Ecole Supérieure d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

الجامعة الدولية للرباط
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵜⴰⵖⵔⴰⴱⵜ ⵏ ⵔⵕⴱⴰⵟ
Université Internationale de Rabat
L'UNIVERSITÉ INNOVANTE

End of Semester Project Delivery :
Movie Recommendation System Project Using Neo4j & GNN.

Presented By :Yusra AL AISSAMI
Nohayla BEN LACHHEB
Imad EL MAKAOUI
Mohamed Taha EL MAHDAOUI

Supervised by : Hamza GAMOUH

Academic year : 2024/2025

Major : Big Data & AI

Table of content :

Abstract

This project focuses on the development of a movie recommendation system using advanced AI and database technologies. The system leverages a ***Graph Neural Network (GNN)*** model and a ***Neo4j graph database*** to provide personalized movie recommendations based on user preferences and interactions. By modeling the relationships between movies, genres, users, and other entities in a graph structure, the system can deliver more accurate and context-aware suggestions. The project involved extracting data from ***Neo4j***, converting it into a graph format using ***PyTorch Geometric***, and using ***GNN-based embeddings*** for recommendation generation. Despite facing challenges, such as the integration of the ***GNN model*** and limitations in frontend development, solutions were implemented to ensure functionality and performance. The system was deployed using a ***Flask API***, connecting the frontend interface with the backend processes. Future enhancements to improve scalability, real-time updates, and model refinement are also discussed. This project demonstrates the practical application of AI and graph database technologies in building a dynamic, data-driven recommendation engine.

Introduction

1.1 Project Overview :

Recommendation systems have become an integral part of digital platforms, enhancing user experiences by delivering personalized content and suggestions. They serve as a powerful tool for filtering vast amounts of information and presenting users with relevant options, improving both user satisfaction and platform efficiency. This project focuses on developing a movie recommendation system that leverages advanced artificial intelligence techniques, particularly a Graph Neural Network (GNN) model, and the Neo4j graph database. The system aims to analyze user preferences and interactions to provide accurate and personalized movie recommendations.

1.2 Objective and Problem Definition

The primary objective of this project is to design and implement an AI-powered recommendation system capable of predicting movies that align with a user's preferences. Traditional recommendation systems often rely on collaborative filtering or content-based filtering, which may struggle to capture complex relationships between data points like user preferences, movie genres, or actor networks.

This project addresses these challenges by integrating:

- Graph Neural Networks (GNNs), which excel at processing relational data and capturing nuanced patterns in user-movie relationships.
- Neo4j, a graph-oriented database, to model and store data in a way that mirrors real-world connections.

By combining these technologies, the system aims to overcome limitations of traditional methods and deliver more accurate, context-aware recommendations.

Problem Definition

2.1 Context and Motivation

Recommendation systems have become indispensable in modern applications, influencing user engagement and platform success. Companies like Netflix and Spotify exemplify the transformative potential of effective recommendation engines, which play a central role in attracting and retaining users. The key to maintaining user interest lies in delivering personalized, relevant, and engaging content, ensuring a seamless experience that keeps users returning to the application.

However, achieving this requires overcoming significant challenges in managing complex relationships, adapting to evolving user behavior, and delivering accurate predictions. Traditional approaches, such as collaborative or content-based filtering, often fall short in capturing the intricate relationships between users, items, and their preferences, limiting the system's ability to offer context-aware recommendations.

2.2 Identified Challenges

1. Managing Complex Relationships:
    - Users, movies, genres, and other entities form a web of interconnected relationships that are difficult to represent and analyze using traditional databases.
    - Modeling these relationships requires a robust and flexible solution, such as a graph database, to store and query data effectively.
2. Dynamic Learning and Adaptation:
    - User preferences and behaviors evolve over time, necessitating a system that can learn from interactions and adapt its recommendations accordingly.
    - Static models struggle to capture these shifts, leading to stale or irrelevant suggestions.
3. Accuracy of Recommendations:
    - Predicting user preferences demands an advanced AI model capable of understanding both explicit and implicit relationships within the data.
    - Traditional algorithms often fail to leverage the full potential of relational data, resulting in suboptimal predictions.

