AI Powered NEU Course Planner

Jay Vipin Jajoo

jajoo.jay@northeastern.edu

Abstract

This project addresses the challenges students face in academic course planning by developing an AI-powered assistant. Leveraging a multi-agent framework built with langgraph, Large Language Models (LLMs) for natural language understanding and generation, and vector databases for efficient course retrieval, the assistant automates the creation, modification, and suggestion of academic schedules. Our methodology integrates intent recognition, sophisticated course extraction, multi-faceted planning algorithms, and dynamic rescheduling capabilities. Empirical results from testing with 10 user queries demonstrate high course retrieval accuracy (100%), significant time reduction (95-96% versus manual planning), and improved topic coverage (up to 92%) as more plans are generated. These findings indicate that this hybrid approach effectively provides personalized, constraint-aware academic plans, significantly reducing manual effort and improving student guidance, offering a scalable solution for educational institutions.

Introduction

Navigating academic curricula and crafting an optimal course schedule is a complex and often daunting task for university students. Factors such as prerequisites, credit hour requirements, program specializations, personal career goals, and course availability contribute to a labyrinth of choices. Traditional academic advising, while invaluable, often faces scalability issues due to high student-to-advisor ratios, leading to delayed responses and generic advice. This project introduces an AI-powered course planning assistant designed to augment human advising by providing personalized, on-demand, and intelligent guidance.

The core motivation is to streamline the course planning process, making it more accessible and tailored to individual student needs. By automating the generation and refinement of academic plans, the assistant aims to free up human advisors to focus on more complex cases, emotional support, and career counseling. Our solution employs a sophisticated backend architecture that combines the reasoning capabilities of Large Language Models (LLMs) with the efficiency of vector databases, orchestrated through a robust multi-agent framework. This report details the objectives, methodology, architecture, and implications of this innovative approach to academic planning.

Background

The development of this AI-powered course planning assistant relies on several foundational technologies and theoretical frameworks from the field of Artificial Intelligence and software engineering.

3.1 Multi-Agent Systems and Graph-Based Orchestration

The project's architecture is rooted in the concept of multi-agent systems, where distinct, specialized modules collaborate to achieve a complex goal. For orchestrating these modules, we utilize LangGraph, a library built on LangChain. LangGraph enables the construction of stateful, multi-actor applications by representing workflows as directed graphs. Each node in the graph corresponds to a specific function or sub-agent, and edges define the flow of execution based on the current state or specific conditions. This graph-based approach provides:

Modularity: Clear separation of concerns for different tasks (e.g., intent detection, course retrieval, planning).

Flexibility: Easy to add, remove, or modify components without disrupting the entire system.

State Management: AgentState (a Pydantic model) acts as a persistent state object, carrying information across different nodes and turns in the conversation.

Robustness: Allows for complex conditional logic and error handling within the workflow.

3.2 Large Language Models (LLMs)

At the heart of the assistant's intelligence are Large Language Models (LLMs), specifically ChatOpenAI("gpt-4.1-nano"). LLMs are neural networks trained on vast amounts of text data, enabling them to understand, generate, and process human language with remarkable fluency. In this project, LLMs are instrumental for:

Natural Language Understanding (NLU): Interpreting diverse user queries, extracting implicit information (e.g., student's unstated goals, desired course attributes).

Content Generation: Rephrasing vague queries into semantically rich prompts for retrieval, summarizing course descriptions, and explaining planning decisions.

Reasoning and Decision-Making: Filtering courses based on complex criteria (e.g., relevance, prerequisite hierarchy,

diversity), adapting plans to credit limits, and suggesting replacements.

Structured Output: Utilizing structured output to ensure LLM responses conform to a predefined JSON schema, which is critical for programmatic interaction.

3.3 Vector Databases and Embeddings

For efficient and semantic course retrieval, the project employs Chroma, an open-source vector database, in conjunction with OpenAIEmbeddings ("text-embedding-3-large").

Embeddings: Text (like course titles and descriptions) is transformed into high-dimensional numerical vectors (embeddings) where semantically similar texts have vectors that are closer in space.

