# Patient Similarity Matching

Sahana Hariharan, Meghana Nair, Ryan Rajesh, Sreehari Sreedev

**Abstract** This project aims to enhance diagnostic efficiency and productivity in healthcare through the use of vector embeddings and AI. By integrating OpenAI and Node2Vec with a Neo4j graph database, the system enables precise similarity matching among patient records. The project addresses common challenges faced by doctors, such as recalling patients with similar symptoms and characteristics, by providing a tool that generates vector embeddings for medical data. The frontend allows users to input symptoms and receive a list of patients with similar profiles, including demographic details. The backend employs Cypher queries to preprocess data and conduct similarity searches, while Node2Vec is utilized to create and store vector embeddings based on patient attributes. Although initial results with OpenAI's embeddings show promising accuracy with similarity scores exceeding 0.9, further testing with larger datasets is needed. The Node2Vec approach, though insightful, faced limitations due to its dependency on pre-existing node relationships. This tool not only facilitates more effective treatment planning but also strives to reduce biases in healthcare by incorporating diverse demographic data, ultimately contributing to more equitable and personalized patient care.

Sahana Hariharan
University of Illinois at Urbana-Champaign, e-mail: sahanahariharan@gmail.com

Meghana Nair
Arizona State University, e-mail: mknair@asu.edu

Ryan Rajesh
Arizona State University, e-mail: ryan.rajesh@gmail.com

Sreehari Sreedev
Arizona State University, e-mail: me@ssree.dev

## 1 Introduction

The Patient Similarity Matching from medical reports project utilizes multimodal AI in order to be able to search from a wide range of patient records given certain filters. As medical applications expand, unimodal tasks are shifting which all around builds the basis for multimodal AI.

To start, the project is centered around efficiency and productivity, specifically for doctors. Oftentimes, doctors spend too much time diagnosing patients because they do not have the patient's symptoms off the top of their head; additionally, they may not remember patients that have had similar symptoms and what condition they had. So, the medical tool built during the project uses openAI in order to save time for doctors because when using the website, they are able to input a variety of symptoms. After inserting the symptoms, the backend data from the tool will generate patients who have had similar symptoms and their general characteristics (age, sex, height, weight, etc). So, once the original patient's file is pulled up, the doctor can see the similarities and differences of what conditions other patients had and they can take a course of action accordingly.

The website we built gives insight into demographic differences between patients and how a doctor can treat patients differently based on demographics. Since the tool has demographic factors built in, a hospital will have records for people of different backgrounds and there will be a large quantity of data to help doctors find similarities. Overall, the tool is a great resource for doctors to save time and even if there is an error with the AI, the doctor will be able to fact check the data.

## 2 Related Work and Background

In the field of medical research, the ability to compare and match patient data across different records is essential in improving diagnostics and treatment outcomes. The ability to extract meaningful insights from large, diverse datasets has become increasingly important as the volume of medical records and research continues to grow. The following section provides an overview of the key approaches and studies that have shaped our understanding and application of patient similarity matching in healthcare.

One such approach is explored in the paper "Use LLMs to Turn CSVs into Knowledge Graphs: A Case in Healthcare" [5] by Rubens Zimbres, which discusses the application of Neo4j Runway, an open-source Python library, to convert CSV data into knowledge graphs with the help of OpenAI's language models. The paper highlights the process of using LLMs to automatically identify nodes and relationships within medical data, creating a structured knowledge graph that can be queried using Cypher without requiring extensive manual coding. This approach is closely related to our project, as it illustrates the practical use of LLMs in transforming raw healthcare data into a more organized format that can be used for advanced analytics and decision support. By leveraging similar techniques, our project aims to develop a system that enhances patient similarity matching by converting medical records into knowledge graphs, improving the efficiency and accuracy of healthcare delivery.

Another related project is demonstrated in the article “A Gen AI-Powered Song Finder in Four Lines of Code” [1] by Christoffer Bergman. This article, published in the Neo4j Developer Blog, showcases the use of Neo4j’s built-in Generative AI capabilities to create a tool that finds songs based on a synopsis of their content. By using vector embeddings to represent the meaning of lyrics, the system can search for songs that match a given description, even when exact keywords are not present. While this project is unrelated to healthcare, the techniques used are comparative to those used in patient similarity matching, where the goal is to find records that are contextually similar, rather than just textually identical. The ability to perform such nuanced searches within a medical database could significantly improve the identification of relevant patient records, leading to more personalized and effective treatment plans.