Technologies Employed

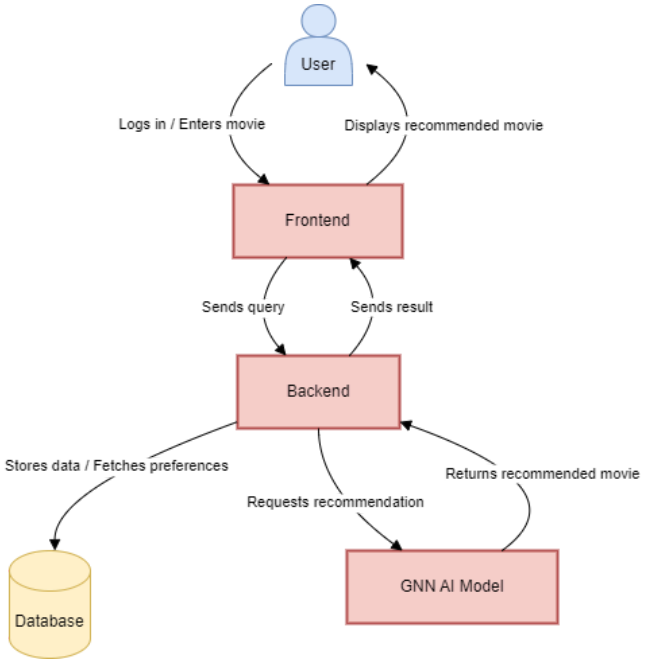
The development of the movie recommendation system involves a combination of cutting-edge tools and technologies, each contributing to a specific aspect of the project:

1. **Graph Neural Network (GNN) AI Model**:
    - Utilized for generating node embeddings and making accurate recommendations based on the graph structure.
    - Built using **PyTorch Geometric**, a library tailored for graph-based deep learning.
2. **Neo4j Graph Database**:
    - Serves as the primary data store, modeling the relationships between movies, users, genres, and other entities.
    - Efficiently handles complex queries and relationship-based data.
3. **Python Programming Language**:
    - Powers the backend, enabling integration with Neo4j, the AI model, and other components.
    - Libraries like **scikit-learn** are used for data preprocessing and evaluation.
4. **Frontend Technologies**:
    - **HTML**, **CSS**, and **JavaScript** are employed for creating an intuitive and responsive user interface.
    - Ensures seamless interaction between users and the recommendation system.

These technologies collectively enable a robust, scalable, and user-friendly recommendation system, addressing both data processing and user interaction requirements.

Architecture

4.2 Data Flow Diagram



The architecture described above includes the following components and interactions:

- **Frontend**: Interacts with the user, allowing login or movie entry, and displays the recommended movie.
- **Backend**: Handles user requests and communicates with other components.
- **GNN AI Model**: Processes user preferences to recommend movies.
- **Database (Neo4j)**: Stores and fetches user preferences and movie data.
- **User interactions**:
    - Logs in or enters movie preferences.
    - Sends queries for recommendations.
- **Workflow**:
    - Data is stored or fetched from the database.
    - Recommendations are requested from the GNN AI model.
    - Results are sent back to the frontend for display.

Database and Backend Integration

5.1 Neo4j Data Model and Queries

The Neo4j database was used to manage and structure the data as a graph, perfectly aligning with the requirements of the recommendation system. It efficiently stores nodes (movies, users, genres) and relationships (e.g., "liked_by," "acted_in"). Queries in Neo4j were designed to extract relevant data and relationships, setting the stage for graph representation and processing.

5.2 Data Extraction and Graph Representation with PyTorch Geometric

After querying Neo4j, the extracted data was converted into a graph format using **PyTorch Geometric**, which plays a critical role in preparing the dataset for processing by the GNN. This process involves two key steps:

1. **Data Extraction**: Neo4j queries were used to extract nodes (entities like movies, users, and genres) and edges (relationships such as "liked_by," "acted_in"). This raw data captures the interconnected structure required for graph representation.
2. **Graph Representation**:
    - Using PyTorch Geometric, the extracted data was converted into a graph object.
    - **Node Definition**: Nodes were assigned to entities like movies, users, and genres, and features such as metadata (e.g., movie genres, ratings) were embedded to provide additional context.
    - **Edge Creation**: Relationships between nodes were represented as edges, defining the graph's structure.
    - **Graph Object Construction**: The resulting graph object encapsulated the entire dataset in a format optimized for graph-based computations.

