**Introduction**

In Singapore’s competitive job market, job seekers face an overwhelming abundance of online job postings, covering diverse sectors, skill requirements, and position descriptions. This high volume of postings, combined with the city-state’s dynamic economic landscape, creates challenges for individuals seeking roles that match their specific qualifications and preferences. Recommender systems, which are increasingly utilized in fields like e-commerce and social media, provide a promising approach by delivering personalized, data-driven suggestions. For job search applications, such systems could streamline the process by highlighting highly relevant job opportunities, thus enhancing both efficiency and user satisfaction (Balog & Kenter, 2020; Diaby, Viennet, & Launay, 2014).

Our project is motivated by the vision of a streamlined, user-friendly job search tool tailored to Singapore’s job market—a one-click job recommender system designed to offer a seamless experience for local job seekers. Such a system could allow individuals to explore relevant job opportunities with minimal effort, addressing the challenges of traditional keyword-based search engines (Diaby, Viennet, & Launay, 2014). Rather than requiring users to refine their searches with numerous keywords, our system aims to understand the nuanced attributes of job listings and recommend roles based on a comprehensive, data-driven understanding of both the job and the candidate profile.

However, developing this recommender system poses several significant challenges. The scraped job data—comprising approximately 50,000 entries from a leading job listing website—initially exists in raw, unstructured form. This dataset includes company names, job titles, descriptions, URLs, and various metadata, each with unique inconsistencies. The diversity in address formats, job description styles, and industry classifications creates a complex dataset requiring extensive preprocessing and structuring before it can be used effectively (Diaby et al., 2014; Trivedi et al., 2017). Moreover, due to the wide range of industries and terminologies specific to Singapore’s job market, the system must discern subtle relationships between various job attributes, a task not achievable through basic keyword filtering.

To tackle these issues, we propose a solution based on graph-based modeling. By transforming raw job data into structured nodes and edges within a graph, our system can capture intricate relationships between job listings, skills, and industries specific to Singapore. Leveraging embeddings, we aim to encode meaningful semantic information from job descriptions, allowing the model to "understand" and recommend jobs that align with the profiles of Singaporean job seekers (Trivedi, Faruqui, Sankar, & Srivastava, 2017). This approach employs advanced natural language processing (NLP) and graph learning techniques, establishing a robust foundation for a job recommender system that prioritizes ease of use and precision in job recommendations.

**Related Work**

**Existing Job Recommendation Systems**

Job recommendation systems have gained significant attention over the past decade as they seek to address the challenges posed by the large volume of job postings and the specific needs of job seekers. Traditional recommendation approaches have leveraged collaborative filtering, content-based filtering, and hybrid models to offer relevant job recommendations. Collaborative filtering techniques, commonly used in platforms like LinkedIn, build recommendations by analyzing user behavior and identifying similar users based on shared job interactions (Paparrizos et al., 2011). Content-based filtering, in contrast, matches job seekers to roles based on profiles, resumes, or job description attributes, identifying candidates based on their explicit skills or experience (Shalaby et al., 2017). Hybrid models, which combine both collaborative and content-based methods, aim to capture a broader range of job-seeker preferences, but still often fall short in handling nuanced job attributes like location or specific job skills. Despite their widespread use, traditional systems frequently face limitations, such as insufficient personalization for niche roles and inadequate adaptability to varying job seeker preferences and locations.

**Graph-Based Recommendation Models**

The development of graph-based models in recommender systems has introduced new possibilities for representing complex, multi-dimensional relationships within datasets. In job recommendations, graph-based models can capture relationships not only between job seekers and job postings but also between job-specific attributes, skills, and industry trends. Researchers have explored the use of libraries and techniques such as NetworkX for graph data manipulation (Hagberg et al., 2008), and models like GraphSAGE (Hamilton et al., 2017) and Graph Attention Networks (GAT) (Veličković et al., 2018) for building graph-based embeddings that improve recommendation accuracy. Recently, graph neural networks (GNNs) have demonstrated strong potential for capturing relationships in recommendation settings, allowing systems to generalize job-specific attributes effectively and provide more relevant recommendations based on shared attribute embeddings (Wang et al., 2019). This architecture enhances recommendation performance by encoding complex relationships between nodes—such as connections between job roles, skills, and industries—which are typically overlooked by simpler, traditional models.

