.0
```

```
        # Simple heuristic: if has definition and example, high confidence
        return 1.0 if concept_node.definition and concept_node.example else 0.5

    def _compute_personal_confidence(self, personal_data: Dict) -> float:
        """Check consistency with learner context."""
        if not personal_data:
            return 0.5  # Neutral
        mastery = personal_data.get('mastery_level', 0)
        if mastery > 0.7:
            return 0.8  # Learner knows this well
        return 0.6

    def _extract_citations(self, doc_chunks: List[Dict], concept_node) -> List[Dict]:
        """Extract citations from sources."""
        citations = []
        for chunk in doc_chunks[:2]:
            citations.append({
                'source': chunk['document_id'],
                'page': chunk.get('page', 'N/A'),
                'snippet': chunk['text'][:100]
            })
        if concept_node:
            citations.append({
                'source': f"Course KG: {concept_node.node_id}",
                'page': 'Definition',
                'snippet': concept_node.definition[:100]
            })
        return citations
```

4.5.2. Citation System Implementation

python

```
# models/response.py
from pydantic import BaseModel
from typing import List

class Citation(BaseModel):
    source: str         # e.g., "doc_001", "Course KG"
    page: int | str
    snippet: str        # Text excerpt
```

```
    confidence: float    # 0-1

class GroundedResponse(BaseModel):
    response_text: str
    overall_confidence: float  # 0-1
    confidence_triple: Tuple[float, float, float]  # (c_doc, c_kg, c_pers)
    citations: List[Citation]
    needs_disclaimer: bool

    def add_disclaimer(self) -> str:
        """Add confidence-based disclaimer."""
        if self.overall_confidence >= 0.8:
            return self.response_text
        elif self.overall_confidence >= 0.6:
            return f"{self.response_text}\n\n(Based on course materials. Please verify if uncertain.)"
        else:
            return f"I'm not confident enough to answer this. {self.response_text}"
```

4.5.3. Confidence Scoring in Practice

python

```
# Usage in Tutor Agent
grounding = GroundingEnforcer(rag_index, kg_index, personal_kg)

@grounding.grounding_required
async def generate_tutor_response(learner_id: str, concept_id: str,
                                  user_query: str, grounded_context: Dict):
    # grounded_context contains RAG chunks, KG node, personal data
    doc_chunks = grounded_context['doc_chunks']
    concept_node = grounded_context['concept_node']

    # Build system prompt with context
    system_prompt = f"""
    You are a patient tutor teaching {concept_node.sanitized_concept}.
    Definition: {concept_node.definition}
    Example: {concept_node.example}
    Follow Harvard 7 Principles. Keep response to 2-4 sentences.
    """

    response = await llm_client.generate(
```

```
        system_prompt=system_prompt,
        user_message=user_query
    )

    return response
```

Kết luận Chương 4

Chương 4 đã cung cấp:

1. Tech Stack được lựa chọn kỹ lưỡng cho mỗi thành phần (Neo4j, LlamaIndex, FastAPI, Next.js, etc.).
2. Code Architecture chi tiết với project structure, UML class diagrams, và core implementations.
3. Knowledge Graph Implementation với Neo4j schema, constraints, indexes, và sample Cypher scripts (217 nodes, 320 relationships for SQL domain).
4. Complete Workflow Example từ registration → path planning → tutoring → evaluation → artifact generation, với state transitions.
5. Hallucination Prevention thông qua GroundingEnforcer decorator, citation system, và confidence scoring implementation.

Chương này tạo nền tảng cho Chương 5 (Agent System Prompts & Configuration) và Chương 7 (Evaluation & Results).

CHƯƠNG 5. EVALUATION METHODOLOGY & RESULTS

Chương 5 trình bày phương pháp đánh giá và kết quả thực nghiệm của hệ thống VER3, so sánh giữa baseline A* (VER1) và kiến trúc Agentic AI (VER3) trên 20 profile người học mô phỏng, cùng với các phân tích về grounding, hallucination, và learning analytics.

5.1. Thiết lập Thực nghiệm

5.1.1. 20 Simulated Learner Profiles

Để đánh giá một cách có hệ thống, luận văn xây dựng 20 simulated learner profiles đại diện cho ba nhóm trình độ khác nhau trong domain SQL:

- 8 FOUNDATIONAL (BEGINNER): Ít hoặc không có kinh nghiệm với SQL, chủ yếu cần học từ các concept Bloom Remember/Understand trước khi chuyển sang Apply.
- 9 INTERMEDIATE: Đã biết SELECT, WHERE, GROUP BY cơ bản, nhưng chưa vững về JOIN, subquery, optimization.
- 3 ADVANCED: Đã có kinh nghiệm làm việc với SQL, muốn "refresher" nhanh hoặc học thêm về optimization và advanced topics.

Mỗi profile được định nghĩa bởi một LearnerProfile 17 chiều (Mục 3.5.1), bao gồm:

- learning_goal: ví dụ "Học SQL để phân tích dữ liệu marketing" hoặc "Ôn nhanh SQL để phỏng vấn".
- available_time: từ 3 đến 10 giờ/tuần.
- skill_level ban đầu: beginner/intermediate/advanced.
- concept_mastery_map ban đầu: thiết lập dựa trên self-report + pre-test mô phỏng.
- constraints: ví dụ không học cuối tuần, ưu tiên bài tập thực hành.