In addition to the methodologies mentioned, another contribution to this field is presented in the article “EchoCLIP: A Method for Self-Supervised Audio-Language Representation Learning,” [2] by Abhinav Kulkarni. This paper introduces EchoCLIP, a model designed to align audio and language representations through self-supervised learning, leveraging CLIP-based architecture. Although the primary focus of EchoCLIP is on audio data, its underlying principles can be adapted to the healthcare domain, where aligning different modalities of data, such as medical reports and corresponding audio recordings or annotations, can enhance the accuracy and relevance of patient similarity matching. The use of self-supervised learning in EchoCLIP showcases the potential for creating representations that can be applied to diverse datasets, enhancing the tools available for patient similarity matching in healthcare.

These approaches highlight the potential of innovative technologies to improve matching and analyzing patient data. By combining ideas from these articles, our project seeks to make patient similarity matching more effective and practical. The integration of these methods aims to enable more precise searches within medical records, helping doctors develop more personalized and effective treatment plans.

## 3 Data and Methodologies

Please include a brief intro for your process of gathering data, which methods you used, and/or why you chose these datasets and methods.

### *3.1 Data Collection*

For this project, we used the ”Disease Symptoms and Patient Profile Dataset” [4] from Kaggle, which includes detailed information on patient profiles and symptoms for over 100 diseases. This dataset provided a rich source of data, capturing the relationships between various symptoms—like fever, cough, fatigue, and difficulty breathing—and patient demographics, including age, gender, blood pressure, and cholesterol levels.

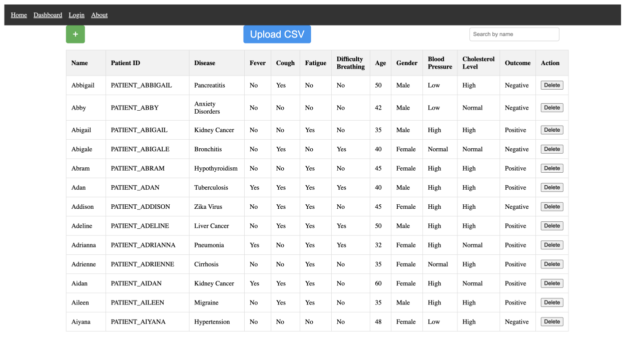
The dataset was chosen because of its comprehensive nature, allowing for an in-depth analysis of patient similarities based on their profiles and reported symptoms. It was used to create the vector embeddings for both the OpenAI and Node2Vec approaches to more effectively determine differences between the methods.

## 3.2 *Methods*

The patient similarity tool consists of a frontend, backend, and graph database. The frontend allows medical professionals to interact with and update patient records, while the backend and database guard access to these sensitive records. The architecture is meant to emphasize security by only returning the minimum amount of data required and making sure different people only have access to the bare minimum data that they need to do their job.

### 3.2.1 Frontend

The frontend is written in modern HTML, CSS, and Javascript using React.js and next.js to simplify page rendering and routing. Javascript ‘fetch()‘ is used to communicate with the backend, which exposes a RESTful API. The data exchanged from the backend is stored with the help of React state variables, which greatly simplifies rendering and interactions.