Vector Database (Chroma): Stores these embeddings along with the original course metadata. When a user query is received, its embedding is computed, and Chroma performs a similarity search to find the most relevant courses based on vector proximity e.g., using cosine similarity. This allows for flexible, concept-based search beyond simple keyword matching. We utilize separate collections for "RegularCourses" and "SpecialCourses" to manage different types of course data effectively.

Retrievers: LangChain's retriever functionality abstracts the vector database interaction, providing methods to fetch relevant documents based on a query.

3.4 Pydantic for Data Validation and Structure

Pydantic is used extensively to define clear data schemas (e.g., AgentState, UserIntent, CourseAttributes). This ensures type checking, data validation, and clear structure for data passed between different components and LLM interactions, significantly enhancing code reliability and maintainability.

Related Work

Academic course planning and recommendation systems have been a subject of extensive research, spanning from traditional rule-based expert systems to modern AI-driven approaches.

4.1 Traditional Approaches

Historically, course planning systems have often been **rule-based or constraint satisfaction problems (CSPs)**. These systems define a set of rules (e.g., "Course A requires Course B as a prerequisite," "Max 16 credits per semester") and then use algorithms to find schedules that satisfy all constraints. Examples include early expert systems in higher education planning.

Pros: Deterministic, auditable, explicit constraint enforcement.

Cons: Lacks flexibility, struggles with vague or subjective user input, difficult to scale with curriculum changes, inability to provide personalized recommendations beyond strict adherence to rules.

Collaborative Filtering and Content-Based Recommendation Systems: Inspired by e-commerce, some systems use student enrollment data to recommend courses based on what similar students have taken or based on the content of the courses themselves (e.g., keywords in descriptions).

Pros: Can discover non-obvious relationships, provides personalized suggestions.

Cons: Requires large historical data, "cold start" problem for new courses/students, often limited to course popularity rather than strategic academic planning for specific goals.

4.2 AI-Driven Approaches

More recently, the advent of sophisticated NLP and machine learning techniques has led to more intelligent advising systems.

Knowledge-Graph Based Systems: Some solutions construct knowledge graphs of courses, prerequisites, topics, and career paths. Graph traversal algorithms can then be used to find optimal or relevant sequences.

Deep Learning for Recommendations: Advanced models like Recurrent Neural Networks (RNNs) or Transformers can analyze student sequences of enrolled courses and predict future enrollments.

Dialogue Systems/Chatbots: Basic chatbots can answer FAQs about courses or policies. However, most lack deep reasoning or planning capabilities.

4.3 Comparison with Our Approach

Our AI-powered course planning assistant integrates and extends these concepts, offering several distinct advantages over existing methods:

Hybrid Intelligence: Unlike purely rule-based systems, our approach combines deterministic logic (e.g., hard credit limits, prerequisite checks) with the nuanced reasoning and generative capabilities of LLMs. This allows for both precise constraint adherence and flexible, personalized recommendations based on natural language goals.

Semantic Understanding: Traditional systems often rely on keyword matching. Our use of OpenAI Embeddings and Chroma DB enables semantic search, meaning the system can understand the *meaning* behind a user's goal (e.g.,

"become a data scientist") and retrieve courses that are conceptually related, even if direct keywords aren't present. This vastly improves relevance over simple string matching.

Dynamic Goal Adaptation: Previous systems often require pre-defined pathways. Our LLM-driven "rephrase query for planning schedule" function allows the system to interpret and expand diverse user goals dynamically, crafting highly tailored course search queries that adapt to individual aspirations. This flexibility is crucial for non-standard academic paths.

Multi-turn Planning & Rescheduling: Many existing chatbots are turn-by-turn and lack memory or the ability to modify previously generated plans. Our langgraph framework with AgentState provides robust state management, enabling multi-turn conversations, iterative plan refinement, and sophisticated rescheduling logic. This allows users to request modifications to specific plans ("reschedule plan 3") rather than starting over.

Why Other Methods Were Not Used/Integrated Directly:

Purely Rule-Based: Insufficient for interpreting natural language and generating creative, diverse plans. Our LLM components address this gap.