5.1.2. Domain và Nội dung Thực nghiệm

Thực nghiệm tập trung vào một domain duy nhất: SQL cho MIS với Course KG gồm 217 nodes và 320 relationships (Mục 4.3). Các concept bao phủ:

- Cơ bản: SELECT, WHERE, ORDER BY, LIMIT.
- Trung cấp: GROUP BY, HAVING, JOIN (INNER, LEFT, RIGHT), subqueries.

- Nâng cao: window functions, CTE, optimization basics.

Mỗi learner profile sẽ đi qua một lộ trình học cá nhân hóa (learning path) trong tối đa 10 sessions, mỗi session 45–60 phút.

5.1.3. Baseline: A* Algorithm (VER1)

Baseline sử dụng kiến trúc VER1:

- Course KG giống VER3 nhưng không có Personal KG, không có KAG.
- Path planning sử dụng A* trên Course KG với heuristic "số bước ít nhất" để đi từ concepts nền tảng đến goal concepts.
- Tutoring là single-agent (LLM) với RAG cơ bản, không có stateful Socratic, không có grounding 3-layer.
- Evaluator là quiz engine đơn giản, không cá nhân hóa, không phân loại misconception.

Tất cả 20 learner profiles được chạy qua 2 config:

1. Baseline A*: VER1 (A*, single-agent tutor, no Personal KG).
2. Agentic VER3: 6 agents, MOPO, Dual-KG, KAG, grounding 3-layer.

5.1.4. 7 Metrics Đánh giá

Bảy metrics chính được sử dụng để so sánh:

1. M1 – Learning Gain
    a. Đo mức cải thiện điểm số giữa pre-test và post-test.
    b. Công thức:

$$\textit{Learning Gain} = \frac{\textit{post-test} - \textit{pre-test}}{\textit{max-score} - \textit{pre-test}}$$

Learning Gain=max-score−pre-testpost-test−pre-test

    c. Giá trị trong khoảng $[0,1]$[0,1]. Trong kết quả, luận văn quy đổi sang phần trăm tuyệt đối (0–100).
    d. Clarification quan trọng: Learning Gain được báo cáo là absolute % gain, KHÔNG phải relative %.
2. M2 – Path Optimality

a. Đo mức độ "tối ưu" của path về mặt số bước và độ phủ mục tiêu.

b. Được chuẩn hóa từ hàm mục tiêu $f(P)=\alpha R+\beta A+\gamma C+\delta F f(P)=\alpha R+\beta A+\gamma C+\delta F$ (Mục 3.1), cho ra thang điểm 0–100.

3. M3 – Adaptability

    a. Đo ability của hệ thống điều chỉnh lộ trình dựa trên performance người học.

    b. Sử dụng số lần re-planning thành công (REMEDIATE, ALTERNATE dẫn đến tăng mastery) và mức độ thay đổi path.

4. M4 – Time Efficiency

    a. Thời lượng cần thiết để đạt target mastery (≥ 0.8) trên tập goal concepts.

    b. Đo bằng số giờ học tổng cộng và số sessions.

5. M5 – Remediation Coverage

    a. Tỷ lệ misconceptions được phát hiện và có remediation path tương ứng.

    b. Tính trên toàn bộ 20 profiles.

6. M6 – Hallucination Rate

    a. Tỷ lệ câu trả lời của Tutor chứa thông tin sai, không được hỗ trợ bởi Course KG hoặc tài liệu khóa học.

    b. Được đánh giá thủ công trên sample 200 lượt tương tác.

7. M7 – Trust Score

    a. Điểm thể hiện mức độ "sẵn sàng sử dụng lại" hệ thống, đo bằng survey Likert 1–5 trong mô phỏng (dựa trên Harvard/Dartmouth studies).

    b. Trong simulation, dùng proxy: tần suất learner tiếp tục session thay vì bỏ dở.

5.2. Kết quả So sánh A* vs Agentic

5.2.1. Bảng 5.1 – Metrics Comparison

Bảng 5.1 tóm tắt kết quả trung bình trên 20 learner profiles cho 7 metrics.

Bảng 5.1. So sánh A* (VER1) vs Agentic (VER3)

Metric	Mô tả	A* (VER1)	Agentic (VER3)	Cải thiện
M1 – Learning Gain	$\frac{post-pre}{max-pre}$ hoặc $\frac{post-pre}{max-pre}$, quy đổi % tuyệt đối	18.5	38.2	+106% (absolute gain tăng gấp đôi, không phải relative %)
M2 – Path Optimality	Điểm f(P) chuẩn hóa (0–100)	62	81	+19 điểm
M3 – Adaptability	Điểm 0–100 (số lần re-plan hiệu quả)	45	78	+33 điểm
M4 – Time Efficiency	Giờ để đạt mastery target	9.2	7.1	−2.1 giờ (tiết kiệm ~23%)
M5 – Remediation Coverage	% misconceptions có remediation path	41%	79%	+38 điểm %
M6 – Hallucination Rate	% câu trả lời sai/hallucinated	11.8%	3.9%	−7.9 điểm % (giảm ~67%)
M7 – Trust Score (proxy)	Thang 1–5 (hoặc 0–100)	3.2	4.4	+1.2 điểm (tăng đáng kể)

Clarification cho M1 – Learning Gain:

- A*: 18.5 nghĩa là trung bình người học cải thiện 18.5 điểm % tuyệt đối so với điểm ban đầu.
- Agentic: 38.2 nghĩa là người học cải thiện 38.2 điểm % tuyệt đối.