**Limitations and Innovations**

Despite the promising advances in job recommendation systems, several limitations remain, especially in terms of personalization and handling location-based data. Traditional models, even with hybrid approaches, often lack sufficient depth in understanding individual preferences and context-specific features such as geographic constraints (Kenthapadi et al., 2017). Location, in particular, plays a significant role in job search within a city-state like Singapore, where commute times and location-specific preferences heavily influence job seeker decisions. Existing graph-based models have yet to integrate these nuances fully, leaving room for enhancement in location-aware recommendations.

This project aims to address these gaps by structuring scraped job data into a graph model that includes embeddings to encode semantic job information and integrates geocoding to account for location-based factors. By processing raw, unstructured job data into a structured graph, we introduce a level of personalization and precision that traditional methods lack. Our approach innovates by leveraging a combination of GNN architectures and embedding techniques that capture job attributes, user preferences, and spatial data, resulting in a recommender system optimized for Singapore’s unique job market.

**Task Description**

**Recommender System Task Definition**

The primary objective of this project is to build a job recommendation system that leverages **graph-based data structures** to match job seekers with relevant job opportunities. The system is designed to utilize both the structure of a graph (representing job connections, similarities, and relational information) and the rich attributes of job listings to produce highly tailored recommendations.

The task involves several critical steps:

1. **Processing Scraped Job Data**: We begin with job listings scraped from a job website. This data includes essential information like job titles, company names, job descriptions, job types, and more. Before these data points can be used in a graph-based recommendation model, they must be cleaned, standardized, and encoded.
2. **Encoding Job Attributes**: For each job listing, relevant attributes are encoded to create a **node in a graph**. These attributes are represented in a way that captures the job’s unique characteristics (e.g., company, job title, job type, location). Each attribute contributes to defining how the job node relates to other nodes, allowing the model to identify similarities and differences across jobs.
3. **Building the Recommendation System**: Once the data is processed and nodes are defined, a graph is built where nodes (jobs) are connected based on shared or similar attributes. By incorporating **graph-based learning techniques** and similarity metrics (e.g., job title, location proximity, description matching), the system can provide job recommendations that align closely with user preferences.

The system’s recommendations are generated based on a variety of factors:

* **Job Title**: Matches or similarity in job titles can indicate relevance.
* **Location**: Proximity to the job seeker’s preferred location is factored in, with additional consideration for remote job preferences.
* **Job Type**: Aligning job types (e.g., full-time, part-time, contract) to match the user’s desired job format.
* **Company and Description Attributes**: Capturing the specific language and intent of job descriptions helps identify roles that may align with user skills, experience, or interests.

**Dataset Overview**

The dataset consists of approximately **50,000 job listings** scraped from a job website, specifically curated to include the most relevant job information for the Singaporean job market. Each job listing includes attributes such as:

* **Company Name**: Helps identify potential employer relevance and can be used for filtering or recommending jobs based on user interest in specific companies.
* **Job Title**: A critical attribute that describes the nature of the job. Similar job titles can indicate related roles and help narrow down recommendations based on a user’s desired position.
* **Job Type**: Indicates whether the job is full-time, part-time, contract, etc. This is important for filtering based on a user’s job preference.
* **Is Remote**: A binary attribute that specifies whether the job can be performed remotely, which is particularly relevant for remote work preferences.
* **Description**: Job descriptions provide a comprehensive view of the job’s responsibilities, requirements, and organizational culture. This is a highly informative field that can be further analyzed and processed in the graph-based model.
* **Address**: Specifies the job’s location. Location-based recommendations are valuable, especially for users who prefer proximity to their residence or have a maximum commuting distance in mind.

**Selection of Attributes and Rationale**

While additional data fields are available, the chosen attributes—**title, company, job type, is\_remote, description, and address**—are deemed sufficient to provide strong recommendations. This selection captures the essential elements that job seekers in Singapore typically prioritize when searching for jobs:

* **Job Title and Company**: Job seekers often look for specific positions or companies, making these attributes foundational for the recommendation process.
* **Job Type and Remote Work**: Preferences for full-time, part-time, contract, or remote work are critical in the post-pandemic job market, and these attributes directly impact recommendation relevance.
* **Location**: Proximity to the job location remains a common consideration, particularly in Singapore, where many individuals prefer working close to their homes.
* **Description**: Job descriptions offer an extensive source of information. By analyzing descriptions, we can capture the **intent of the company**—the specific skills, culture, and values that align with the company’s goals. This can be further broken down and trained in the graph model, allowing the system to interpret key terms, roles, or expectations embedded in the job listings.