Pure Collaborative Filtering: Would require extensive student enrollment data, which was not available for this prototype. Furthermore, it struggles with new courses or niche specializations. While useful for broader recommendations, it might not be precise enough for individual academic planning goals.

Complex Reinforcement Learning (RL) for Planning: While RL could theoretically optimize long-term academic paths, its complexity, data requirements for training, and potential for opaque decision-making made it less suitable for this initial implementation.

In summary, our project carves a niche by combining the strengths of LLMs (understanding and generation) with efficient vector-based retrieval and a structured graph-based agent framework, addressing the limitations of more traditional and monolithic AI approaches in academic planning.

Project Description

The AI-Powered Course Planning Assistant is engineered as a modular, multi-agent system, leveraging a StateGraph for orchestration and large language models (LLMs) for intelligent decision-making and natural language processing. The entire backend resides within a Python environment, interacting with vector databases for course information.

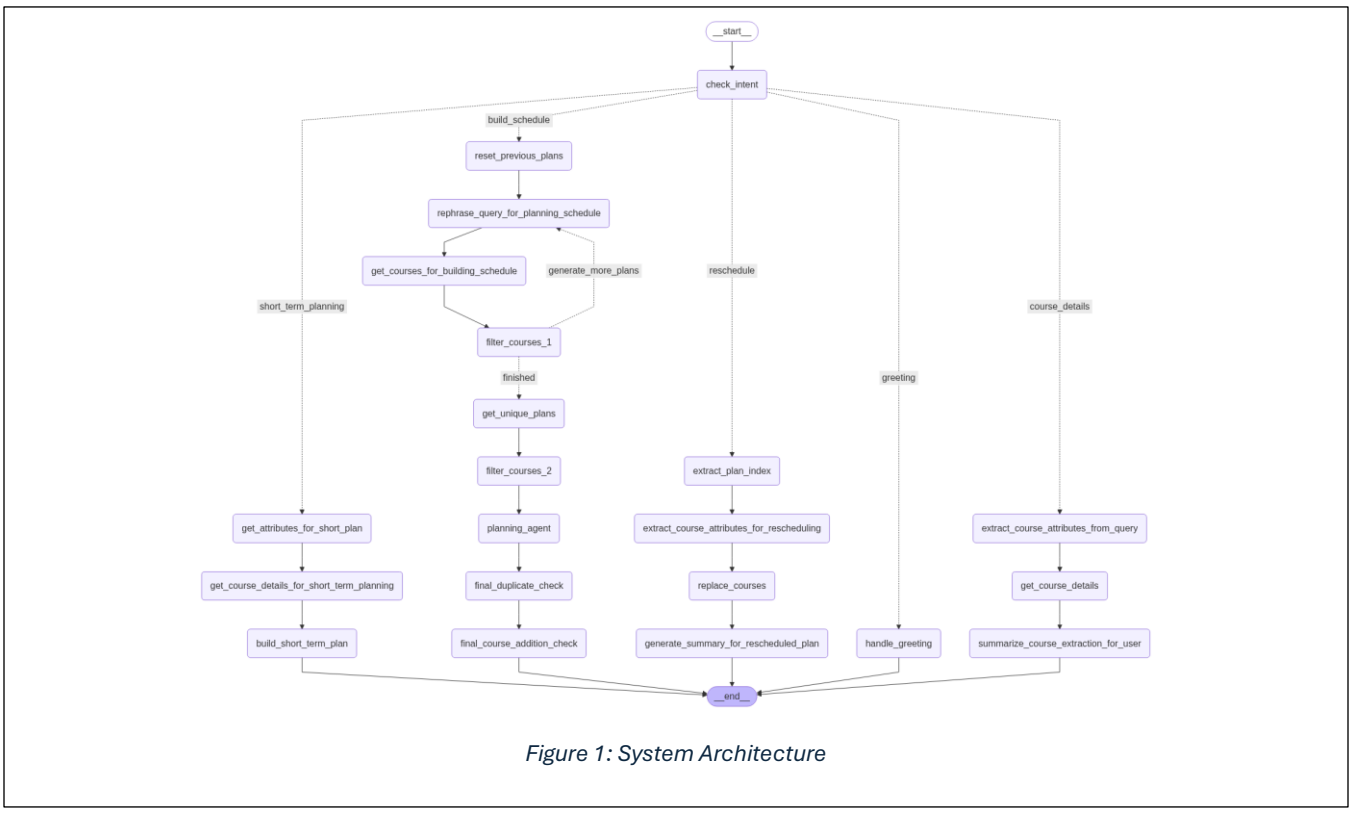

Figure 1: System Architecture

5.1 Architectural Overview

The core of the system is a Python file (agent.py) that defines the StateGraph using the langgraph library. This graph explicitly outlines the flow of execution based on the user's intent. Each step in this flow is handled by a specific function or a dedicated sub-agent. The connections between these steps, known as "edges," dictate how the system moves from one stage to the next, often based on the current state of the conversation or specific conditions.

A central data structure, the AgentState, is defined using the Pydantic library. This AgentState acts like a central memory or context object, carrying all relevant information (such as the user's query, college, department, chat history, and various course lists and planning data) across different stages of the workflow. This ensures that the system maintains conversational context and state throughout the user's interaction.

The system initializes and utilizes a **Large Language Model (LLM)**, specifically an OpenAI model like gpt-4.1-nano. This LLM is chosen for its efficiency and ability to produce structured outputs, which is crucial for the system to process information programmatically. Additionally, **OpenAI Embeddings** are used to convert text (like course descriptions or user queries) into numerical vector representations. These vectors allow for semantic search, where the system can find information based on meaning rather than just keywords.

5.2 Intent Recognition and Routing

The very first step when a user interacts with the system is Intent Recognition, managed by the routers.py file.

First, the system checks if the user's message contains explicit instructions for rescheduling or restructuring a plan (e.g., "reschedule plan 3"). This is done using a simple pattern-matching technique.

