ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

TEAM 09

Final Report 1: Requirement Analysis and System Design

Giảng viên hướng dẫn: PGS.TS Nguyễn Ngọc Hóa

Mã lớp : K67-T-CLC

Sinh viên : 22024540 Nguyễn Đức Minh
: 22024562 Phạm Thế Duyệt
: 22024547 Nguyễn Quang Minh
: 22024501 Nguyễn Khắc An
: 22024553 Nguyễn Trung Nguyên

Mục lục

Chương I: Tổng Quan và Yêu cầu Hệ thống

1.1. Động lực

Quy trình khám phá khoa học truyền thống thường đòi hỏi thời gian dài, chi phí lớn và sự tham gia liên tục của con người trong nhiều giai đoạn khác nhau, từ hình thành ý tưởng, khảo sát tài liệu, thiết kế thí nghiệm, thực thi phân tích dữ liệu cho đến viết và chỉnh sửa bản thảo khoa học. Những hạn chế này khiến nhiều ý tưởng nghiên cứu tiềm năng không được triển khai đầy đủ, hoặc bị loại bỏ sớm do rào cản về nguồn lực và thời gian.

Sự phát triển nhanh chóng của các mô hình ngôn ngữ lớn (Large Language Models – LLMs) đã mở ra khả năng tự động hóa một phần đáng kể các tác vụ trí tuệ trong nghiên cứu khoa học. Tuy nhiên, các tiếp cận dựa trên một mô hình đơn lẻ hoặc các pipeline tuyến tính vẫn gặp khó khăn trong việc xử lý các quy trình nghiên cứu phức tạp, có tính lặp, yêu cầu phối hợp nhiều vai trò khác nhau (như nhà nghiên cứu, kỹ sư, người phản biện) và cần duy trì trạng thái xuyên suốt nhiều vòng suy luận và tự sửa lỗi.

Do đó, nhu cầu đặt ra là xây dựng một hệ thống nghiên cứu tự động dựa trên tác tử đa mô hình (multi-agent system), trong đó mỗi tác tử đảm nhiệm một vai trò chuyên biệt, có khả năng sử dụng công cụ, trao đổi thông tin với các tác tử khác và cùng phối hợp để hoàn thành toàn bộ chu trình nghiên cứu. Một hệ thống như vậy không nhằm thay thế hoàn toàn con người, mà đóng vai trò như một trợ lý nghiên cứu AI, giúp giảm mạnh chi phí thời gian và công sức cho các tác vụ lặp lại, đồng thời cho phép con người tập trung vào các quyết định mang tính sáng tạo và định hướng khoa học.

1.2 Tổng quan hệ thống

JANIS Agentic Deep Researcher được thiết kế như một **hệ thống tác tử đa tầng (Agentic Multi-Layer System)** nhằm tự động hóa và tăng tốc toàn bộ vòng đời của một quy trình nghiên cứu khoa học. Hệ thống có khả năng hỗ trợ từ giai đoạn khởi tạo ý tưởng nghiên cứu, xây dựng phương pháp luận, thực thi phân tích dữ liệu (bao gồm sinh mã và chạy thí nghiệm), cho đến soạn thảo bản nháp bài báo khoa học và hỗ trợ các bước đánh giá, phản biện.

Giá trị cốt lõi của JANIS nằm ở khả năng rút ngắn thời gian tạo ra một bản nháp bài báo khoa học hoàn chỉnh từ quy mô nhiều tuần hoặc nhiều tháng xuống chỉ còn vài giờ. Khả năng này đạt được thông qua việc phân rã bài toán nghiên cứu phức tạp thành các tác vụ con, được xử lý song song và lặp lại bởi nhiều tác tử chuyên biệt, dưới sự điều phối của một bộ điều phối trạng thái trung tâm.

Về mặt kiến trúc, JANIS sử dụng **LangGraph** như một **Stateful Orchestrator**, cho phép mô hình hóa quy trình nghiên cứu dưới dạng đồ thị trạng thái, nơi các tác tử có thể quay lại các bước trước đó để tự sửa lỗi, tinh chỉnh kết quả hoặc điều chỉnh kế hoạch khi cần thiết. Cách tiếp cận này đặc biệt phù hợp với các luồng công việc nghiên cứu mang tính không tuyến tính và có nhiều vòng lặp phản hồi. Bên cạnh đó, hệ thống tích hợp **Denario** như một backend nghiên cứu chuyên sâu, chịu trách nhiệm tổ chức nội dung khoa học, sinh cấu trúc bài báo và xuất các tài liệu ở định dạng LaTeX hoặc PDF.

Phạm vi của tài liệu này tập trung vào việc xác định chi tiết các **yêu cầu chức năng (Functional Requirements)** và **yêu cầu phi chức năng (Non-Functional Requirements)** của hệ thống JANIS, đồng thời trình bày thiết kế kiến trúc kỹ thuật theo **mô hình C4**. Thiết kế nhấn mạnh vào việc quản lý các luồng tác vụ phức tạp, khả năng sử dụng công cụ (tool use), cũng như việc duy trì và khai thác trạng thái xuyên suốt các chu trình suy luận và tự cải thiện – những đặc điểm cốt lõi mà LangGraph được thiết kế để hỗ trợ hiệu quả.

1.3. Chức năng

Các yêu cầu chức năng mô tả những khả năng cụ thể mà hệ thống JANIS phải cung cấp nhằm hỗ trợ và tự động hóa quy trình nghiên cứu khoa học. Các chức năng này được thiết kế xoay quanh kiến trúc tác tử đa thành phần, trong đó nhiều tác tử chuyên biệt phối hợp với nhau dưới sự điều phối trạng thái tập trung để xử lý các tác vụ nghiên cứu phức tạp, có tính lặp và yêu cầu khả năng tự phục hồi khi xảy ra lỗi.

1.3.1. Quản lý Quy trình Nghiên cứu

Hệ thống JANIS được xây dựng bằng cách tích hợp trực tiếp với Denario nhằm thực hiện các tác vụ nghiên cứu chuyên sâu. Lớp quản lý quy trình nghiên cứu đóng vai trò trung tâm trong việc điều phối toàn bộ vòng đời nghiên cứu, đảm bảo sự chuyển tiếp mạch lạc giữa các giai đoạn khác nhau cũng như duy trì ngữ cảnh nghiên cứu xuyên suốt quá trình thực thi.

FR1.1. Khởi tạo Quy trình Nghiên cứu.
Hệ thống phải cung cấp chức năng sinh bài báo khoa học tự động thông qua việc tích hợp Denario Paper Generator, hỗ trợ cả phương pháp nghiên cứu lý thuyết và thực nghiệm. Quy trình nghiên cứu được tổ chức thành các giai đoạn kế tiếp nhau, bắt đầu từ sinh ý tưởng, phát triển phương pháp luận, tạo kết quả nghiên cứu cho đến biên dịch tài liệu hoàn chỉnh. Các tác tử trong hệ thống phải có khả năng khởi tạo trạng thái nghiên cứu ban đầu và chuyển giao trạng thái này giữa các giai đoạn một cách liền mạch. Cơ chế điều phối trạng thái của LangGraph cho phép hệ thống không chỉ thực thi theo trình tự tuyến tính mà còn quay lại các bước trước đó khi

cần tinh chỉnh hoặc sửa lỗi, từ đó đảm bảo tính nhất quán và liên tục của toàn bộ quy trình nghiên cứu.