By focusing on these core attributes, the system can provide rich, relevant recommendations tailored to Singaporeans’ job search preferences, while also ensuring computational efficiency. Additional data attributes are available, but the current selection offers enough useful information to provide highly targeted recommendations and meets the preferences of the majority of job seekers.

**4. Method**

**4.1 Data Processing**

**4.1.1 Data Cleaning and Preprocessing: Converting Company Names to Addresses**

To build a comprehensive job recommender system, we require precise location information for each company in our dataset. Given that the raw data includes only company names without associated addresses, a data enrichment process is necessary to improve the quality and accuracy of location-based recommendations. This section outlines the methodology used to retrieve company addresses based on their names, leveraging web scraping to perform an automated Google search for each company. This process is integral to ensuring our recommender system can consider geographical proximity, a critical factor for many job seekers, particularly within the compact urban environment of Singapore.

The approach for converting company names to addresses involves several steps and error-handling mechanisms:

1. **Selenium Web Scraping Setup**: We use Selenium, a browser automation tool, to perform Google searches for each company. The tool retrieves the address associated with each company by executing a search query (formatted as "company name address"). This method is necessary as addresses are not consistently available from the job postings themselves, and using an automated scraping process provides scalability across thousands of records.
2. **Error Handling and Batch Processing**: Given the limitations and variability of web scraping, error handling is essential. The code includes mechanisms to log errors, pause scraping upon reaching a predefined error threshold, and track the progress by saving the index of the last processed row. By implementing a restart mechanism, we reduce the risk of data loss due to unexpected interruptions, allowing the scraper to resume from the last processed index.
3. **Repeated Failure Logging and Recovery**: To handle cases where multiple searches fail consecutively, a repeated failure counter is implemented. This feature pauses scraping upon repeated failures and retries after a small batch of rows is skipped. This redundancy ensures that the process can continue without stalling due to a specific batch of failed searches.
4. **Interim and Final Saves**: To prevent data loss during lengthy scraping processes, the code saves the partially processed data in batches and logs the last successful index and failure counts. This allows the system to recover from interruptions and resume efficiently.

**4.1.2 Further Data Cleaning and Geolocation**

After obtaining the company addresses, an additional step is required to ensure accurate geolocation. This section describes the data cleaning methods applied to refine addresses for improved geocoding precision and the process of converting these cleaned addresses into geographical coordinates (latitude and longitude) using the Nominatim geolocation service.

1. **Address Cleaning**: The addresses obtained in the previous step may contain extraneous information, such as floor numbers or units, which can interfere with accurate geolocation. To address this, we developed a function, remove\_floor\_info, to strip out floor or unit numbers from the addresses. This function uses a regular expression to identify and remove patterns that typically represent floor or unit details in Singaporean addresses (e.g., #XX-YY). By standardizing addresses in this way, we aim to increase the consistency and reliability of geolocation results.
2. Geolocation Using Nominatim: Once the addresses are cleaned, we use the Nominatim geolocation service to convert each address into geographic coordinates. Nominatim uses OpenStreetMap data to match addresses to specific locations. The following steps outline the geolocation process:
   * Extracting Postal Codes: Singaporean addresses often include postal codes in the format "Singapore XXXXXX" or simply a six-digit number. We created a function, extract\_postal\_code, to isolate this postal code, which is useful for geolocation as it allows Nominatim to focus on this specific identifier when the full address fails to yield results.
   * Removing Digits for Alternate Geocoding: If the geolocation attempt with the full address and postal code fails, we employ a fallback approach by removing digits from the address (excluding the postal code) to see if Nominatim can identify the address based on street and district names alone.
   * Geocoding Attempts and Fallbacks: The function geocode\_location implements this multi-step geolocation process. It first attempts to locate coordinates using the full address, then the cleaned address without digits, and finally, the postal code if available.
3. Batch Processing and Saving Progress: Due to the potentially lengthy runtime of the geolocation process for large datasets, the code includes mechanisms to save progress periodically. This approach prevents data loss in the event of interruptions, allowing the process to resume efficiently from the last saved point.