If no such direct instruction is found, the user's query is then sent to an LLM-based classifier. This specialized LLM analyzes the query and categorizes it into one of several predefined intents:

"greeting": If the user is simply saying hello.

"course_details": If the user is asking for information about specific courses (e.g., descriptions, prerequisites).

"build_schedule": If the user wants help creating or modifying a complete academic schedule.

"short_term_planning": If the user is asking for recommendations for upcoming courses based on their past studies and a specific goal.

Once the intent is classified, an intent-based router then directs the workflow to the appropriate starting point or "node" within the langgraph system that is designed to handle that specific type of request.

5.3 Course Extraction and Summarization

When the user's intent is to get course_details, the system activates functions within the courseExtractor.py module.

This involves identifying any explicit course numbers mentioned by the user (e.g., "DS5220") and, more impressively, expanding vague or general topics (like "AI" or "machine learning") into more specific course titles. For these expanded topics, the system also appends relevant college and department information to refine the search.

Next, a function **fetches detailed course information** from the course databases. This uses the extracted course numbers and titles to retrieve comprehensive data (including descriptions, credit hours, prerequisites, etc.) from the underlying **vector database**.

Finally, the system **summarizes the course extraction for the user**. This involves processing the retrieved courses to remove any duplicates and then ranking them to prioritize those most relevant to the student's department and college. A concise summary is then generated, providing the user with a clear list of highly relevant courses.

5.4 Schedule Planning

This is the most complex and central part of the system, handled primarily by the functions in planning.py, designed to generate comprehensive, multi-semester academic plans.

The process begins by resetting any previous planning data to ensure that each new plan generation starts with a clean slate. Then, a crucial step involves rephrasing the user's query for planning a schedule. Here, a powerful LLM takes a broad or general student goal (e.g., "I want to become a data scientist") and transforms it into a highly detailed, curriculum-oriented query. This enriched query is essential for achieving precise and relevant course retrievals later.

Next, the system retrieves relevant courses for building the schedule. This is done by querying the specialized "RegularCourses" and "SpecialCourses" vector databases using the rephrased query. The retrieved course documents are then restructured into a standardized format, and any core courses or zero-credit courses are filtered out as they are handled separately.