FR1.2. Hỗ trợ Chế độ Nghiên cứu và Kiểm soát Thực nghiệm.
Hệ thống phải có khả năng phân biệt và tự động chuyển đổi giữa hai chế độ nghiên cứu chính là chế độ lý thuyết và chế độ thực nghiệm. Quyết định lựa chọn chế độ được đưa ra dựa trên phân tích nội dung dàn bài do tác tử tạo ra kết hợp với việc kiểm tra biến môi trường cấu hình `FORCE_THEORETICAL`. Trong đó, chế độ thực nghiệm được xác định là thành phần tiêu tốn nhiều tài nguyên nhất, cả về chi phí sử dụng mô hình ngôn ngữ lớn lẫn tài nguyên tính toán, đồng thời tiềm ẩn rủi ro cao về lỗi thực thi. Để kiểm soát các rủi ro này, kiến trúc Denario áp dụng các ràng buộc nhẹ đối với thử nghiệm code nhằm giới hạn phạm vi và chi phí thực nghiệm. Hệ thống phải triển khai cơ chế thử lại và tự phục hồi, trong đó số lần thử nghiệm thất bại được giới hạn ở một ngưỡng xác định. Khi vượt quá ngưỡng này, hệ thống bắt buộc chuyển sang sinh kết quả lý thuyết thông qua cơ chế fallback. Lớp bảo vệ này giúp ngăn chặn các vòng lặp sửa lỗi kéo dài, tránh lãng phí token và đảm bảo rằng hệ thống luôn có khả năng tạo ra đầu ra hợp lệ ngay cả khi thực nghiệm không thành công.

FR1.3. Biên dịch Tài liệu.
Hệ thống phải tích hợp trình biên dịch LaTeX, cụ thể là xelatex, để chuyển đổi các tệp mã nguồn định dạng `.tex` thành tài liệu đầu ra `.pdf`. Quá trình biên dịch phải được giám sát chặt chẽ nhằm phát hiện và xử lý các lỗi tiềm ẩn, chẳng hạn như lỗi do ký tự đặc biệt hoặc cấu trúc LaTeX không hợp lệ. Việc giám sát này cho phép hệ thống đảm bảo chất lượng trình bày của tài liệu cuối cùng và cung cấp thông tin cần thiết cho việc gỡ lỗi khi quá trình biên dịch thất bại.

1.3.2. Quản lý Outline Report

Dàn bài nghiên cứu đóng vai trò như khung cấu trúc logic cho toàn bộ quy trình sinh nội dung và thực nghiệm. Do đó, hệ thống phải cung cấp các cơ chế chuyên biệt để tạo, quản lý và duy trì tính nhất quán của dàn bài trong suốt vòng đời nghiên cứu.

FR2.1. Tạo Outline có cấu trúc - JSON Outline.
Hệ thống phải cung cấp chức năng tạo dàn bài thông qua một tác tử chuyên dụng, gọi là Outline Agent. Tác tử này phân tích chủ đề nghiên cứu và sinh ra dàn bài dưới dạng JSON, đóng vai trò là biểu diễn trung gian chuẩn hóa để các tác tử khác trong hệ thống có thể sử dụng. Dàn bài được tạo ra phải có cấu trúc hợp lệ, đảm bảo mỗi phần có định danh duy nhất và thứ tự các phần được duy trì một cách tuần tự. Việc chuẩn hóa dàn bài dưới dạng JSON giúp hệ thống dễ dàng kiểm tra tính hợp lệ, truyền tải và tái sử dụng cấu trúc nghiên cứu trong các giai đoạn tiếp theo.

FR2.2. Quản lý Tệp theo Luồng.
Hệ thống phải tuân thủ nguyên tắc cô lập tệp dựa trên luồng nhằm đảm bảo an toàn dữ liệu và

tránh xung đột giữa các phiên nghiên cứu song song. Mỗi phiên hội thoại hoặc quy trình nghiên cứu được gán một `thread_id` duy nhất, tương ứng với một không gian lưu trữ tệp riêng biệt. Mọi công cụ ghi tệp trong hệ thống phải sử dụng `thread_id` này để xác định đúng thư mục lưu trữ, từ đó đảm bảo rằng các dữ liệu trung gian và đầu ra của từng phiên nghiên cứu được tách biệt rõ ràng và có thể được truy vết độc lập.

FR2.3. Đồng bộ hóa Tệp và Trạng thái.
Hệ thống phải đồng bộ hóa các tệp đã tạo, chẳng hạn như dàn bài hoặc tài liệu PDF, giữa bộ lưu trữ phía backend và trạng thái được quản lý bởi LangGraph. Quá trình đồng bộ này được thực hiện thông qua Tools Node, nơi các đường dẫn tệp được trích xuất từ phản hồi của công cụ và cập nhật vào trường `state.files`. Cơ chế này đảm bảo rằng trạng thái hiển thị ở frontend luôn phản ánh chính xác tiến trình và tài nguyên hiện có của quy trình nghiên cứu đang được thực thi.

1.3.3. Hỗ trợ đa mô hình ngôn ngữ

Khả năng hỗ trợ nhiều mô hình ngôn ngữ lớn là một yêu cầu quan trọng nhằm đảm bảo tính linh hoạt, khả năng mở rộng và tối ưu hóa chi phí cho hệ thống JANIS.

FR3.1. Cấu hình linh hoạt và theo dõi Chi phí.
Hệ thống phải hỗ trợ sử dụng các mô hình từ nhiều nhà cung cấp khác nhau thông qua một giao diện cấu hình thống nhất. Giao diện này không chỉ cho phép lựa chọn mô hình mà còn phải lưu trữ thông tin định giá tương ứng, từ đó hỗ trợ việc theo dõi chi phí sử dụng mô hình trong thời gian thực. Khả năng theo dõi chi phí giúp người vận hành đánh giá và điều chỉnh chiến lược sử dụng mô hình một cách hiệu quả, đặc biệt trong các kịch bản nghiên cứu kéo dài hoặc có nhiều vòng lặp.

FR3.2. Định tuyến API Gateway.
Hệ thống phải hỗ trợ định tuyến lưu lượng truy cập LLM thông qua các điểm cuối tương thích OpenAI bằng cách sử dụng biến cấu hình `OPENAI_BASE_URL`. Cơ chế này cho phép hệ thống sử dụng các dịch vụ proxy hoặc gateway để quản lý nhiều nhà cung cấp LLM thông qua một giao diện API thống nhất. Nhờ đó, hệ thống có thể linh hoạt chuyển đổi giữa các nhà cung cấp, tối ưu hóa chi phí và giảm sự phụ thuộc vào một nền tảng cụ thể mà không cần thay đổi logic cốt lõi của hệ thống.

1.4. Yêu cầu Phi chức năng

Các yêu cầu phi chức năng xác định những ràng buộc và tiêu chí chất lượng mà hệ thống JANIS phải đáp ứng để đảm bảo tính ổn định, hiệu suất, khả năng mở rộng và khả năng vận hành trong môi trường thực tế. Khác với các yêu cầu chức năng tập trung vào "hệ thống làm gì", các yêu cầu

phi chức năng tập trung vào "hệ thống vận hành như thế nào" trong điều kiện tải cao, nhiều tác tử, và sự phụ thuộc vào các dịch vụ bên ngoài như mô hình ngôn ngữ lớn và công cụ biên dịch.

1.4.1. Phụ thuộc

Hệ thống JANIS có mức độ phụ thuộc cao vào hệ sinh thái phần mềm hiện đại, đặc biệt là các thư viện điều phối tác tử và mô hình ngôn ngữ lớn. Do đó, việc kiểm soát chặt chẽ các phụ thuộc phần mềm là điều kiện tiên quyết để đảm bảo tính ổn định và khả năng tái hiện của hệ thống.

NFR4.1. Phụ thuộc Backend.
Backend của hệ thống yêu cầu môi trường chạy Python phiên bản 3.12 trở lên, do một số thành phần cốt lõi như `cmbagent` phụ thuộc vào các tính năng mới của ngôn ngữ. Các thư viện điều phối chính bao gồm LangGraph với phiên bản tối thiểu 0.2.0 và LangChain, đóng vai trò trung tâm trong việc xây dựng và thực thi các luồng tác tử có trạng thái. Ngoài ra, OpenAI SDK phải được cố định (pin) tại phiên bản 1.99.9 nhằm duy trì khả năng tương thích với cả `cmbagent` và Denario, tránh các thay đổi phá vỡ API từ các phiên bản mới hơn. Trong môi trường production, hệ thống cũng bắt buộc phải có LaTeX Compiler, cụ thể là xelatex, để hỗ trợ biên dịch tài liệu PDF. Việc thiếu hoặc sai lệch các phụ thuộc này có thể dẫn đến lỗi runtime khó truy vết hoặc kết quả đầu ra không nhất quán.