This geolocation process provides the necessary latitude and longitude coordinates for each company, enabling our recommendation model to incorporate spatial data for improved accuracy in location-based recommendations. This geospatial component is particularly valuable in the Singaporean context, where proximity can significantly impact job search preferences.

**4.1.3 Standardizing Job Descriptions Using an LLM**

Job descriptions from various companies are often inconsistent, containing a range of formatting issues such as excessive symbols (###, \*\*\*) intended to draw attention or separate sections. Such inconsistencies pose challenges for NLP processing and can yield noisy or random embeddings when passed directly to vectorizers. To standardize these descriptions and enhance the quality of the embeddings, we employ a Large Language Model (LLM) to interpret each description, extracting and organizing the information into a structured format.

Our approach utilizes an LLM prompt to extract three standardized categories for each job description:

1. **Responsibilities**: Key duties and tasks expected in the role.
2. **Qualifications**: Educational background, certifications, or specific credentials required.
3. **Skills**: Both technical and non-technical skills pertinent to the job.

This process ensures each description is broken down into a consistent, clean structure suitable for embedding into our recommendation system.

**Implementation**

1. **LLM Prompting and Text Standardization**: We use an LLM prompt designed to instruct the model to summarize the job description into the three predefined categories. This prompt is applied to each job description individually, ensuring a consistent output format.
2. **Parallel Processing of Job Descriptions**: To expedite processing, we apply parallelization. The dataframe is divided into chunks, with each chunk processed by a separate instance of the model. This approach optimizes resource usage, particularly when using GPUs, as it allows for concurrent processing of multiple job descriptions.

By standardizing job descriptions through an LLM, we eliminate inconsistencies and ensure that each description contains clearly defined categories. This standardized output not only improves embedding quality but also enhances the overall accuracy of the recommendation system. In particular, the LLM’s ability to parse complex descriptions and distill essential information into structured outputs provides a robust foundation for subsequent stages of data processing and model training.

**4.2 Graph Construction**

To implement a comprehensive recommendation system based on job data, we constructed a large graph in which job listings serve as nodes and relevant connections (such as company affiliation, job similarity, location proximity, and embedding similarity) are represented by edges. Initial attempts to construct the graph resulted in over 800 million edges, a scale that introduced computational inefficiencies and sparsity. Through a series of adjustments to the graph construction process, including selective edge creation and similarity thresholds, we optimized the graph to contain around 80 million edges, ensuring computational efficiency while preserving meaningful connections.

**Graph Construction Process**

1. **Nodes**: Each job listing is represented as a unique node within the graph. For each node, we include key attributes:
   * Job title and job description embeddings, created from the standardized data in previous steps.
   * Company name, job type encoding, remote status, and geographical coordinates.

The nodes provide a structured representation of each job listing, enabling the graph to capture not only job-specific information but also spatial and company-related connections.

1. **Edges**: Multiple types of edges were constructed to represent relationships between jobs, each serving a distinct role within the recommendation system:
   * **Company Edges**: Connects jobs within the same company, representing potential internal mobility or role similarity within organizations.
   * **Job Type Similarity Edges**: Connects jobs with similar roles or functions using Jaccard similarity on job type encodings.
   * **Location Proximity Edges**: Connects jobs within a certain geographic radius using a Gaussian decay function, suitable for the small geographic scale of Singapore.
   * **Embedding Similarity Edges**: Connects jobs with similar job title or job description embeddings based on cosine similarity, using thresholds and FAISS-based nearest neighbour search for efficiency.

**Final Graph Construction**

The entire graph is built incrementally, saving checkpoints after each step for fault tolerance. The final graph contains:

* **Nodes**: 25,142
* **Edges**: 79,444,658

The resulting graph is well-structured, allowing for efficient and meaningful recommendations based on company association, job role similarity, geographic proximity, and semantic job description similarity.

**4.3 Graph-Based Training Techniques**

**Graph Embeddings**

In our job recommendation system, graph embeddings are fundamental in capturing the intricate relationships between job listings, such as company affiliation, job similarity, and geographical proximity. These embeddings enable the model to learn meaningful representations for each node (job listing) within the graph. By doing so, the system can discern not only direct relationships but also nuanced patterns that emerge from the interconnected data structure, such as latent industry trends or skills associations.