This transformation ensures the data is ready for use in GNN-based processing, leveraging both the content and the relationships within the graph.

5.3 Backend GNN Model Integration

The core of the backend involved integrating a GNN model. The model generated **node embeddings**, representing movies and other entities in a high-dimensional space. This part was divided into:

1. **Embeddings Generation**: Capturing the graph's structure and relationships.
2. **Learning Non-linear Patterns**: Using non-linear activation functions to uncover complex patterns.

3. The result was a rich set of embeddings that effectively represented the dataset.

5.4 API Integration

To expose the system's functionality, we built a RESTful API using the **Flask** framework. Flask was chosen for its simplicity and flexibility in creating lightweight, scalable web applications. The API provides endpoints that allow users to:

- **Fetch Movies**: Retrieve movies or recommendations based on specific queries.
- **Return Results**: Deliver recommendations back to the user or frontend.

These endpoints facilitate seamless communication between the frontend and backend, ensuring a smooth user experience.

5.5 Recommendation Generation

The final step combined embeddings with user queries to generate personalized recommendations. By applying distance metrics like **Euclidean distance** or **cosine similarity**, the system matched user preferences with movie embeddings, producing a list of recommended movies.

This workflow highlights the seamless integration of Neo4j, PyTorch Geometric, Flask, and the GNN model, ensuring that the backend efficiently supports dynamic, scalable, and accurate recommendations.

Frontend Development

6.1 User Interface Design and Features

The frontend of the movie recommendation system was designed to provide an intuitive and user-friendly interface that allows users to interact seamlessly with the application. It was built using a combination of **HTML**, **CSS**, and **JavaScript**, chosen for their simplicity, flexibility, and wide compatibility across browsers.

- **HTML**:
  Structured the content and layout of the application. It served as the backbone for designing forms, displaying recommendations, and presenting user information.
- **CSS**:
  Styled the application to enhance visual appeal and usability. CSS was used to create responsive designs, ensuring the application adapts well to different screen sizes and devices.
- **JavaScript**:
  Enabled interactivity and dynamic functionality. It handled actions like:
  - Fetching movie recommendations from the API.
  - Displaying results dynamically without reloading the page.
  - Managing user inputs and interactions.

6.2 Interaction with Backend Services

The frontend communicates with the backend via the RESTful API endpoints created using Flask. This integration enables the following functionalities:

- **Fetching Data**: User queries are sent to the backend, where preferences are processed, and recommendations are generated.
- **Displaying Recommendations**: The recommendations are fetched as JSON responses and dynamically rendered on the user interface using JavaScript.

This seamless interaction ensures that users receive real-time responses to their queries and enjoy a smooth experience.

The combination of HTML, CSS, and JavaScript ensures that the frontend is visually appealing, responsive, and highly interactive, providing users with an efficient way to explore movie recommendations. The integration with the backend API completes the loop, enabling dynamic communication and a rich user experience.

Challenges and Difficulties

7.1 Technical Challenges

1. **Integration of the GNN Model**:
   One of the primary technical challenges was the integration of the **Graph Neural Network (GNN)** model. Initially, the model was only processing the nodes from the Neo4j database and not considering the relationships between them. This limited the model's ability to capture the full context of the data. Relationships between nodes are essential in graph-based recommendation systems, and the model needed to learn from the connections. To resolve this issue, we modified the model to incorporate both nodes and relationships, ensuring that the GNN could effectively learn from the entire graph structure.

2. **Frontend Development Constraints**:
   A key technical difficulty was the inability to use **Node.js** for the frontend. We had to rely solely on **HTML** and **CSS** for building the user interface. This limited our ability to implement more dynamic, interactive features that Node.js could have provided. Despite these constraints, we developed a responsive and functional frontend, but the absence of Node.js required creative solutions to ensure smooth integration with the backend.