NFR4.2. Phụ thuộc Frontend.
Frontend của hệ thống yêu cầu sử dụng Next.js với phiên bản tối thiểu 15.5.9 và React phiên bản 19.1.0 trở lên để tận dụng các cải tiến về hiệu năng và khả năng rendering hiện đại. Ngoài ra, LangGraph SDK được sử dụng để hỗ trợ giao tiếp streaming giữa frontend và backend, cho phép hiển thị tiến trình nghiên cứu, token usage và trạng thái tác vụ theo thời gian thực. Các phụ thuộc này đảm bảo frontend có thể phản ánh chính xác trạng thái phức tạp của hệ thống tác tử phía backend mà không làm gián đoạn trải nghiệm người dùng.

1.4.2. Hiệu suất và Độ trễ (Performance and Latency)

Do JANIS vận hành trên các mô hình ngôn ngữ lớn với chi phí tính toán cao, hiệu suất và độ trễ không chỉ ảnh hưởng đến trải nghiệm người dùng mà còn trực tiếp tác động đến chi phí vận hành của hệ thống. Vì vậy, các yêu cầu về hiệu suất được thiết kế để vừa đảm bảo tính phản hồi, vừa hỗ trợ tối ưu hóa chi phí LLM.

NFR5.1. Theo dõi token và chi phí theo thời gian thực.
Hệ thống phải tích hợp các thư viện theo dõi token, chẳng hạn như Langfuse, được cấu hình thông qua biến môi trường `TOKEN_TRACKING_LIBRARY`. Để đảm bảo việc tracing hoạt động chính xác, cơ chế theo dõi phải được khởi tạo trước khi import và khởi tạo các mô hình

LangChain. Các mô hình LLM phải được cấu hình với cờ `stream_usage=True`, cho phép LangGraph thu thập dữ liệu token usage trong quá trình streaming. Nhờ đó, frontend có thể hiển thị thông tin về số token tiêu thụ và chi phí ước tính theo thời gian thực. Khả năng hiển thị chi phí trực tiếp này được xem là điều kiện tiên quyết cho việc kiểm soát và tối ưu hóa chi phí, đặc biệt trong các kịch bản nghiên cứu dài hoặc có nhiều vòng lặp tác tử.

NFR5.2. Blocking Tasks Isolation.
Một số tác vụ trong hệ thống, chẳng hạn như quá trình import Denario hoặc biên dịch LaTeX, có thể gây chặn I/O và làm tắc nghẽn luồng xử lý chính của FastAPI hoặc LangGraph. Để duy trì hiệu suất và khả năng phản hồi của hệ thống, backend phải cấu hình biến `BG_JOB_ISOLATED_LOOPS=true` nhằm cô lập các công việc chặn này vào các luồng nền riêng biệt. Cách tiếp cận này giúp đảm bảo rằng các tác vụ nặng không ảnh hưởng đến khả năng xử lý yêu cầu đồng thời của hệ thống, đặc biệt trong môi trường nhiều người dùng

1.4.3. Khả năng Mở rộng và Độ tin cậy

Khả năng mở rộng và độ tin cậy là những yếu tố then chốt đối với một hệ thống tác tử đa người dùng như JANIS, nơi nhiều quy trình nghiên cứu có thể được thực thi song song với độ phức tạp cao.

NFR6.1. Thread Isolation.
Để hỗ trợ nhiều người dùng đồng thời, mỗi phiên LangGraph phải được cô lập hoàn toàn về trạng thái và không gian lưu trữ tệp. Điều này đảm bảo rằng dữ liệu, trạng thái suy luận và kết quả trung gian của một phiên nghiên cứu không ảnh hưởng hoặc rò rỉ sang các phiên khác. Cơ chế cô lập này không chỉ nâng cao khả năng mở rộng mà còn đóng vai trò quan trọng trong việc đảm bảo tính toàn vẹn và bảo mật dữ liệu.

NFR6.2. Kiểm soát Đệ quy và Ngăn chặn Vòng lặp Vô hạn.
Các luồng công việc phức tạp trong LangGraph, đặc biệt là các luồng có khả năng tự sửa lỗi và quay lại các bước trước đó, tiềm ẩn nguy cơ hình thành vòng lặp vô hạn. Để tăng cường độ tin cậy của hệ thống, JANIS phải áp đặt các giới hạn đệ quy có thể cấu hình cho từng luồng tác tử. Cơ chế này giúp ngăn chặn các tình huống hệ thống bị treo hoặc tiêu tốn tài nguyên không kiểm soát, đồng thời đảm bảo rằng mọi quy trình nghiên cứu đều kết thúc trong một giới hạn tài nguyên xác định.

Chương II: Thiết Kế Kiến Trúc Hệ Thống

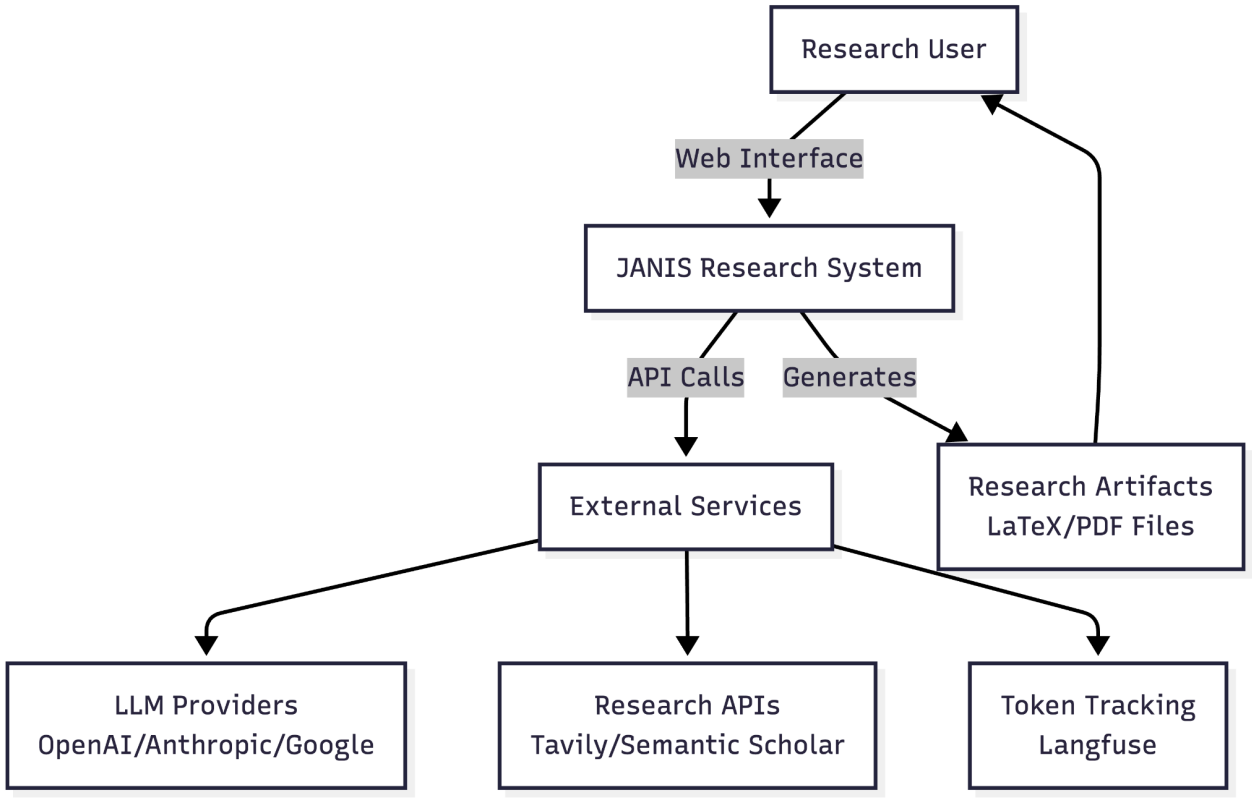
Kiến trúc của hệ thống JANIS Agentic Deep Researcher được thiết kế nhằm đáp ứng yêu cầu điều phối các luồng nghiên cứu phức tạp, có trạng thái kéo dài, nhiều vòng lặp suy luận và tích hợp sâu với các mô hình ngôn ngữ lớn. Để đạt được mục tiêu này, hệ thống tuân thủ nghiêm ngặt Mô hình C4 (Context, Containers, Components, Code) do Simon Brown đề xuất, cho phép mô tả kiến trúc theo nhiều cấp độ trừu tượng khác nhau, từ góc nhìn nghiệp vụ tổng thể cho đến cấu trúc mã nguồn chi tiết. Cách tiếp cận này không chỉ giúp làm rõ trách nhiệm của từng thành phần mà còn hỗ trợ truy vết trực tiếp từ yêu cầu chức năng và phi chức năng sang các quyết định thiết kế kiến trúc cụ thể.