| Name | Patient ID | Disease | Fever | Cough | Fatigue | Difficulty Breathing | Age | Gender | Blood Pressure | Cholesterol Level | Outcome | Action |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Abbigail | PATIENT_ABBIGAIL | Pancreatitis | No | Yes | No | No | 50 | Male | Low | High | Negative | Delete |
| Abby | PATIENT_ABBY | Anxiety Disorders | No | No | No | No | 42 | Male | Low | Normal | Negative | Delete |
| Abigail | PATIENT_ABIGAIL | Kidney Cancer | No | No | Yes | No | 35 | Male | High | High | Positive | Delete |
| Abigale | PATIENT_ABIGALE | Bronchitis | No | Yes | No | Yes | 40 | Female | Normal | Normal | Negative | Delete |
| Abram | PATIENT_ABRAM | Hypothyroidism | No | No | Yes | No | 45 | Female | High | High | Positive | Delete |
| Adan | PATIENT_ADAN | Tuberculosis | Yes | Yes | Yes | Yes | 40 | Male | High | High | Positive | Delete |
| Addison | PATIENT_ADDISON | Zika Virus | No | Yes | Yes | No | 45 | Female | High | High | Negative | Delete |
| Adeline | PATIENT_ADELINE | Liver Cancer | No | Yes | Yes | Yes | 50 | Male | High | High | Positive | Delete |
| Adrianna | PATIENT_ADRIANNA | Pneumonia | Yes | No | Yes | Yes | 32 | Female | High | Normal | Positive | Delete |
| Adrienne | PATIENT_ADRIENNE | Cirrhosis | No | No | Yes | No | 35 | Female | Normal | High | Positive | Delete |
| Aidan | PATIENT_AIDAN | Kidney Cancer | Yes | Yes | Yes | No | 60 | Female | High | Normal | Positive | Delete |
| Aileen | PATIENT_AILEEN | Migraine | No | Yes | Yes | No | 35 | Male | High | High | Positive | Delete |
| Aiyana | PATIENT_AIYANA | Hypertension | No | No | No | No | 48 | Female | Low | High | Negative | Delete |

**Fig. 1** Here is an image of the frontend of our project. It offers utilities to search for, add, and delete patients.

### 3.2.2 Backend

The backend uses the Flask python framework, along with nginx acting as a reverse proxy and more capable HTTP server than the default WSGI server. The backend's main task is to gather raw data from the database and format it in a way that is easier to use in the frontend. This preprocessing is done by specially formed Cypher queries, which allow for data processing to happen on the database itself, before being sent over the wire to the backend and ultimately the frontend. In addition, basic access control is supported, to allow for information security.

An additionally important task the backend is entrusted with is enabling the similarity search. This is done at two stages: the creation of the data and for the actual similarity lookups. When a new patient is inserted into the project, the data must be transformed into a vector space to allow efficient searches. The backend performs this task by calling into the code that generates the vector embeddings and storing this data properly in the database. When the frontend requests a similarity search, the backend performs a cosine search on all the vector embeddings with the help of a Cypher query. Since this Cypher query is performed on the database, minimal data is transferred between the database and backend to support this request.

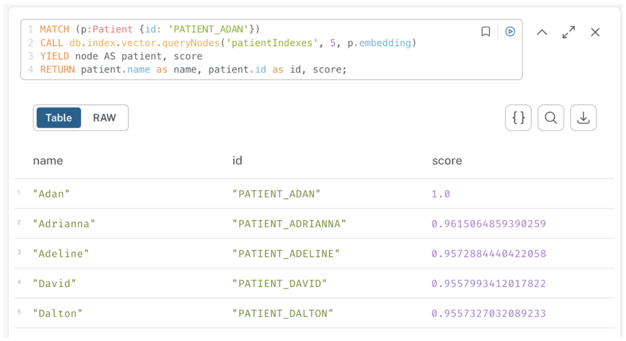
The database is an instance of Neo4j, where we store all required nodes and relationships. Vector embeddings are stored as a node property in each node, and are generated based on every other property of the node and its associated value.

For the vector embeddings using Node2Vec, we implemented the Node2Vec algorithm referrencing a prior project by Grover, Leskovec, and Kocijan in 2016 [3], which we found on GitHub. Initially, we considered implementing Node2Vec directly as a Cypher query within our Neo4j database. However, this approach required pre-existing relationships between nodes, which was not feasible for our dataset. Instead, we opted to use the Python implementation, where we manually created the relationships and simulated random walks on the graph to generate vector embeddings that capture the relationships between patients. These embeddings were then stored in the database to support similarity searches.

Given the setup, our goal was to measure the efficacy of different vector embedding methods, namely OpenAI's 'text-embedding-ada-002' and Node2Vec. OpenAI was fully integrated into the frontend and backend, while Node2Vec was only able to operate independently.