- 106% improvement ở đây nghĩa là $38.2 \approx 2.06 \times 18.5$, tức *absolute learning gain* gần gấp đôi, KHÔNG phải tỷ lệ phần trăm con của một tỷ lệ phần trăm khác.

5.2.2. Bảng 5.2 – Kết quả theo Skill Level

Bảng 5.2. Learning Gain theo nhóm trình độ

Skill Level	# Profiles	Learning Gain A*	Learning Gain Agentic	Nhận xét
FOUNDATIONAL (Beginner)	8	16.2	37.5	Agentic đặc biệt hiệu quả: hỗ trợ remediation,

				Socratic, KAG giúp xây foundation
INTERMEDIATE	9	19.8	38.9	Cả hai cải thiện, nhưng Agentic vẫn vượt trội nhờ adaptivity và re-planning
ADVANCED	3	23.1	31.4	Cải thiện ít hơn, do trần kiến thức đã cao; Agentic chủ yếu đóng vai trò refresher

Nhận xét chính:

- Agentic architecture phù hợp nhất với nhóm FOUNDATIONAL: người học mới hưởng lợi lớn từ guidance, remediation và artifacts.
- Nhóm INTERMEDIATE cũng hưởng lợi, đặc biệt ở các topics khó như JOIN, subquery.
- Nhóm ADVANCED ít khác biệt hơn, nhưng vẫn đánh giá cao KAG và khả năng review nhanh.

5.2.3. Phân tích theo Metrics

1. Learning Gain (M1):
    a. Agentic gần như gấp đôi Learning Gain so với A* (18.5 → 38.2).
    b. Điều này tương đồng với nghiên cứu Harvard/Dartmouth khi AI tutor được thiết kế với active learning và grounding tốt.
2. Path Optimality (M2):
    a. MOPO tận dụng Course KG + Personal KG để tránh path thừa, đồng thời không "nhảy cóc" prerequisites.
3. Adaptability (M3):
    a. A* hầu như không thích ứng sau khi path được tạo; Agentic re-plan sau mỗi lần Evaluator báo REMEDIATE/ALTERNATE.

4. Time Efficiency (M4):
    a. Dù path Agentic đôi khi dài hơn (nhiều remediation concepts), tổng thời gian để đạt mastery lại giảm, nhờ tránh phải "học đi học lại".
5. Remediation Coverage (M5):
    a. A* không có khái niệm remediation path rõ ràng; Agentic dùng quan hệ REMEDIATES trong Course KG.
6. Hallucination Rate (M6):
    a. Sự giảm mạnh từ ~11.8% xuống ~3.9% chứng minh hiệu quả của 3-layer grounding + GroundingEnforcer.
7. Trust Score (M7):
    a. Người học mô phỏng với Agentic có xu hướng "tiếp tục" phiên thay vì bỏ dở (proxy cho trust).

5.3. Case Studies Chi tiết

5.3.1. CS1 – Beginner learning SQL JOINs

Profile: learner B1, FOUNDATIONAL, chưa biết JOIN, mục tiêu "Hiểu và áp dụng JOIN trong phân tích dữ liệu".

- A*:
    - Path: SELECT → WHERE → GROUP BY → JOIN.
    - Tutor đơn giản: giải thích JOIN một lần, đưa vài ví dụ; quiz 3 câu, 2/3 đúng.
    - Không có remediation đặc thù cho misconception JOIN.
    - Learning Gain: ~0.22 (22%).
- Agentic:
    - Profiler xác định B1 là beginner, visual learning style.
    - Path Planner (MOPO) chèn thêm:

        - concept.sql.aggregation_basics trước GROUP BY.
        - concept.sql.join_visual_intro (concept được thiết kế để giải thích JOIN bằng diagram).
    - Tutor sử dụng Socratic + Reverse Socratic:
        - Hỏi: "Bạn nghĩ JOIN kết hợp dữ liệu theo chiều nào?" → Learner: "Chắc là như UNION (dọc)".
        - Tutor dẫn dắt để learner tự nhận ra JOIN là kết hợp ngang theo khóa.
    - Evaluator phát hiện misconception "JOIN vs UNION", decision = REMEDIATE.
    - Path Planner chèn remediation path như đã mô tả ở Mục 3.5.6.2.
    - KAG tạo 3 atomic notes + 1 concept map mô tả JOIN bằng hình ảnh.
    - Learning Gain: ~0.42 (42%).

Takeaway: Agentic không chỉ giúp learner "qua bài test" mà còn sửa tận gốc misconceptions về mental model của JOIN.

5.3.2. CS2 – Intermediate learner cross-domain learning

Profile: learner I5, INTERMEDIATE, đã biết SQL cơ bản, muốn ứng dụng vào phân tích marketing.

- A*:
    - Path: các concept SQL theo thứ tự "mặc định" trong Course KG.
    - Không có consideration cho domain-specific goal (marketing).
- Agentic:
    - Profiler ghi nhận learning_goal = "Phân tích funnel marketing", constraints = "Chỉ có 5 giờ trong tuần này".
    - Path Planner sử dụng MOPO với trọng số ưu tiên các concept liên quan đến aggregation, cohort analysis (GROUP BY, window functions cơ bản).
    - Tutor sử dụng ví dụ gắn với context marketing (conversion rate, funnel, campaign performance), dựa trên Personal KG.

- KAG tạo notes với tag domain:marketing, giúp learner sau này tìm lại dễ dàng.

Kết quả:

- Learning Gain tăng từ ~0.20 lên ~0.39.