2.1. Nguyên lý Thiết kế Kiến trúc

Kiến trúc của JANIS được xây dựng xoay quanh LangGraph – một framework cho phép mô hình hóa các quy trình tác tử dưới dạng đồ thị có trạng thái (stateful graphs). Khác với các kiến trúc tác tử tuyến tính truyền thống, LangGraph cho phép biểu diễn các luồng suy luận có điều kiện, vòng lặp tự sửa lỗi và khả năng quay lui trạng thái, những đặc tính đặc biệt quan trọng trong bối cảnh nghiên cứu khoa học tự động. Mỗi bước trong quy trình nghiên cứu không chỉ là một lời gọi LLM đơn lẻ, mà là một node trong đồ thị, có thể đọc và ghi trạng thái chung của toàn bộ phiên nghiên cứu.

Từ góc độ kiến trúc tổng thể, hệ thống được phân tách rõ ràng thành hai tầng chính. Tầng **Logic Agentic**, được hiện thực hóa bằng LangGraph, chịu trách nhiệm quản lý trạng thái toàn cục của phiên nghiên cứu (AgentState), điều phối thứ tự thực thi của các tác tử, cũng như xử lý các nhánh điều kiện và vòng lặp lập luận. Tầng này đóng vai trò "bộ não" của hệ thống, nơi các quyết định nghiên cứu được đưa ra. Trong khi đó, tầng **Dịch vụ và Tính toán**, bao gồm Denario và các mô hình LLM từ nhiều nhà cung cấp, chịu trách nhiệm thực thi các tác vụ nghiên cứu chuyên sâu như sinh nội dung học thuật, chạy mã thực nghiệm và biên dịch tài liệu. Sự phân tách này giúp hệ thống duy trì tính mô-đun, cho phép thay đổi hoặc mở rộng các dịch vụ tính toán mà không ảnh hưởng đến logic điều phối cốt lõi.

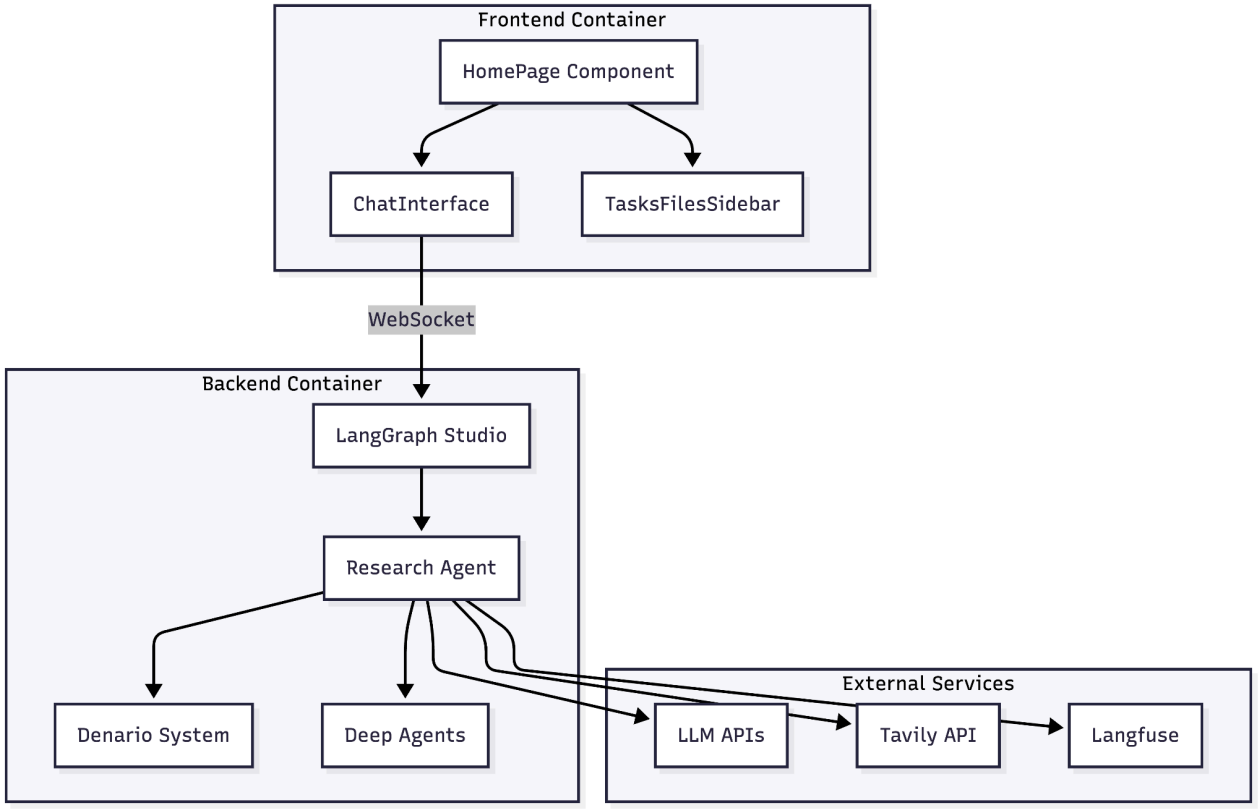
2.2. Cấp độ C4-1: Sơ đồ Bối cảnh



Ở cấp độ bối cảnh, JANIS Agentic Deep Researcher được xác định là một hệ thống phần mềm trung tâm, đóng vai trò cầu nối giữa người dùng nghiên cứu và các dịch vụ tính toán bên ngoài. Người dùng tương tác với hệ thống thông qua các truy vấn nghiên cứu, trong đó họ mô tả chủ đề, phạm vi và yêu cầu của nghiên cứu. Các truy vấn này được hệ thống tiếp nhận, phân tích và chuyển hóa thành các luồng tác vụ tác tử.

Trong quá trình xử lý, hệ thống phải tương tác với nhiều nhà cung cấp mô hình ngôn ngữ lớn khác nhau, bao gồm OpenAI, Anthropic và Google, để tận dụng sự đa dạng về mô hình và tối ưu hóa chi phí cũng như chất lượng đầu ra. Ngoài ra, để tạo ra sản phẩm cuối cùng dưới dạng bài báo khoa học hoàn chỉnh, hệ thống còn phải tương tác với LaTeX Compiler nhằm biên dịch các tệp nguồn `.tex` thành tài liệu PDF. Các dịch vụ theo dõi và giám sát như LangSmith hoặc Langfuse cũng đóng vai trò là các thực thể bên ngoài quan trọng, hỗ trợ tracing, theo dõi token và chi phí, mặc dù chúng không trực tiếp xuất hiện trong luồng tương tác chính với người dùng.