7.2 System Design Challenges

1. **Handling Complex Graph Data**:
   One of the major system design challenges was ensuring that the **Neo4j graph database** was structured in a way that allowed efficient querying and processing of complex relationships between users, movies, and other entities. Designing the database schema to accurately represent the interconnected nature of the data while ensuring efficient retrieval and integration with the GNN model was a complex task.

2. **Scalability and Performance**:
   As the system involved processing large amounts of data, another system design challenge was ensuring the solution was scalable and could handle increasing amounts of data without sacrificing performance. The combination of graph processing and AI model inference required careful consideration of how to manage and optimize data flow, particularly in terms of how data was passed between the backend and frontend.

These challenges were crucial learning experiences, emphasizing the importance of both technical expertise and system-level thinking in overcoming obstacles during

development. Despite these issues, we were able to deliver a functional and efficient movie recommendation system.

Proposed Solutions

8.1 Addressing Technical Challenges

To overcome the technical challenges we faced, we took a few key steps:

1. **GNN Model Integration**:
   The issue with the GNN model only processing nodes and not relationships was resolved by modifying the model to take both nodes and relationships into account. We updated the data flow so that the model could fully leverage the relational data, allowing it to capture the complex patterns between users, movies, and genres. This change enabled the model to generate more accurate and context-aware recommendations.

2. **Frontend Development Constraints**:
   Since we couldn't use **Node.js** for the frontend, we worked with **HTML** and **CSS**, which limited some dynamic interactions. To still ensure a smooth experience, we focused on using **JavaScript** for fetching data from the backend and dynamically updating the page without reloading. This approach allowed us to add a bit of interactivity, even with the constraints we had.

8.2 Enhancing System Performance

1. **Optimizing Data Handling**:
   For handling large amounts of data from Neo4j, we focused on optimizing the database queries. By indexing key data points and making sure the relationships between nodes were efficiently structured, we improved the speed and accuracy of the queries. This was important to ensure that the system could handle increasing amounts of data without slowdowns.

2. **Scalability Considerations**:
   As the system needed to scale, we considered how to efficiently manage the flow of data between the frontend, backend, and the GNN model. We used caching mechanisms for frequently accessed data to reduce processing time. Additionally, we ensured that the GNN model could process data in batches, improving overall performance and responsiveness.

These solutions helped us tackle the technical challenges while also ensuring that

the system could scale and perform efficiently as more users interacted with the recommendation engine.

Conclusion

9.1 Summary of Work Done

In this project, we developed a movie recommendation system that uses advanced technologies like **Neo4j** (for graph data storage) and a **Graph Neural Network (GNN)** model to deliver personalized recommendations. The system was designed to process complex relationships between users, movies, and other entities, leveraging graph data to improve the accuracy of recommendations. We faced several challenges along the way, such as the integration of the GNN model and frontend constraints, but through careful problem-solving, we were able to create a fully functional system. The frontend was designed using **HTML**, **CSS**, and **JavaScript**, providing a responsive and interactive user interface, while the backend was built using **Flask** to connect the frontend with the GNN model and Neo4j database.

9.2 Future Enhancements

Looking ahead, there are several areas for potential enhancement:

1. **Improved GNN Model**:
   The current GNN model could be fine-tuned further, potentially incorporating more advanced techniques like attention mechanisms or graph convolution layers to improve recommendation quality.

2. **Real-time Updates**:
   Implementing real-time updates based on user interactions and preferences could allow the system to adapt more quickly to changing tastes. This could involve incorporating **user feedback** into the recommendation model in real-time.

3. **Expanded Data Sources**:
   Adding more data sources, such as user reviews, ratings, or social media activity, could enrich the recommendation system. This would help create a more personalized and accurate prediction model that factors in additional user behavior.

4. **Scalability**:

As the number of users and data increases, the system can be optimized for better scalability. This could involve moving to more powerful cloud services or improving database and model handling to support higher traffic and data volume.

These enhancements would make the recommendation system even more powerful and adaptable, providing users with even more accurate and relevant movie suggestions.