## 4 Results

Based on our success with OpenAI, we were able to perform efficient and accurate similarity searches on the dataset. Since vectors only need to be generated for changed data rather than the entire database for every change, we noticed that it scaled well too. By clicking on one patient and requesting similar patients, we can get a list of those patients, ranked by similarity score. The first result was always the original patient, because the patient is indeed most similar to itself. However, the results were often followed by results with scores greater than 0.9. This seems promising, however more testing must be done with more data to rigorously understand the efficacy. This is shown in Figure 2.

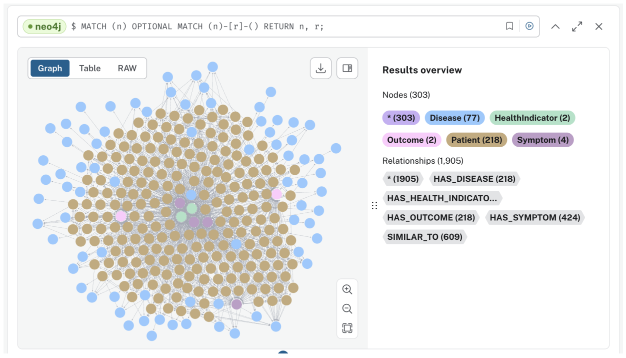


```
MATCH (p:Patient {id: 'PATIENT_ADAN'})
CALL db.index.vector.queryNodes('patientIndexes', 5, p.embedding)
YIELD node AS patient, score
RETURN patient.name as name, patient.id as id, score;
```

| | name | id | score |
|---|---|---|---|
| 1 | "Adan" | "PATIENT_ADAN" | 1.0 |
| 2 | "Adrianna" | "PATIENT_ADRIANNA" | 0.9615064859390259 |
| 3 | "Adeline" | "PATIENT_ADELINE" | 0.9572884440422058 |
| 4 | "David" | "PATIENT_DAVID" | 0.9557993412017822 |
| 5 | "Dalton" | "PATIENT_DALTON" | 0.9557327032089233 |

**Fig. 2** This is an image of the OpenAI implementation of the patient similarity search within Neo4J.

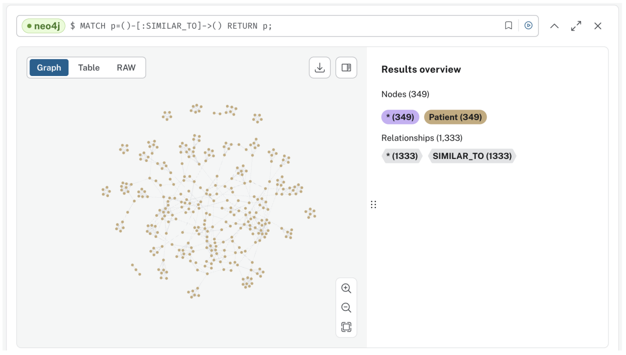
On the other hand, Node2Vec requires relationships to be created between all understood attributes in order to embed them and perform similarity searches. To do this, manual if statements are created to link nodes if they have some kind of text similarity. The results of the derived knowledge graph are shown in Figure 3.



```
neo4j $ MATCH (n) OPTIONAL MATCH (n)-[r]-() RETURN n, r;
```

**Fig. 3** Here is an image of the Node2Vec implementation of the patients in the database. Relationships are manually created between the patients and their characteristics.

Pictured in Figure 4 is the precomputed graph of patients' similarity to each other. The requested degree of similarity was 5, so there should be 1745 'SIMILAR_TO' relationships in total. However, we can clearly see that only 1333 relationships were created, which means that all patients do not have the required degrees of similarity. This can be attributed to deficiencies in the relationships generated by the manual if statements.



```
neo4j $ MATCH p=()-[:SIMILAR_TO]->() RETURN p;
```

Results overview

Nodes (349)

* (349) Patient (349)

Relationships (1,333)

* (1333) SIMILAR_TO (1333)

**Fig. 4** Node2Vec implementation showcasing the `SIMILAR_TO` relationships between patients.

## 5 Conclusion

### *5.1 Applications*

The Patient Similarity Matching tool we developed has significant potential for improving the way doctors create treatment plans for their patients. By utilizing detailed patient data, including symptoms and demographic factors such as age, gender, and ethnicity, the tool allows doctors to compare a patient's demographic and current condition with historical cases of patients who had similar profiles. This approach helps in identifying the most effective treatment plans based on real-world outcomes, tailored to the specific needs of each patient.