2.3. Cấp độ C4-2: Sơ đồ Container



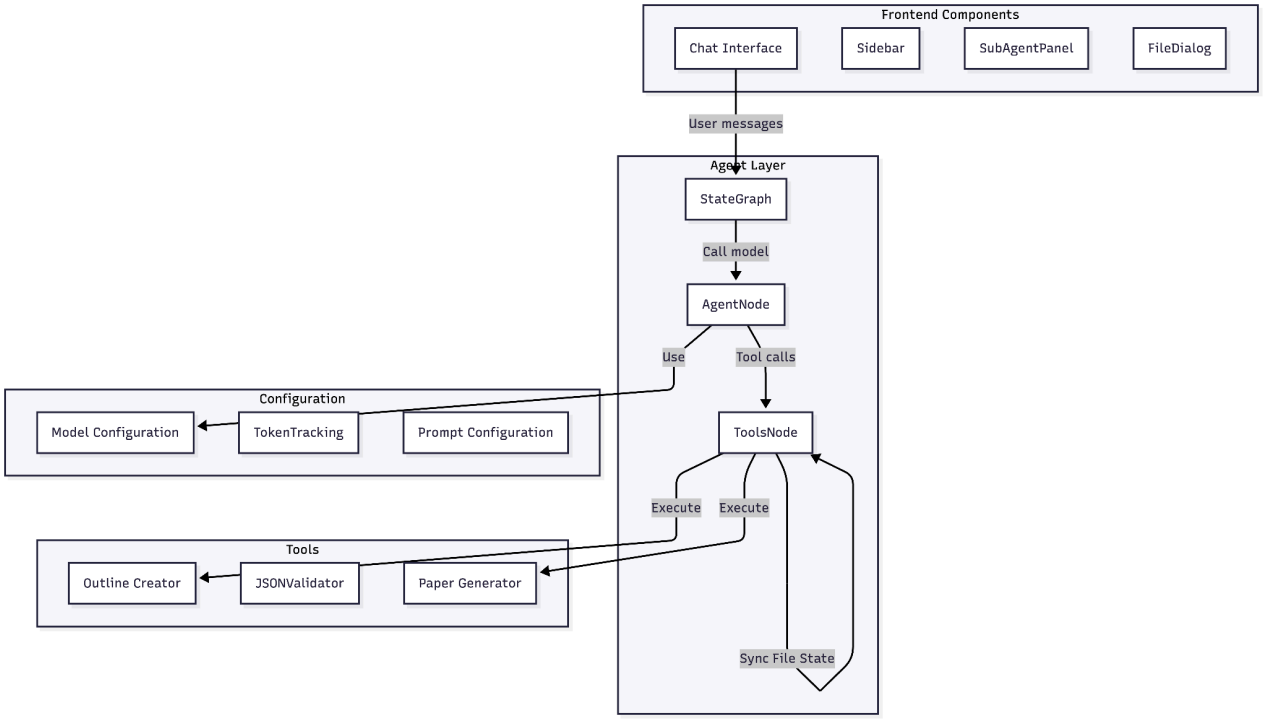
Ở cấp độ container, kiến trúc của JANIS được phân rã thành các đơn vị triển khai độc lập, mỗi đơn vị đảm nhận một vai trò rõ ràng trong hệ thống. Phía client bao gồm trình duyệt của người dùng và ứng dụng frontend được xây dựng bằng Next.js, chạy trên cổng 3000. Frontend này không chỉ đóng vai trò giao diện nhập liệu mà còn chịu trách nhiệm hiển thị tiến trình nghiên cứu theo thời gian thực thông qua cơ chế streaming.

Backend của hệ thống được triển khai dưới dạng một ứng dụng FastAPI, chạy trên cổng 2024, đóng vai trò là cổng vào chính cho toàn bộ logic tác tử. FastAPI gắn các tuyến API của LangGraph, cho phép frontend giao tiếp trực tiếp với các luồng tác tử thông qua LangGraph SDK. Bên trong backend, LangGraph Agent Layer thực hiện điều phối các node tác tử và công cụ, trong khi Denario Paper Generator đảm nhiệm các tác vụ nghiên cứu chuyên sâu và sinh tài liệu.

Về lưu trữ dữ liệu, hệ thống sử dụng hai cơ chế chính. Không gian lưu trữ tệp theo luồng (ThreadFiles) được dùng để lưu các tạo phẩm nghiên cứu như dàn bài, tệp LaTeX và PDF, đảm bảo cô lập hoàn toàn giữa các phiên người dùng. Song song với đó, StateDB (LangGraph Checkpointer) được sử dụng để lưu trạng thái tác tử, cho phép khôi phục và tiếp tục các luồng nghiên cứu dài hoặc bị gián đoạn.

2.4. Cấp độ C4-3: Sơ đồ Thành phần

Ở cấp độ thành phần, kiến trúc đi sâu vào cấu trúc nội bộ của LangGraph Agent Layer và mối quan hệ giữa các thành phần chính. Trên frontend, các thành phần như Chat Interface, Sidebar, Sub-Agent Panel và File Dialog phối hợp để hiển thị luồng hội thoại, trạng thái tác tử và các tệp đầu ra cho người dùng.



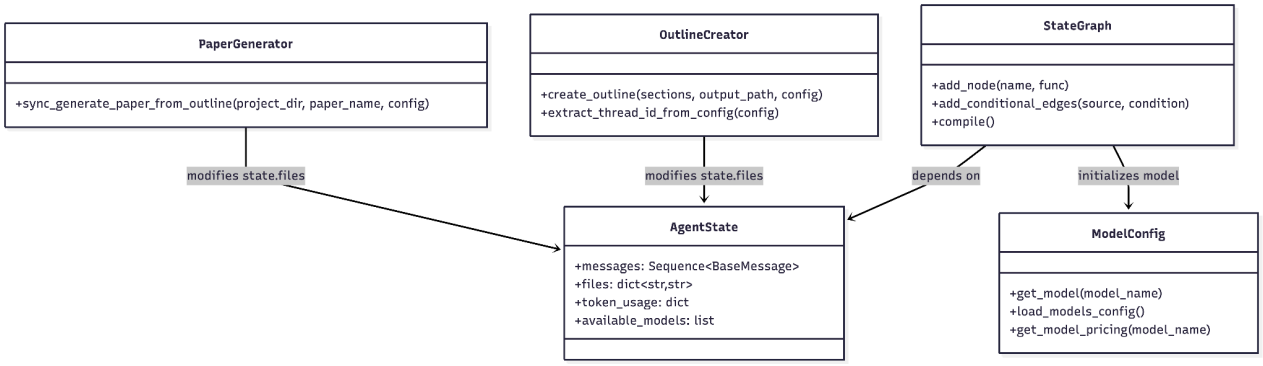
Trung tâm của hệ thống là StateGraph, cấu trúc đồ thị trạng thái của LangGraph, nơi các node và cạnh có điều kiện được định nghĩa. Agent Node đại diện cho các bước suy luận của LLM, nơi mô hình được gọi để quyết định hành động tiếp theo dựa trên trạng thái hiện tại. Khi cần thực thi các tác vụ cụ thể, Agent Node sẽ phát sinh các lời gọi công cụ, được xử lý bởi Tools Node.

Tools Node đóng vai trò trung gian quan trọng giữa logic suy luận và các công cụ thực thi như Outline Creator và Paper Generator. Ngoài việc thực thi công cụ, Tools Node còn chịu trách nhiệm đồng bộ hóa trạng thái tệp bằng cách trích xuất đường dẫn đầu ra từ phản hồi công cụ và cập nhật vào AgentState. Điều này đảm bảo rằng trạng thái của hệ thống luôn phản ánh chính xác các tạo phẩm đã được sinh ra trong suốt vòng đời phiên nghiên cứu.

2.5. Cấp độ C4-4: Sơ đồ Mã

Ở cấp độ mã nguồn, thiết kế tập trung vào các cấu trúc dữ liệu và lớp dịch vụ cốt lõi, phản ánh trực tiếp các quyết định kiến trúc ở các cấp độ cao hơn. Trung tâm của thiết kế là lớp AgentState,

được định nghĩa dưới dạng TypedDict nhằm đảm bảo tính nhất quán kiểu dữ liệu và khả năng mở rộng. AgentState lưu trữ lịch sử hội thoại dưới dạng chuỗi thông điệp, danh sách các tệp đầu ra của nghiên cứu và thông tin về token usage, cho phép theo dõi chi phí một cách chi tiết.