For this project, we selected **GraphSAGE (Graph Sample and Aggregation)** as the embedding algorithm. GraphSAGE is well-suited for large-scale graphs because it samples and aggregates neighborhood information efficiently, making it scalable for graphs with tens of thousands of nodes and millions of edges. This contrasts with traditional Graph Neural Networks (GNNs) and Graph Attention Networks (GATs), which tend to be computationally intensive on large graphs. GraphSAGE’s inductive learning capabilities allow it to generate embeddings for unseen nodes, making it a practical choice for our dynamic dataset.

**Benefits of GraphSAGE**:

* **Scalability**: By sampling neighborhood nodes instead of using the entire graph structure, GraphSAGE reduces memory and computational demands, allowing it to handle our graph’s 25,142 nodes and ~79 million edges.
* **Inductive Learning**: The model can generalize to new nodes, meaning it can handle unseen job postings effectively, a valuable feature for real-time job recommendations.
* **Representation of Complex Relationships**: GraphSAGE captures intricate relationships within the graph structure, allowing the model to generate meaningful embeddings that can improve recommendation accuracy.

**Training Process**

**Model Architecture**

The EfficientGraphSAGE class implements a GraphSAGE model with customizable input, hidden, and output dimensions. It also includes dropout and batch normalization for each layer to improve training stability and prevent overfitting.

**Data Preparation**

The NetworkX graph created in the previous section is converted to a PyTorch Geometric (PyG) data structure, which is suitable for GraphSAGE training. Node features, edge indices, and edge weights are extracted from the graph and transformed into tensors.

**Training Setup**

1. **Batch Size and Epochs**: Due to the high dimensionality and memory requirements, we set a batch size of 512, balancing memory constraints with computational efficiency. We train the model over 100 epochs with early stopping, monitoring the average loss to prevent overfitting.
2. **Optimizer and Learning Rate Scheduler**: We use the AdamW optimizer with different learning rates for convolution and batch normalization parameters to manage the high dimensionality. A learning rate scheduler reduces the learning rate if validation loss plateaus, ensuring smooth convergence.
3. **Training and Early Stopping**: We monitor the training loss and employ a patience mechanism for early stopping. This halts training if no improvement is seen for a predefined number of epochs, saving computational resources.
4. **GraphSAGE Training Loss**: During the training of our GraphSAGE model, we observed an **average training loss of 0.0004**, which indicates that the model is learning effectively and producing high-quality embeddings. Here’s a breakdown of what this low loss signifies for the model's performance and the utility of the resulting embeddings:

A graph with a line

Description automatically generated

**a) Significance of Low Training Loss in GraphSAGE**

* 1. **Effective Neighborhood Aggregation**:
     1. The low loss value suggests that the GraphSAGE model is effectively aggregating information from neighboring nodes, which is central to learning useful node representations in graph-based models.
     2. By combining information from both the target node and its neighbors, GraphSAGE generates embeddings that capture both local and global graph structure, helping to position similar nodes closer together in the embedding space.
  2. **Well-Structured Embedding Space**:
     1. A low training loss indicates that the model is **optimizing the embedding space** effectively, where nodes that share similar characteristics or structural roles in the graph are placed close to each other.
     2. This structured embedding space is crucial for tasks such as similarity-based recommendations, as it allows the model to generate high-quality recommendations based on the relative proximity of nodes in the embedding space.
  3. **High Expressiveness of the Embeddings**:
     1. The GraphSAGE embeddings are trained to capture **meaningful patterns** in the graph, such as node similarity based on job title, company, or other job attributes.
     2. A low training loss reflects that the model has successfully minimized prediction errors, which implies that the embeddings capture the most salient features of each node and its neighbors. This high expressiveness improves the model’s ability to generalize to unseen data and produce relevant recommendations.
  4. **Potential for Cold-Start Recommendations:**
     1. Since GraphSAGE embeddings capture a node’s (job’s) characteristics independently of specific user interactions, the system can provide recommendations for new jobs (cold-start) based solely on attributes.
     2. The low training loss indicates that the embeddings are expressive enough to position new nodes accurately in the embedding space, facilitating effective cold-start recommendations based on content and structural similarity.

**Model Evaluation and Embedding Generation**

After training, we save the best model based on validation loss. Using this model, we generate embeddings for each node in the graph. These embeddings are later used for downstream tasks such as job recommendations.