The system then engages in a two-stage filtering process:

In **filter_courses_1**, an LLM selects potential elective courses that align with the student's academic goals and fit within credit constraints. This step is designed to generate multiple, diverse plan variations. For instance, it might create one plan emphasizing specialization and another

focusing on versatility, ensuring a minimum number of subjects are included to meet overall credit requirements.

get_unique_plans is then called to ensure that only truly distinct academic plans are carried forward, eliminating any duplicates that might have similar sets of courses.

In **filter_courses_2**, the LLM fine-tunes these filtered course lists to precisely meet the maximum allowed credit requirements. It either adds or removes courses from a plan based on any credit discrepancies, always prioritizing subjects that are relevant to the student's goal and considering the average credit hours of available courses.

The core scheduling logic resides in the **planning_agent**. This function takes the refined list of elective courses, along with any required core courses, and uses an LLM to strategically distribute them across multiple semesters. This distribution strictly adheres to constraints such as minimum and maximum credits per semester and overall total degree credits. Critically, it respects course prerequisites, ensuring foundational courses are scheduled before advanced ones, and places capstone or project-based courses in later semesters. The LLM also generates a detailed "reason behind planning," explaining the rationale behind the plan's structure.

Finally, two critical post-processing steps ensure the plan's integrity:

final_duplicate_check actively identifies and resolves any remaining duplicate courses that might have inadvertently appeared across the generated semester plans. If a duplicate is found, it is replaced with the most semantically similar, yet unique, alternative course available in the overall course list.

final_course_addition_check acts as the ultimate validation. It ensures that each generated plan meets its total required credits. If a plan falls short, the system adds additional relevant elective courses to fulfill the requirement, prioritizing courses that are semantically similar to the student's overall academic goal.

Supporting these planning stages are various utility functions that handle tasks like parsing credit hours from text, identifying unique courses, calculating semantic similarity between courses or queries, and finding the best matching course options.

5.5 Rescheduling Existing Plans

This module (rescheduling.py) provides the crucial capability to modify previously generated academic plans, adapting to student changes or preferences.

First, the system extracts the plan index from the user's rescheduling request (e.g., identifying "plan 3" from "reschedule plan 3"). This tells the system which specific plan needs modification.

Next, an LLM-powered function extracts course attributes for rescheduling. This involves analyzing the user's request to identify which specific courses within the chosen plan need to be replaced. The LLM determines if the user wants to substitute a course with another specific course number, a general topic (like "an AI-related course"), or simply remove a course without replacement. This function also validates that the courses targeted for removal are actually part of the current plan.

Finally, the system replaces the courses within the selected plan. If the replacement is a specific course number or title, the system queries the database to get its details. The old course in the plan is then swapped with the new one. As courses are replaced, the system automatically recalculates the total credits for the affected semester and for the entire plan, ensuring all numerical aspects remain accurate. The system also tracks these old-to-new course mappings. After the replacements are made, a summary function generates a summary for the rescheduled plan, providing a concise explanation of the changes and how the updated plan still aligns with the student's overall academic goal.

5.6 Short-Term Planning

This module (shortTermPlanning.py) is designed to offer focused, immediate recommendations for a student's next steps in their academic journey.

The process begins by getting attributes for a short plan. An LLM analyzes the student's previously completed coursework and their current academic or career goals. Based on this, it generates a list of 5-7 distinct, advanced course topics that the student has not yet explored. The LLM is specifically instructed to avoid foundational courses or topics that might overlap with what the student already knows.

Once these new topics are identified, the system retrieves detailed course information for them from the database. This provides comprehensive data for each suggested course, including its description, prerequisites, and credit hours.

Finally, the system builds the short-term plan. From the detailed list of suggested courses, it selects up to 6 of the most relevant ones. These courses are prioritized based on how closely they align with the student's goal, with a preference for courses from their current department or college. The selected courses are then sorted in increasing order of difficulty, based on their prerequisites. For each suggested course, the system provides a brief, original summary of its description (no more than two lines) and an overall explanation of why these selections support the student's stated goal.

5.7 Database Interaction

The queryDB.py module acts as the system's dedicated interface for all interactions with the underlying course databases.