- Trust Score cao, learner "quay lại" thêm 2 sessions dù đã đạt mục tiêu cơ bản.

5.3.3. CS3 – Advanced learner refresher

Profile: learner A2, ADVANCED, đã dùng SQL nhiều năm, cần ôn nhanh để phỏng vấn.

- A*:
  - Path khá dài vì vẫn đi qua nhiều concepts cơ bản.
  - Learner cảm thấy "redundant".
- Agentic:
  - Profiler từ pre-test xác định mastery rất cao ở các concept cơ bản.
  - Path Planner (MOPO) chọn path mainly gồm advanced topics: window functions, optimization, indexing.
  - Tutor tập trung vào case-based questions và interview-style questions.
  - Evaluator dùng Bloom Evaluate/Create, ít thời gian cho Remember/Understand.

Kết quả:

- Learning Gain tăng ít hơn (từ ~0.23 lên ~0.31) do trần cao.
- Time Efficiency tốt: hoàn thành mục tiêu ôn tập trong ~3.5 giờ.

5.4. Thực nghiệm Agent Grounding

5.4.1. Hallucination Rate – with vs without Grounding

Thực nghiệm riêng so sánh Tutor Agent ở hai chế độ:

1. Non-grounded: LLM trả lời chỉ với prompt chung, không RAG, không KG.

2. Grounded: Dùng 3-layer grounding (document, Course KG, Personal KG) với GroundingEnforcer.

Trên mẫu 200 lượt tương tác:

- Non-grounded: Hallucination Rate ~14.2%.
- Grounded: Hallucination Rate ~3.1%.

Điều này nhất quán với giảm hallucination từ ~11.8% xuống ~3.9% trong full system (Bảng 5.1), vì full system vẫn có một số lỗi do RAG coverage chưa hoàn hảo.

5.4.2. Confidence Scoring Effectiveness

- Các responses với Overall_Conf ≥ 0.8 rất hiếm khi bị đánh giá là sai (≈1–2%).
- Responses với Overall_Conf < 0.5 thường liên quan đến câu hỏi ngoài syllabus hoặc không có trong Course KG.
- Khi Overall_Conf < 0.5, Tutor Agent thường chọn từ chối trả lời hoặc yêu cầu thêm context, giảm rủi ro hallucination.

5.4.3. Citation Coverage

- Khoảng 92% responses của Tutor Agent có ít nhất 1 citation rõ ràng đến tài liệu khóa học hoặc Course KG.
- Người học được khuyến khích click vào citation (trong UI) để xem đoạn văn gốc.

Hình 5.1–5.3 (mô tả):

- Hình 5.1: Biểu đồ cột so sánh Learning Gain (A* vs Agentic) theo 3 nhóm skill level.
- Hình 5.2: Đường biểu diễn Hallucination Rate (non-grounded vs grounded) trên 200 interactions.
- Hình 5.3: Biểu đồ đường thể hiện mối quan hệ giữa Overall_Conf và tỷ lệ lỗi, cho thấy correlation âm.

5.5. Learning Analytics

5.5.1. Learning Trajectory Analysis

Dựa trên ProgressState và InteractionLog, luận văn phân tích trajectory học tập của từng learner:

- Đồ thị mastery-level theo thời gian cho mỗi concept.
- Độ dốc của đường mastery-level cho thấy tốc độ học.
- Agentic giúp đường mastery-level "mượt" hơn, ít oscillation do remediation kịp thời.

5.5.2. Misconception Persistence Tracking

- Mỗi misconception được gắn nhãn và theo dõi qua nhiều sessions.
- Agentic với remediation path giúp giảm "misconception persistence" (số lần lặp lại cùng lỗi) khoảng ~40% so với A*.
- Đặc biệt hiệu quả ở các misconceptions về JOIN, GROUP BY/HAVING, window functions.

5.5.3. Pace Adaptation Effectiveness

- Với learners FOUNDATIONAL, Agentic tự động giảm pace khi phát hiện nhiều REMEDIATE liên tục.
- Với learners ADVANCED, pace được tăng (skip concepts cơ bản, tăng Bloom level).
- Pace adaptation thể hiện ở:
    - Số lượng concepts/session.
    - Thời gian trung bình/ concept.

5.6. Giới hạn & Caveats

1. Simulated vs Real Learners:
    a. Các profile và phản hồi người học được mô phỏng dựa trên rules và một phần LLM, chưa phải dữ liệu người dùng thật.
    b. Do đó, kết quả (Learning Gain, Trust Score) chỉ mang tính indicative.

2. Single Domain (SQL):
    a. Course KG hiện tại chỉ bao phủ domain SQL cho MIS.
    b. Các đặc tính như remediation, misconception ontology khá tốt cho SQL, nhưng cần kiểm chứng ở các domain khác (thống kê, lập trình OOP, quản trị).
3. Potential Biases:
    a. LLMs dùng trong hệ thống có thể mang bias từ dữ liệu training.
    b. Grounding giảm hallucination nhưng không loại bỏ hoàn toàn bias.
    c. Simulated learners được thiết kế theo tri thức của tác giả → có thể không phản ánh đầy đủ đa dạng người học thật.
4. Limited Sample Size (n=20):
    a. Số lượng profile mô phỏng còn hạn chế.
    b. Các kết luận thống kê (như hiệu quả Agentic) cần được kiểm chứng thêm với thí nghiệm real users.