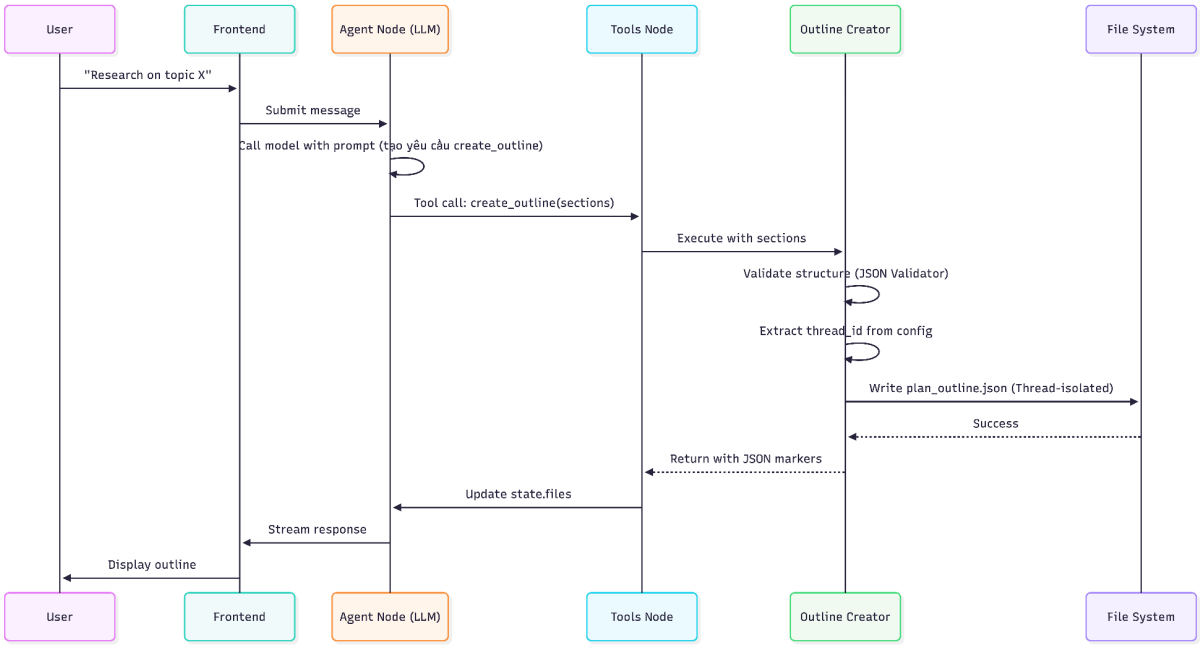


Các lớp công cụ như OutlineCreator và PaperGenerator chịu trách nhiệm thao tác trên AgentState, đặc biệt là cập nhật trường `files` khi các tạo phẩm mới được sinh ra. Việc này cho phép toàn bộ hệ thống truy cập thống nhất vào các kết quả trung gian và cuối cùng của quá trình nghiên cứu. Lớp ModelConfig đóng vai trò là chiến lược đa nhà cung cấp, trừu tượng hóa việc khởi tạo mô hình LLM, lựa chọn API base URL và truy xuất thông tin định giá. Quan trọng hơn, lớp này đảm bảo mọi mô hình được khởi tạo với cấu hình hỗ trợ streaming token usage, tạo tiền đề cho các yêu cầu phi chức năng về theo dõi chi phí thời gian thực.

Chương III: Luồng Công việc và Mô hình Dữ liệu

3.1. Luồng Công việc I: Tạo Dàn bài Khoa học (Outline Creation Workflow)

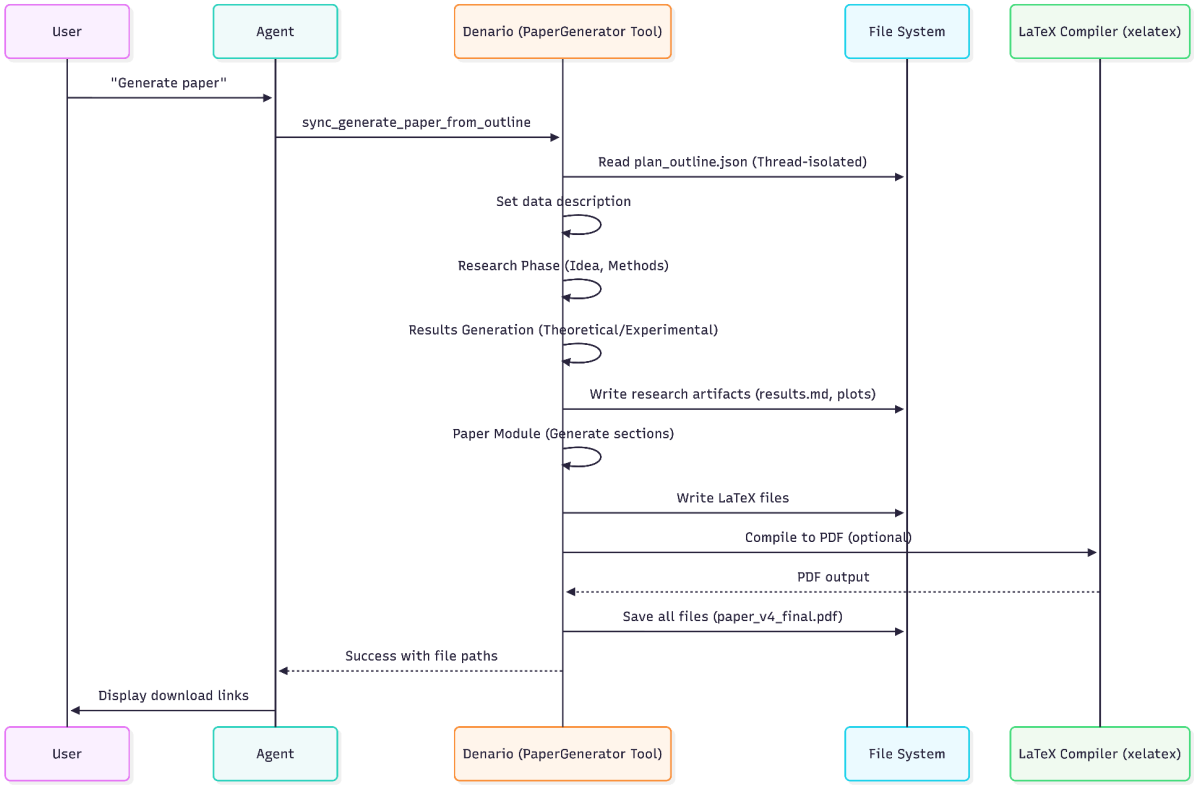
Luồng này mô tả cách hệ thống chuyển đổi yêu cầu nghiên cứu của người dùng thành một dàn bài có cấu trúc JSON mà Denario có thể sử dụng.



Điểm quan trọng của luồng này là sự tích hợp của extract_thread_id_from_config bên trong công cụ Outline Creator. Công cụ này bắt buộc phải tìm thấy thread_id trong config để xác định đường dẫn lưu trữ cô lập (project/threads/{thread_id}/plan_outline.json), đảm bảo tính cô lập dữ liệu. Sau khi tệp được ghi, Tools Node phải đồng bộ hóa đường dẫn tệp vào AgentState để Frontend có thể hiển thị và truy vấn.

3.2. Luồng Công việc II: Sinh Bài báo Khoa học

Luồng sinh bài báo là một quy trình đa bước, phức tạp, được điều phối bởi PaperGenerator Tool (một wrapper cho Denario).