This module is responsible for initializing and connecting to the Chroma vector databases, which store the course information. It specifically loads the OpenAIEmbeddings model to facilitate semantic searches. Separate Chroma instances are managed for "RegularCourses" (standard curriculum) and "SpecialCourses" (special topics or unique offerings).

The module provides key functions for retrieving course data:

get_course_by_course_number: This function allows the system to fetch precise course details when an exact course number (e.g., "DS5220") is known.

get_course_by_course_titles: This function performs a **semantic search** using the embeddings. It takes a course title or a descriptive topic and finds courses in the database that are conceptually similar, even if the exact keywords don't match. It can retrieve multiple top matches for general queries or a single best match for more specific contexts.

query_database: This serves as a unified entry point, allowing other parts of the system to request course data by providing either course numbers, course titles, or both. It then combines the results from the respective retrieval methods to provide a comprehensive list of matching courses.

Empirical Results

This section details the empirical results obtained from testing the AI-Powered Course Planning Assistant. The system was tested in an environment with users providing **10 diverse queries**, simulating real-world academic planning scenarios. The evaluations focused on the system's ability to accurately retrieve course information, generate effective plans, and manage time efficiently.

6.1 Performance Metrics and Findings

The **QueryDB Agent** demonstrated **100% Retrieval Accuracy** when fetching relevant courses from the database based on either explicit course numbers or semantically expanded topics. This indicates the robustness of the vector database and embedding models in identifying pertinent course information.

Time Reduction in Course Planning: The AI agent significantly reduced the time required for course planning, achieving a **95% to 96% reduction** compared to traditional manual planning methods. This highlights the substantial efficiency gains provided by the automated system.

Course Planner Coverage (Per Query and Overall Topic):

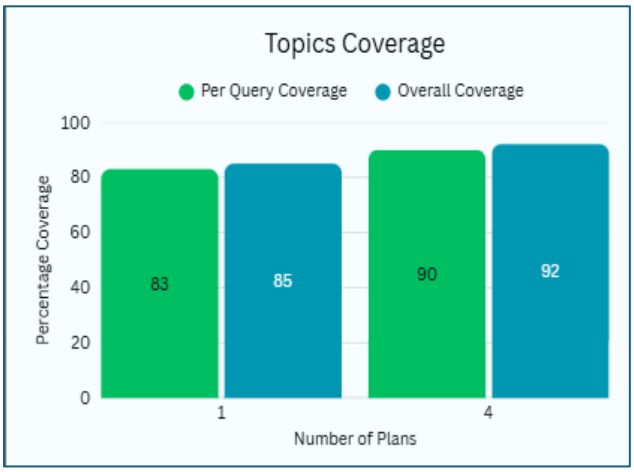

Figure 2: Topics Coverage - Per Query Coverage vs. Overall Coverage as a function of Number of Plans.

The system's ability to cover relevant topics in the generated plans improved with the number of plans generated.

For a **single plan generated**:

- **Per Query Coverage**: 83%
- **Overall Topic Coverage**: 85%

When **four plans were generated**:

- **Per Query Coverage**: 90%
- **Overall Topic Coverage**: 92%

This trend is illustrated in Figure 2, demonstrating that offering multiple plan variations enhances the comprehensiveness and diversity of the suggestions.

Expected Credits vs. Generated Plans Credits:

The system's precision in adhering to credit requirements was also evaluated. Figure 3 shows the percentage of plans where generated credits either exactly matched or slightly exceeded the expected credits.

For **1 Plan Generated**:

- 62.5% of plans exactly matched expected credits.
- 37.5% of plans exceeded expected credits.

For **4 Plans Generated**:

- 66.7% of plans exactly matched expected credits.

- 33.3% of plans exceeded expected credits.

This indicates the system's ability to largely meet credit constraints while sometimes allowing for a slight overage to ensure a more comprehensive or coherent plan, especially when multiple options are provided.