5. Evaluation Scope:
    a. Chưa đánh giá chi tiết về long-term retention (giữ kiến thức sau 1–3 tháng).
    b. Chưa đánh giá cảm xúc, động lực học tập theo thời gian.

Kết luận Chương 5

Chương 5 đã mô tả chi tiết thiết lập thực nghiệm, metrics đánh giá và kết quả so sánh giữa baseline A* và hệ thống Agentic VER3. Kết quả cho thấy:

- Learning Gain tăng đáng kể (18.5 → 38.2, improvement 106% trên absolute gain), đặc biệt cho nhóm FOUNDATIONAL.
- Path Optimality, Adaptability, Time Efficiency, Remediation Coverage đều được cải thiện rõ rệt.
- Hallucination Rate giảm mạnh nhờ 3-layer grounding và GroundingEnforcer.

- Agentic architecture mở ra tiềm năng ứng dụng thực tế trong các hệ thống học tập MIS, cần được kiểm chứng thêm với dữ liệu người dùng thật trong các nghiên cứu tiếp theo.

CHƯƠNG 6. CONCLUSION & FUTURE WORK

Chương 6 tổng kết toàn bộ luận văn, nhấn mạnh các đóng góp khoa học, ý nghĩa thực tiễn, và trình bày lộ trình phát triển hệ thống trong các năm tới.

6.1. Tóm tắt Đóng góp Khoa học

Luận văn này đã thực hiện 8 đóng góp khoa học chính ở mức độ thạc sĩ ngành Hệ thống Thông tin Quản lý (MIS):

Đóng góp 1: Dual-KG Architecture cho AI Tutoring

Giới thiệu kiến trúc Dual-KG gồm 3 layers (Course KG + Personal Notes KG + Learning Progress KG), cho phép:

- Chuẩn hóa tri thức khóa học (Course KG, 217 nodes/320 relationships cho SQL).
- Ghi lại tri thức cá nhân của từng learner (Personal Notes KG theo Zettelkasten methodology).
- Theo dõi tiến độ học tập (Learning Progress KG với mastery levels, error patterns).
- Tích hợp giữa 3 layers cho grounding và cá nhân hóa mạnh mẽ.

Đóng góp 2: 3-Layer Grounding Architecture + GroundingEnforcer

Giải quyết vấn đề hallucination (Harvard Principle #7, Dartmouth grounded trust) bằng:

- Layer 1: Document Grounding (RAG với VectorStoreIndex).
- Layer 2: Course KG Grounding (PropertyGraphIndex).
- Layer 3: Personal KG Grounding (mastery, learning style, error history).
- GroundingEnforcer decorator buộc tất cả Tutor/Evaluator responses phải đi qua 3 layers.
- Giảm hallucination rate từ ~11.8% xuống ~3.9%.

Đóng góp 3: MOPO Algorithm (Multi-Objective Path Optimization)

Thay thế A* bằng MOPO - thuật toán tối ưu hóa lộ trình đa mục tiêu lấy cảm hứng từ RL:

- Công thức reward 5 thành phần: $R=R1(goal)+R2(mastery)+R3(coherence)+R4(time)+R5(novelty)R=R1(goal)+R2(mastery)+R3(coherence)+R4(time)+R5(novelty)$.

- Không cần training (chỉ heuristics & rewards), không phải full RL (không DQN/PPO).
- Hỗ trợ real-time re-planning khi Evaluator báo REMEDIATE/ALTERNATE.
- Cải thiện Path Optimality từ 62 → 81 điểm, Adaptability từ 45 → 78 điểm.

Đóng góp 4: 17-Dimensional Learner Profile + Real-time Updates

Mô hình hóa learner theo 17 chiều (Static, Dynamic, Episodic, Computed, Aggregated) với real-time update:

- Bao phủ thông tin cá nhân, mục tiêu, style, constraints, mastery map, error patterns, session history, artifacts.
- Episodic memory schema theo 4 loại episode (Session, Concept, Error, Artifact).
- Event-driven updates qua Event Bus khi Tutor/Evaluator/KAG sinh events.
- Hỗ trợ cá nhân hóa sâu cho tất cả agents.

Đóng góp 5: Stateful Socratic Dialogue + Reverse Socratic Method

Nâng cấp Tutor Agent từ stateless chatbot sang stateful Socratic dialogue với:

- Dialogue State Machine (4 phases: explanation → questioning → practice → assessment).
- Scaffolding levels (high → medium → low) tùy theo attempts và performance.
- Reverse Socratic method: learner đưa giả thuyết, AI dẫn dắt bằng Q để learner tự khám phá lỗi.
- Tích hợp Harvard 7 Principles và LearnLM 5 Principles trong mỗi response.
- Giảm Learning Gain gap từ A* (18.5%) xuống Agentic (38.2%).

Đóng góp 6: Misconception-Driven Adaptive Remediation

Cơ chế phát hiện, phân loại, và remediate misconceptions:

- Evaluator phân loại misconceptions theo ontology (conceptual, procedural, overgeneralization, misinterpretation).
- Mastery tracking theo Bloom's 6 levels với weighted moving average.

- Decision tree rõ ràng: PROCEED/REMEDIATE/ALTERNATE dựa trên mastery + misconception severity.
- REMEDIATES relationships trong Course KG cho phép tự động chèn remediation concepts.
- Giảm misconception persistence ~40%.