One of the critical benefits of this tool is its ability to address and reduce biases in healthcare, particularly those centered around the predominance of white male subjects in medical research. Traditionally, many medical treatments have been developed based on data that is not fully representative of the diverse patient populations that exist today. By incorporating a broad spectrum of demographic data, the Patient Similarity Matching tool can help doctors make more informed decisions that take into account the unique characteristics of patients from different backgrounds. This leads to more equitable healthcare outcomes and ensures that all patients receive the most appropriate care for their needs.

In addition, the tool's ability to quickly analyze large volumes of data and provide actionable insights can help doctors save time and reduce the cognitive load involved in diagnosing and treating complex cases. By presenting doctors with a list of similar cases and their outcomes, the tool serves as a valuable decision-support system, enhancing the overall efficiency of the healthcare process.

This tool not only aids in developing personalized treatment plans but also plays a crucial role in making healthcare more inclusive and equitable. Its implementation can lead to better patient outcomes and a more just healthcare system.

### *5.2 Concerns and Next Steps*

From the beginning of the project one of our biggest concerns when developing the product were privacy concerns. With the massive quantity of private information that the product will be storing there is a genuine concern about unauthorized users accessing our database to use for their own purposes. While a basic password system could be implemented to meet a baseline of security we also came up with a few more steps in order to make sure that this sensitive data would only be accessed by the right hands. A few suggestions we have is to upgrade the password security to two factor authentication with the second layer of security being routed to a secure device that only the medical staff will have access to. Another suggestion could be to have a system that keeps track of user logins and the data that each user has accessed. This way if a medical practitioner's account is used by a malicious party to gain access to our database the compromised account can be identified.

Another technical issue that we considered was with the constraints of the database itself. As it stands the database only stores a very limited amount of symptoms in each patient node, which lends to a very significant possibility that the tool could match two patients with similar symptoms but with different diseases. One method we thought could help with this is by increasing the depth of information that each node carries. If the nodes could store more symptoms it could in turn lend to more accurate feedback and help doctors make more accurate diagnoses.

While that would help with that specific problem, another more abstract concern we had was that the medical professional using the tool could just end up just diagnosing the patient just based on the information the tool gives them. With that specific problem on our end the best way of going about addressing that is to make it absolutely clear in our documentation that this tool is not meant as a way to instantly diagnose patients. While admittedly there isn't much our team can do to address this issue short of informing the user, it's still something worth acknowledging and discussing.

As our team was working on implementing Node2Vec we noted that since it requires relationships to exist between data it would only be practical if we couldn't rely on LLM models. Since we were using OpenAI we weren't able to properly integrate Node2Vec into the website. Based on that we came to the understanding that the best case usage for Node2Vec would be on a project where using an LLM would be either impossible or impractical. If the designer of the project knew that there were certain points of data that weighed more than the rest of the data set that they could then manually adjust, then Node2Vec could give more reliable and accurate results than an LLM, but generally speaking it will be more practical to use LLMs.

**Acknowledgements** A special thanks to our mentor, Kiran Karunakaran, for his guidance on the project and providing us with the idea.

## References

1. Christoffer Bergman. A gen ai-powered song finder in four lines of code. *Neo4j Developer Blog*, February 2024. https://neo4j.com/blog.
2. Matthew Christensen, Milos Vukadinovic, Neal Yuan, and David Ouyang. Vision–language foundation model for echocardiogram interpretation. *Nature Medicine*, 30:1481–1488, April 2024.
3. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.
4. Laksika Tharmalingam. Disease symptoms and patient profile dataset. https://www.kaggle.com/datasets/uom190346a/disease-symptoms-and-patient-profile-dataset, 2023. Updated 7 months ago.
5. Rubens Zimbres. Use llms to turn csvs into knowledge graphs: A case in healthcare. *Medium*, June 2024. https://medium.com.