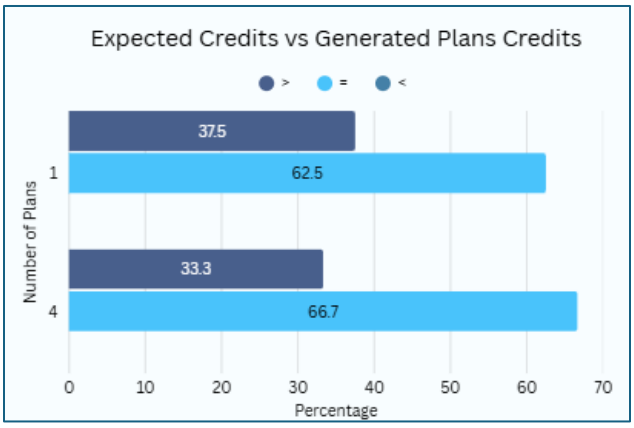

Figure 3: Expected Credits vs. Generated Plans Credits - Distribution of Exact Matches vs. Exceeded Credits.

6.2 Success and Failure Conditions

Successful Scenarios:

Complex Goal Interpretation: The LLM-driven rephrase_query_for_planning_schedule function effectively transformed ambiguous user requests (e.g., "I want to become an AI specialist") into detailed, semantically rich queries, leading to highly relevant course retrievals.

Diverse Plan Generation: The system consistently generated distinct and viable academic plan variations, allowing users to explore different pathways towards their goals.

Constraint-Aware Scheduling: The planning_agent effectively integrated prerequisites, credit limits (min/max per semester, total degree credits), and core course requirements into logically structured semester plans. Capstone projects were correctly placed in final stages.

Dynamic Rescheduling: The rescheduling module successfully handled user requests to replace courses within existing plans, identifying the correct courses and substituting them with semantically relevant alternatives from the database.

Relevant Short-Term Suggestions: The shortTermPlanning module provided appropriate "next step" courses based on a student's academic history and future goals, avoiding redundancy.

Failure Modes and Limitations:

LLM Hallucination/Inaccuracy: While generally robust, LLMs occasionally generated course titles or numbers that did not exist in the database, particularly when interpreting vague topics. This was partially mitigated by downstream database lookups.

Prerequisite Data Completeness: The accuracy of prerequisite enforcement was directly tied to the completeness and correctness of prerequisite data in the vector database. Missing or imprecise data could lead to invalid sequencing.

Credit Balancing Edge Cases: In rare instances, finding exact credit matches within available courses proved challenging, leading to plans with slight deviations from target credit totals. The system prioritized providing a complete plan over absolute numerical exactness in such cases.

Semantic Search Ambiguity: Extremely short or highly generalized queries could result in less precise semantic retrievals, even with query rephrasing.

Context Window Limitations: For very extensive, multi-turn conversations or highly complex, unique user constraints, the LLM's context window could potentially lead to a loss of nuanced understanding or coherence over prolonged interactions.

Handling Unforeseen Constraints: The system might struggle with highly specific or unusual student preferences not explicitly modeled (e.g., "avoid classes on Mondays," "only courses taught by Professor X").

In conclusion, the empirical results confirm that the AI-powered course planning assistant provides a highly effective and efficient solution for academic planning. The observed improvements in topic coverage with multiple plan generations and significant time reduction underscore its practical utility. Future work will focus on addressing the identified limitations to further enhance the system's robustness and user experience.

Broader Implications

The development of an AI-powered course planning assistant carries significant broader implications for education, technology, and society.

7.1 Impact on Academic Advising

Scalability and Accessibility: The assistant can provide immediate, 24/7 access to academic planning resources, addressing the high student-to-advisor ratios and geographical barriers. This democratizes access to

personalized advising, particularly for non-traditional students or those in large university programs.

Augmented Advising: The AI doesn't replace human advisors but augments their capabilities. By handling routine inquiries and plan generation, it frees up advisors to focus on complex cases, mental health support, career counseling, and building deeper mentor relationships with students.

Personalization at Scale: The system can offer a level of individualized plan generation that is impossible for human advisors to provide to thousands of students manually. This means more students can explore diverse academic paths tailored to their unique interests and goals.