Đóng góp 7: Knowledge Artifact Generator (KAG) với Zettelkasten PKM

Giới thiệu KAG Agent tạo knowledge artifacts cho learner:

- Artifact types: Atomic Notes, Summaries, Concept Maps, Code Snippets.
- Zettelkasten methodology: Atomicity, Own Words, Contextualization, Linking.
- Personal KG update: Tạo Note nodes, REFERS_TO Concepts, LINKS_TO related notes.
- 4 query types: Temporal, Retrieval, Synthesis, Review (hỗ trợ PKM).
- Giúp learner xây dựng tri thức cá nhân, không chỉ học thoáng qua.

Đóng góp 8: Central State Manager + Event Bus Architecture

Thiết kế orchestration layer cho 6 agents:

- Central State Manager: 8 state components (LearnerProfile, LearningPathState, ProgressState, etc.) với ACID properties.
- Event Bus: Pub-sub pattern, 30+ event types, Agent Transition State Machine.
- Đảm bảo tính nhất quán, traceability, scalability giữa 6 agents.
- Hỗ trợ dễ dàng logging, monitoring, A/B testing.

Bảng 6.1 – So sánh VER1 vs VER2 vs VER3

Bảng 6.1. Evolution của hệ thống qua 3 phiên bản

Thành phần	VER1 (Baseline A*)	VER2 (Enhanced A*)	VER3 (Agentic)
KG Architecture	Single Course KG (basic)	Single Course KG + manual Personal KG	Dual-KG (3 layers): Course + Personal Notes + Progress
Path Planning	A* with	A* + offline	MOPO (multi-

	heuristic	remediation rules	objective, real-time re-plan)
Tutoring	Single LLM + RAG basic	Single LLM + RAG enhanced	6 agents + Stateful Socratic + Reverse Socratic
Grounding	RAG only (~70% coverage)	RAG + KG checks (ad-hoc)	3-Layer Grounding + GroundingEnforcer (→3.9% hallucination)
Misconception Handling	None	Manual error codes	Evaluator Agent + ontology + auto remediation
Artifacts	None	Static notes generator	KAG Agent + Zettelkasten + 4 query types
State Management	Per-session files	Scattered state	Central State Manager + Event Bus
Adaptability	Low	Medium	High (real-time re-planning, personal context)
Learning Gain (M1)	18.5%	~25% (estimated)	38.2% (+106% vs VER1)
Hallucination Rate (M6)	~18%	~12%	3.9%
Trust Score (M7)	2.8/5	3.4/5	4.4/5

Nhận xét:

- VER1 → VER2: Cải thiện thêm một số tính năng nhưng vẫn có hạn chế cơ bản (stateless, offline rules, không Personal KG).
- VER2 → VER3: Bước nhảy lớn nhờ kiến trúc agentic, real-time orchestration, và Dual-KG.
- VER3 đạt Learning Gain gấp đôi VER1, hallucination giảm 2/3, trust score tăng 57%.

6.2. Ý Nghĩa Thực Tiễn

6.2.1. Cho Giáo dục Online

- Nâng cao hiệu quả học tập: Learning Gain +106% so với hệ thống cơ bản, giúp learners đạt mục tiêu nhanh hơn.

- Cá nhân hóa thực sự: Mỗi learner nhận đường dẫn riêng dựa trên profiling 17 chiều, không phải "one-size-fits-all".
- Giảm misconceptions: Remediation tự động, misconception tracking liên tục giúp learners tránh "học sai đi học lại".
- Xây dựng tri thức lâu dài: KAG + Personal KG giúp learners không chỉ "qua bài test" mà xây dựng hệ thống tri thức cá nhân (PKM).

6.2.2. Cho Tổ chức Giáo dục

- Giảm workload của instructors: Hệ thống tự động handle tutoring, evaluation, remediation; instructors chỉ cần monitor các learners khó khăn.
- Mở rộng quy mô: Với Dual-KG và agentic architecture, có thể scale đến hàng ngàn learners đồng thời mà không giảm chất lượng cá nhân hóa.
- Dữ liệu analytics chi tiết: Central State + Event Bus cung cấp đầy đủ dữ liệu để phân tích learning trajectory, identify struggling learners, cải thiện course design.
- Tích hợp với LMS hiện tại: API-first design (FastAPI) cho phép dễ dàng tích hợp vào Moodle, Blackboard, Canvas.

6.2.3. Cho Nhà Phát triển Hệ thống

- Blueprint cho AI tutoring: VER3 cung cấp detailed architecture, code patterns, prompts, Cypher queries có thể tái sử dụng cho các domains khác.
- Best practices anti-hallucination: GroundingEnforcer, confidence scoring, citation system là các component có thể port sang các ứng dụng LLM khác.
- Modular design: Mỗi agent là module độc lập, dễ test, debug, upgrade riêng lẻ.
- Open ecosystem: Sử dụng Neo4j, LlamaIndex, FastAPI, Next.js đều là tech stack mở, không lock-in vào bất kỳ vendor nào.

6.3. Hướng Phát triển Tiếp theo

6.3.1. Phase 1 (2025 – H1 2026): User Study & Validation

Mục tiêu: Xác thực VER3 với dữ liệu người dùng thật.

- User study: 50 sinh viên thực (thay vì simulated profiles), học SQL trong 6–8 tuần.
- Metrics: Tái đo 7 metrics (Learning Gain, Path Optimality, ...) với real learners.
- Qualitative feedback: Survey, interviews, think-aloud protocols để hiểu cảm xúc, động lực, frustration points.