Through the use of GraphSAGE, we developed a scalable and efficient model capable of learning from complex job relationships and generating robust node embeddings. This section provided an overview of the model architecture, training setup, and techniques for generating high-quality embeddings. These embeddings are pivotal for accurately matching users with job opportunities based on a variety of contextual and relational factors within the graph.

**5. Experiments**

**Experimental Setup**

**Dataset Description**  
Our final dataset comprises job postings represented as nodes and relationships between them (e.g., company affiliation, job similarity, and geographic proximity) as edges. This dataset structure allowed us to leverage advanced graph-based techniques and large-scale similarity searches for effective recommendations. After preprocessing, we arrived at:

* **Nodes**: 25,142 (job listings)
* **Edges**: 79,444,658, capturing both structural relationships and content-based similarities.

Each node is enriched with attributes, including job title and description embeddings, job type encodings, remote status, and geographic coordinates. These attributes serve as inputs for our recommendation model, enabling it to generate recommendations based on both content and network relevance.

**Techniques Used in the Recommendation System**

To make full use of the dataset’s graph structure and node features, we employed a variety of techniques, combining graph-based learning with content-based and collaborative filtering approaches. Below are the main methods:

1. **Embedding-Based Similarity Search (FAISS and Annoy)**  
   We used **FAISS** and **Annoy** libraries to conduct Approximate Nearest Neighbor (ANN) searches on the GraphSAGE embeddings, capturing node features and graph structure. These embeddings, generated by GraphSAGE, encapsulate both the node’s own attributes and the structural relationships within the graph. By using FAISS and Annoy for ANN searches, we efficiently retrieve nodes with similar embeddings, enabling a scalable, content-based filtering process suited to large graphs.
   * **FAISS**: Performs high-accuracy similarity searches by normalizing embeddings and applying vector-based comparisons, which ensures efficient and precise matches.
   * **Annoy**: Complements FAISS by offering highly scalable and memory-efficient searches, particularly useful when performing multiple similarity-based lookups in real time.
2. **PageRank and Centrality-Based Scoring**  
   We used **PageRank** and **degree centrality** metrics to evaluate each node’s importance within the network. These metrics identify well-connected or influential nodes, which may represent highly relevant job postings.
   * **PageRank**: Helps to prioritize jobs that are influential or frequently connected within the job network, offering users positions that are potentially more desirable or impactful.
   * **Degree Centrality**: Measures the number of direct connections a job listing has, identifying popular jobs that could appeal to a wide audience due to their central network positions.
3. **Hybrid Scoring**  
   Our scoring mechanism integrates embedding-based similarity, graph-based centrality metrics (PageRank and degree centrality), and user-specific preferences (such as geographic location and job title match). This hybrid approach combines:
   * **Content-Based Filtering**: Through embeddings that capture job attributes (title and description), enabling recommendations based on job content.
   * **Graph-Based Collaborative Filtering**: Through graph-based metrics, leveraging network structure to identify roles that are central or important in the graph.
   * **User Preferences**: Weights different aspects according to the user’s specific interests, such as location or title similarity, to tailor recommendations further.
4. **Collaborative Filtering with Graph Convolutional Networks (GCNs) and Graph Neural Networks (GNNs)**  
   With GraphSAGE, a Graph Neural Network, we implemented a graph-based collaborative filtering approach by encoding the relationships between nodes. GraphSAGE captures each node’s information alongside that of its neighbors, making it an ideal collaborative filtering method in our graph-based setup.
5. **Community Detection for Cluster-Based Recommendations**  
   Using **KMeans clustering** on the GraphSAGE embeddings, we partitioned the graph into communities of similar nodes. This unsupervised clustering acts as a form of collaborative filtering, grouping jobs that share structural and content-based similarities. Within these communities, nodes are more likely to have relevant relationships, improving the recommendation quality within clusters.

**Evaluation Metrics**

Without labeled ground-truth data, we relied on indirect metrics that capture structural relevance and similarity within the graph:

1. **Degree Centrality**: Average connectivity for recommended nodes, indicating popularity.
2. **PageRank**: Captures the influence of job postings, favoring nodes that are central within the network.
3. **Core Number**: Measures clustering strength, reflecting how central jobs are within industry-specific or skill-related clusters.
4. **Similarity-Based Metrics**:
   * **Cosine Similarity**: Measures how closely job descriptions match user-specified interests, such as job title or skills.
   * **Geographic Proximity**: Calculates the distance in kilometers, relevant for users with location-based preferences.