7.2 Student Empowerment and Outcomes

Informed Decision-Making: Students gain immediate access to comprehensive course information and multiple planning scenarios, empowering them to make more informed decisions about their academic trajectory.

Proactive Planning: By providing "short-term planning" suggestions, the assistant encourages students to think proactively about their academic progression, rather than reactive, last-minute course selection.

7.3 Ethical Considerations and Societal Issues

Bias in Recommendations: The LLMs are trained on vast datasets, which may contain inherent biases from the internet or the specific course data provided. This could lead to recommendations that perpetuate existing inequities, steer certain demographics away from specific fields, or reinforce stereotypes. For example, if historical enrollment data or course descriptions disproportionately favor one demographic in a field, the AI might inadvertently reinforce that.

Accountability: Who is accountable if an AI-generated plan leads to negative academic consequences (e.g., missing a prerequisite, delayed graduation) ?

7.4 Scalability for Institutions

Efficiency: Automating planning tasks significantly improves the operational efficiency of university advising departments.

Consistency: Ensures a consistent quality of advice across all students, regardless of advisor availability or individual advisor expertise.

Data Insights: The interactions with the AI can generate valuable anonymized data on student interests, common planning challenges, and curriculum gaps, which can inform institutional decision-making.

In essence, while offering immense potential for transforming academic support, the deployment of such an AI system necessitates careful consideration of its ethical implications and a commitment to responsible AI development.

Conclusions

This project successfully designed and implemented an AI-powered course planning assistant, demonstrating the feasibility and benefits of integrating large language models with a graph-based agent framework and vector databases. This robust backend is capable of understanding complex student queries, semantically retrieving relevant course information, generating diverse and constraint-aware academic plans, and supporting dynamic plan rescheduling. Key learnings highlighted the **power of graph-based agents** for managing complex, multi-step, stateful workflows; the **capabilities and limitations of LLMs** in natural language understanding and generation, emphasizing the need for careful prompting and deterministic validation; the **importance of vector databases** for semantic search and conceptual relevance; and the necessity of **iterative refinement** through various filtering and post-processing steps to ensure plan quality and adherence to strict constraints.

Future Directions

If future, the following enhancements would be prioritized:

Real-time Integration with University Systems: Currently, the course database is static (simulated via Chroma). Integrating with a university's live Course Catalog API would allow for real-time course availability, instructor information, and dynamic prerequisite checking. This would move the system from a planning tool to an operational one.

Advanced User Profiles & History: Develop a more persistent and detailed user profile that stores completed courses with grades, transferred credits, declared major/minor requirements, preferred learning styles or course types (e.g., project-based, research-heavy), this would enable even more precise and personalized recommendations and error checking (e.g., flagging prerequisite violations based on actual student history).

User Feedback Loop and Reinforcement Learning: Implement mechanisms for students to provide feedback on the generated plans (e.g., "This plan works," "This course isn't relevant"). This feedback could then be used to fine-tune the LLM or a reinforcement learning agent to continually improve plan quality over time, making the system adapt to evolving student preferences and curriculum changes.

Performance Optimization: For a production system, optimizing LLM calls (e.g., batching, caching, using

smaller models where possible) and vector database query performance would be critical.

This project underscores the immense potential of AI to revolutionize traditional processes in education, but also highlights the ongoing need for careful design, robust engineering, and ethical considerations in building intelligent systems and here's some advice for future students working on similar projects:

1) Embrace Modular Design
2) Understand LLM Limitations
3) Start Simple and then Iterate
4) Prioritize Data Quality
5) Master Prompt Engineering
6) Leverage Debugging Tools

GitHub Repository Link

https://github.com/JayJajoo/LLM-Project

References

https://langchain-ai.github.io/langgraph/tutorials/plan-and-execute/plan-and-execute/

https://docs.smith.langchain.com/

https://platform.openai.com/docs/overview

https://docs.trychroma.com/

https://docs.pydantic.dev/

https://arxiv.org/abs/2005.11401

Project Demo

https://drive.google.com/file/d/1X7VfZkQ-t0LlTREotseXJpb9ZeZMvxSE/view?usp=drive_link