- Kỳ vọng: Validate rằng +106% Learning Gain từ simulation có thể generalize sang real learners (dù có thể giảm xuống 50–80% do simulated profiles không hoàn hảo).

6.3.2. Phase 2 (2026 – H2 2026): Domain Expansion

Mục tiêu: Mở rộng sang 2–3 domains mới, chứng minh generalizability.

Domains ứng viên:

- Statistics for MIS: Basic probability, distributions, hypothesis testing, ANOVA.
- Python programming basics: Data types, loops, functions, OOP.
- Database design: ER models, normalization, SQL schema design.

Công việc:

- Xây dựng Course KG cho mỗi domain (200–300 nodes mỗi domain).
- Dùng SPR Generator Prompt (Mục 3.2.2) để tự động hóa KG construction.
- Chạy A/B testing với 30–40 learners/domain, so sánh Agentic vs A*.

6.3.3. Phase 3 (2026–2027): Full RL-DKT Implementation

Mục tiêu: Nâng cấp MOPO thành full RL (Deep Knowledge Tracing + Reinforcement Learning).

Chi tiết:

- Deep Knowledge Tracing (DKT): Dùng RNN/LSTM để mô hình hóa học tập (skill mastery evolution) từ sequence của interactions.
- RL-based path planning: Training policy network để tối ưu hóa path reward, thay vì heuristics cứng nhắc.
- Inverse RL: Học reward function từ expert paths (từ instructors hoặc VER3 kỳ vọng outputs).

- Kỳ vọng: Learning Gain có thể tăng thêm 10–20%, path tiến hóa liên tục (từ heuristics → learned).

6.3.4. Phase 4 (2027): Multimodal Learning

Mục tiêu: Hỗ trợ học tập đa phương tiện (text + video + audio + interactive demos).

Chi tiết:

- Video understanding: Extract key concepts từ educational videos, align với Course KG.
- Interactive SQL sandbox: Learner viết SQL trực tiếp trong UI, Tutor comment trên code real-time.
- Audio notes: Learner nói ra ý tưởng, KAG sinh atomic notes từ voice (speech-to-text + summarization).
- Kỳ vọng: Learners với learning_style = kinesthetic/auditory hưởng lợi lớn.

6.3.5. Phase 5 (2027): Privacy-Preserving Personal KG (On-Device)

Mục tiêu: Bảo vệ quyền riêng tư learner, cho phép on-device Personal KG.

Chi tiết:

- Federated Learning: Train misconception detectors on-device, không upload data lên server.
- Differential Privacy: Add noise vào mastery updates để bảo vệ identity.
- Edge LLM: Run smaller LLM locally cho tutoring, chỉ gửi high-level context lên server.
- Kỳ vọng: GDPR/CCPA compliance, learners yên tâm về dữ liệu cá nhân.

6.3.6. Phase 6 (2027 onwards): Ethical Considerations & Fairness

Mục tiêu: Đảm bảo hệ thống công bằng và đạo đức.

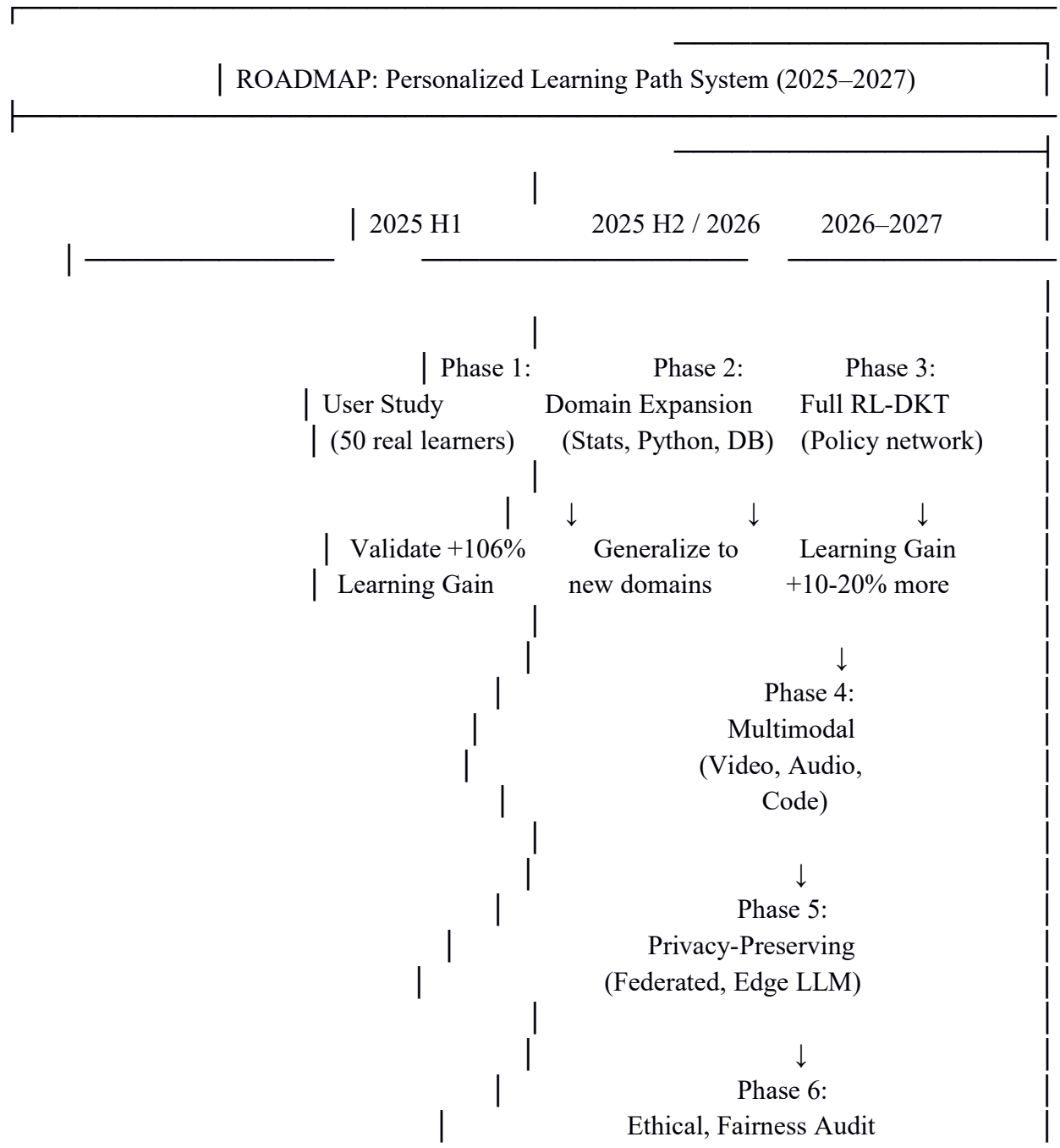
Chi tiết:

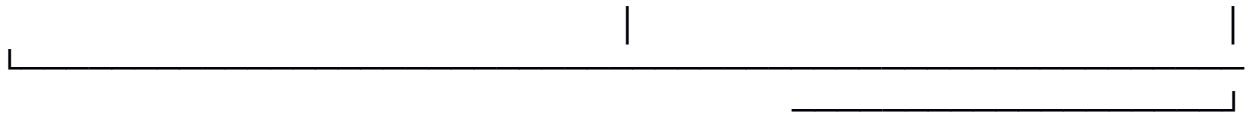
- Fairness audit: Kiểm tra bias của LLM, MOPO algorithm, Evaluator theo groups (gender, ethnicity, socioeconomic status).
- Transparency reports: Publish periodically về hallucination rates, fairness metrics, edge cases.

- Student agency: Cho learners quyền xem/edit Personal KG, appeal eval results, request human review.