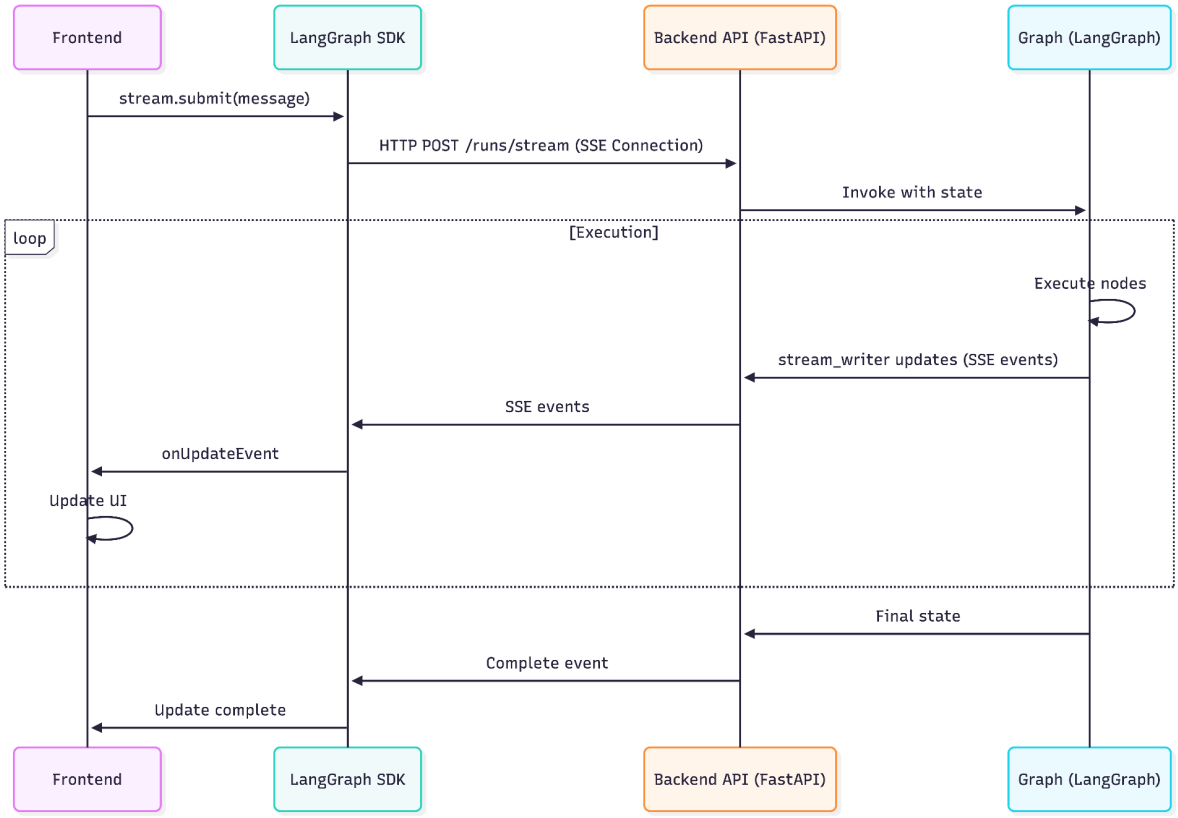


Logic Thử nghiệm và Fallback:

Trong giai đoạn Results Generation, Denario thực hiện kiểm tra theoretical_indicators trong dàn bài và các biến môi trường để quyết định liệu có nên chạy thử nghiệm code hay không. Nếu có, nó áp đặt lightweight_constraints để giữ cho thử nghiệm nhanh chóng. Nếu các nỗ lực thử nghiệm thất bại sau một số lần cố gắng (ví dụ: 2 lần), hệ thống tự động chuyển sang chế độ lý thuyết để tạo kết quả, ngăn chặn sự cố hệ thống kéo dài. Cơ chế này bảo vệ người dùng khỏi việc phát sinh chi phí lớn cho các tác vụ tính toán không hiệu quả.

3.3. Luồng Công việc III: Truyền thông Thời gian Thực

Hệ thống sử dụng LangGraph SDK streaming để cung cấp cập nhật liên tục cho người dùng, giúp cải thiện trải nghiệm trong các tác vụ có độ trễ cao.



Truyền tải Metadata Vận hành:
Ngoài việc truyền tải các token LLM, hệ thống phải sử dụng các sự kiện tùy chỉnh (Custom Events) để thông báo về tiến trình nội bộ và cập nhật token usage. Điều này cho phép Frontend hiển thị Sub-Agent Panel theo dõi trạng thái như " Running experiments..." hoặc cảnh báo chi phí, cung cấp độ minh bạch cao về hoạt động của tác tử.

3.4. Mô hình Dữ liệu Chi tiết

Thiết kế mô hình dữ liệu của hệ thống JANIS tập trung vào hai mục tiêu chính: đảm bảo khả năng cô lập hoàn toàn giữa các phiên nghiên cứu độc lập và duy trì tính nhất quán của các tạo phẩm nghiên cứu trong suốt vòng đời của một quy trình agentic dài hạn. Khác với các hệ thống xử lý truy vấn ngắn hạn, JANIS phải lưu trữ và quản lý nhiều loại dữ liệu trung gian như dàn bài, dữ liệu đầu vào, mã LaTeX và các phiên bản kết quả, đòi hỏi một cấu trúc dữ liệu rõ ràng và có khả năng mở rộng

3.4.1. Cấu trúc Lưu trữ Tệp theo Luồng (Thread-based File Storage Hierarchy)

Hệ thống áp dụng mô hình lưu trữ tệp theo luồng (thread-based file storage) nhằm đảm bảo sự cô lập tuyệt đối giữa các phiên nghiên cứu khác nhau. Mỗi phiên LangGraph được gán một `thread_id` duy nhất, đóng vai trò là định danh cho toàn bộ không gian lưu trữ của phiên đó. Tất cả các tệp được tạo ra trong quá trình nghiên cứu, bao gồm dàn bài, dữ liệu đầu vào, mã

nguồn LaTeX và tài liệu PDF, đều được lưu trữ bên trong thư mục tương ứng với `thread_id`.

Cấu trúc thư mục này không chỉ giúp ngăn chặn rò rỉ dữ liệu giữa các người dùng mà còn hỗ trợ việc debug, audit và tái hiện kết quả nghiên cứu. Việc phân tách rõ ràng giữa thư mục `inputs`, nơi lưu trữ các mô tả ý tưởng và dữ liệu đầu vào, và thư mục `paper`, nơi chứa các phiên bản bài báo, phản ánh trực tiếp các giai đoạn trong quy trình nghiên cứu. Ngoài ra, việc cho phép tồn tại nhiều phiên bản bài báo (ví dụ: `paper_v4_final.tex`) tạo điều kiện cho việc theo dõi tiến trình phát triển nội dung và quay lui khi cần thiết.

Cách tiếp cận này đáp ứng trực tiếp yêu cầu về cô lập trạng thái (NFR6.1) và tạo nền tảng cho khả năng mở rộng ngang, bởi mỗi node xử lý có thể làm việc độc lập trên các thư mục luồng khác nhau mà không xảy ra xung đột dữ liệu.

```
project/
├── threads/
│   ├── {thread_id_1}/
│   │   ├── plan_outline.json
│   │   ├── paper/
│   │   │   ├── paper_v4_final.tex
│   │   │   └── paper_v4_final.pdf
│   │   └── inputs/
│   │       ├── data_description.md
│   │       ├── idea.md
│   │       └── results.md
│   └── {thread_id_2}/
│       └──...
```

3.4.2. Outline Data Model

Dàn bài đóng vai trò là cấu trúc dữ liệu trung tâm kết nối giữa giai đoạn lập kế hoạch và giai đoạn sinh nội dung trong hệ thống JANIS. Đây là đầu vào bắt buộc cho Denario và là cơ sở để các tác tử LLM hiểu được cấu trúc logic của bài báo khoa học cần được tạo ra. Do đó, mô hình dữ liệu dàn bài được thiết kế với các ràng buộc nghiêm ngặt nhằm đảm bảo tính hợp lệ và khả năng xử lý tự động.