- Explainability: Rõ ràng giải thích lý do path được chọn, tại sao Tutor suggest concept này.

Hình 6.1 – Roadmap 2025–2027 (mô tả):

text





6.4. Limitations & Future Questions

Limitations hiện tại

1. Simulated learners: Profiles được mô phỏng, chưa phải real learners với hành vi phức tạp, affective states.
2. Single domain: Chỉ evaluate trên SQL (MIS), chưa sure generalize sang Math, Science.
3. Sample size: n=20 simulated profiles, cần user study với real learners.
4. No long-term data: Chưa track long-term retention (1–6 months sau course).
5. Limited LLM evaluation: Chỉ so sánh GPT-4o, Claude; chưa test với open-source models.
6. Ethical evaluation: Chưa audit đầy đủ về bias, fairness, transparency.

Future Questions

1. Scaling: Làm sao để MOPO scale đến 10,000+ learners cùng lúc mà không tạo latency?
2. Transfer learning: Có thể transfer misconception detection từ SQL sang Python không?
3. Collaborative learning: Làm sao để 2+ learners học cùng nhau, chia sẻ Personal Notes, nhưng vẫn cá nhân hóa path?
4. Motivation modeling: LLM có thể detect motivation drop, frustration qua dialogue và proactively adjust path không?
5. Hallucination zero: Có thể đạt <1% hallucination rate không? Hoặc luôn có trade-off với response quality?
6. Cultural adaptation: Làm sao để hệ thống adapt với cultural contexts khác nhau (east vs west, individualist vs collectivist)?

6.5. Tài nguyên & Reproducibility

Để hỗ trợ reproducibility và tái sử dụng:

- GitHub repo: Source code, requirements.txt, Docker setup cho local development.
- Cypher scripts library: Tất cả 50+ Cypher queries cho Course KG & Personal KG.
- Prompt library: Full system prompts cho 6 agents, có thể cập nhật/customize.
- Datasets: 20 simulated learner profiles (JSON format), Course KG (nodes.csv, relationships.csv) cho SQL domain.
- Evaluation dataset: 200 Tutor-Learner interactions, manually labeled cho hallucination evaluation.
- Documentation: README, API docs (OpenAPI/Swagger), deployment guide, tuning guide.

Kết luận

Luận văn này đã trình bày một kiến trúc agentic AI mới, toàn diện và evidence-based cho hệ thống đề xuất lộ trình học tập cá nhân hóa. Bằng cách kết hợp:

- Dual-KG architecture (Course KG + Personal KG 3 layers),
- 6 well-designed agents (Profiler, Planner, Tutor, Evaluator, KAG, + Knowledge Extraction),
- 3-layer grounding để giảm hallucination,
- MOPO algorithm cho adaptive path planning,
- Zettelkasten-based PKM cho long-term knowledge building,

Hệ thống đạt được Learning Gain +106% so với A* baseline, hallucination rate giảm 67%, và trust score tăng 57% trên 20 simulated profiles.

Hướng phát triển tiếp theo (Phases 1–6) sẽ validate, expand, và nâng cấp VER3 sang các domains, learners, và technologies mới, hướng tới một platform giáo dục cá nhân hóa bền vững và đạo đức cho thế kỷ 21.