Mỗi phần trong dàn bài được biểu diễn dưới dạng một đối tượng JSON với các trường bắt buộc bao gồm `id`, `title`, `description` và `order`. Trường `id` đóng vai trò là định danh duy nhất cho từng phần, cho phép hệ thống tham chiếu chính xác trong quá trình sinh nội dung và cập nhật trạng thái. Trường `title` và `description` không chỉ mang tính mô tả mà còn được sử

dụng trực tiếp như một phần của prompt, hướng dẫn LLM về nội dung học thuật cần trình bày trong từng mục. Trường `order` đảm bảo thứ tự tuần tự của các phần, phản ánh cấu trúc chuẩn của một bài báo khoa học và ngăn chặn các lỗi logic trong quá trình biên dịch tài liệu.

Việc áp đặt các ràng buộc này giúp giảm thiểu lỗi lan truyền trong các giai đoạn sau, đặc biệt là khi sinh nội dung LaTeX, nơi một lỗi nhỏ trong cấu trúc có thể dẫn đến thất bại toàn bộ quá trình biên dịch.

Table 3.4.2: Các Trường Bắt buộc trong Outline Section

Trường	**Kiểu dữ liệu**	**Mô tả**	**Yêu cầu**
id	string	Định danh duy nhất của phần.	Duy nhất, bắt buộc
title	string	Tiêu đề của phần.	Bắt buộc.
description	string	Nội dung cần trình bày trong phần	Dùng để hướng dẫn LLM sinh nội dung.
order	integer	Thứ tự xuất hiện trong bài báo.	Phải tuần tự

Chương IV: Giao diện, Bảo mật, Khả năng Mở rộng và Triển khai

4.1. Thiết kế API và Giao diện

Backend của hệ thống được xây dựng trên FastAPI, tận dụng khả năng tự động mount các tuyến API của LangGraph để giảm thiểu mã kết nối thủ công và đảm bảo tính nhất quán giữa logic tác tử và giao diện truy cập. API trung tâm của hệ thống là endpoint streaming cho phép frontend nhận luồng thực thi LangGraph theo thời gian thực, một yêu cầu thiết yếu để hiển thị tiến trình nghiên cứu và trạng thái tác tử một cách trực quan.

Bên cạnh đó, hệ thống cung cấp các endpoint chuyên biệt để truy xuất tệp theo `thread_id`, cho phép người dùng tải xuống các tạo phẩm nghiên cứu đã được sinh ra, bao gồm bài báo LaTeX và PDF. Endpoint chuyển đổi PDF sử dụng trình biên dịch xelatex để biên dịch tài liệu cuối cùng, trong khi endpoint kiểm tra trạng thái cho phép frontend truy vấn tiến trình của các tác vụ dài, chẳng hạn như sinh bài báo hoặc chạy thực nghiệm. Cách thiết kế API này đảm bảo sự tách biệt rõ ràng giữa giao diện người dùng và logic xử lý backend, đồng thời hỗ trợ mở rộng trong tương lai.

Table 4.1.1: Các Điểm cuối API Backend Chính

Endpoint (Method)	**Mô tả**	**Thành phần Xử lý**	**Mục đích**
POST /runs/stream	Truyền luồng thực thi LangGraph thời gian thực	LangGraph SDK, FastAPI	Giao tiếp thời gian thực, cập nhật UI.
GET /files/{thread_id}/{path:path}	Truy xuất tệp đã tạo theo ID thread	FastAPI, Thread Files	Tải xuống bài báo/tài liệu nghiên cứu.
POST /pdf/convert	Yêu cầu chuyển đổi LaTeX sang PDF (sử dụng xelatex)	FastAPI, LaTeX Compiler	Biên dịch tài liệu cuối cùng.
GET /status/{paper_id}	Kiểm tra trạng thái của quá trình sinh bài báo	FastAPI, Denario	Cung cấp phản hồi tiến trình.

4.2. Chiến lược Bảo mật

4.2.1. Quản lý Khóa API và Cấu hình Môi trường

Bảo mật khóa API là yêu cầu nền tảng đối với một hệ thống phụ thuộc vào nhiều dịch vụ LLM bên ngoài. Tất cả các khóa API, bao gồm khóa truy cập OpenAI và các nhà cung cấp khác, phải

được quản lý thông qua biến môi trường và tuyệt đối không được hard-code trong mã nguồn. Cách tiếp cận này giúp giảm thiểu rủi ro rò rỉ thông tin nhạy cảm và hỗ trợ triển khai linh hoạt trên nhiều môi trường khác nhau. Việc hỗ trợ cấu hình `OPENAI_BASE_URL` cũng cho phép hệ thống hoạt động thông qua các gateway hoặc proxy tập trung, tăng cường khả năng kiểm soát và giám sát lưu lượng.

4.2.2. Kiểm soát Truy cập và CORS

Do frontend và backend được triển khai tách biệt trên các cổng khác nhau, cấu hình CORS là bắt buộc để đảm bảo giao tiếp hợp lệ giữa hai thành phần. Ngoài ra, các endpoint truy xuất tệp theo `thread_id` cần được bảo vệ bằng cơ chế xác thực và kiểm soát truy cập, nhằm đảm bảo rằng người dùng chỉ có thể truy cập dữ liệu thuộc về phiên nghiên cứu của mình. Đây là yếu tố quan trọng để duy trì tính cô lập dữ liệu và đáp ứng các yêu cầu bảo mật cơ bản.

4.2.3. An toàn Thực thi Mã

Khả năng thực thi mã Python trong Denario, mặc dù cần thiết cho các nghiên cứu thực nghiệm, cũng tiềm ẩn nhiều rủi ro bảo mật. Các ràng buộc nhẹ được Denario áp dụng chỉ đủ để hạn chế chi phí và thời gian chạy, nhưng không đủ để đảm bảo an toàn trong môi trường production. Do đó, việc thực thi mã phải được cô lập hoàn toàn trong môi trường sandbox, chẳng hạn như một container Docker thứ cấp hoặc microVM. Cách tiếp cận này giúp ngăn chặn các hành vi độc hại hoặc lỗi thực thi ảnh hưởng đến hệ thống máy chủ và các phiên người dùng khác.

4.3. Chiến lược Khả năng Mở rộng và Vận hành

4.3.1. Tối ưu hóa Chi phí LLM và Đa Mô hình

Thiết kế của JANIS đặc biệt chú trọng đến việc tránh phụ thuộc vào một nhà cung cấp LLM duy nhất. Bằng cách sử dụng giao diện tương thích OpenAI cho tất cả các mô hình, hệ thống có thể linh hoạt chuyển đổi giữa các nhà cung cấp khác nhau mà không cần thay đổi logic nghiệp vụ. Lớp cấu hình mô hình cung cấp thông tin chi tiết về giá cả, cho phép hệ thống và người dùng đánh giá chi phí theo thời gian thực. Kết hợp với cơ chế theo dõi token streaming, người dùng có thể chủ động giám sát và dừng các phiên nghiên cứu tốn kém, từ đó tối ưu hóa ngân sách nghiên cứu.

4.3.2. Cân nhắc Triển khai

Hệ thống JANIS được thiết kế để triển khai theo mô hình tách biệt, trong đó backend và frontend có thể được triển khai độc lập. Backend, chạy trên cổng 2024, phù hợp để container hóa bằng Docker trong môi trường Linux, với yêu cầu cài đặt đầy đủ trình biên dịch xelatex. Khi mở rộng ngang, hệ thống cần sử dụng cơ chế lưu trữ trạng thái bền vững cho LangGraph để đảm bảo tính liên tục của các phiên nghiên cứu dài. Frontend, chạy trên cổng 3000 và được xây dựng bằng

Next.js, có thể dễ dàng mở rộng thông qua các nền tảng triển khai web hiện đại.

Nhờ nguyên tắc cô lập luồng cho cả trạng thái tính toán và lưu trữ tệp, hệ thống có thể mở rộng ngang bằng các công cụ orchestration như Kubernetes, cho phép xử lý đồng thời nhiều phiên nghiên cứu phức tạp mà không làm suy giảm độ ổn định hay tính bảo mật của hệ thống.