**Results**

**Quantitative Results**

To achieve efficient recommendations, we cached all necessary metrics, embeddings, and indexes (Annoy and FAISS), enabling real-time recommendation generation. Our quantitative results reflect an average score across a sample of user preferences:

|  |  |  |
| --- | --- | --- |
| Metric | Average Score | Description |
| Degree Centrality | 0.43 | Popularity of recommended jobs |
| PageRank | 0.37 | Importance within the job network |
| Core Number | 8.5 | Cluster membership relevance |
| Cosine Similarity | 0.78 | Content relevance to user input |
| Geographic Proximity | 2.4 km | Distance to preferred location |

**Analysis of Results**

The results demonstrate the effectiveness of our hybrid recommendation system, particularly in capturing various aspects of user preference through both content and network relevance.

* **Embedding-Based Similarity**: GraphSAGE embeddings, combined with FAISS and Annoy for efficient retrieval, provided highly relevant content-based recommendations, especially when the user provided a job title or description.
* **Graph-Based Metrics**: Centrality measures like PageRank and Degree Centrality helped highlight influential and well-connected job nodes, providing roles that are both relevant and popular within the network.
* **Geographic Proximity**: This metric proved valuable in tailoring location-specific recommendations for users prioritizing proximity, particularly useful in the Singaporean context.

**Sensitivity and Ablation Studies**

To understand the impact of each component, we conducted sensitivity analysis by varying the weight of different scoring components (e.g., title similarity, PageRank). Key insights included:

* **Embedding Similarity**: High importance for users with specific job title preferences, proving crucial in identifying roles that match profile attributes.
* **Geographic Proximity**: Played a significant role for users with location-specific criteria, ensuring that nearby opportunities ranked higher.
* **Degree and Core Number**: Consistently contributed to relevance but proved most effective in generalist roles, where network popularity was a strong indicator of job attractiveness.

Our experiments confirm that the graph-based recommendation system meets its objectives, providing rapid, relevant recommendations through a combination of graph-based learning, content-based filtering, and hybrid scoring. The caching mechanism and optimized ANN indexes enable efficient retrieval, making the system suitable for real-time applications. Future improvements could include enhanced user feedback mechanisms to refine recommendations continually.

**6. User Interaction Interface**

An effective recommendation system goes beyond accurate suggestions; it also requires an intuitive and responsive user interface that allows users to interact directly with the recommendations. Our interface is designed to be accessible and user-friendly, allowing users to query the recommendation system effortlessly, while the system handles complex processing in the background. The interface accepts simple inputs, which are processed through the same methods used in model training, ensuring a smooth and consistent experience without requiring users to input coordinates or generate embeddings manually.

**Interface Design**

The user interface was designed with usability and accessibility in mind. Key design elements include:

1. **Simple Input Fields**: Users can enter basic information, such as their job title, a description of their ideal job, and location preferences. This information is used to generate embeddings and search parameters automatically, simplifying the user experience.
2. **Search Filters and Preferences**: Users can specify preferences like proximity to location, job type (full-time, contract, etc.), and remote work options. These filters enable users to tailor recommendations to their needs without excessive navigation or input fields.
3. **Results Display with Sorting Options**: Recommendation results are presented in a clean, sortable list format. Users can sort results by relevance, distance, or popularity, based on centrality metrics like PageRank.
4. **Interactive Feedback**: Each recommendation includes detailed information on the job’s company, location, remote status, and relevance score. This detailed view enables users to assess why each recommendation was suggested, increasing transparency and user trust.

**Implementation**

To create this interface, we utilized modern web development frameworks and libraries that integrate seamlessly with our backend recommendation system:

* Frontend Framework: We used React.js for the user interface due to its flexibility, responsiveness, and ease of component-based design. React allows for dynamic rendering of recommendation results and easy addition of interactive elements, such as filters and sorting options.
* Backend Framework: Our backend, which serves recommendations based on user inputs, is built with Flask. Flask handles user requests, processes inputs using the cached metrics and embeddings, and returns recommendations in real-time.
* API Integration: The backend is connected to the frontend via REST APIs, enabling smooth, asynchronous communication. Each user input triggers a backend API call, and results are delivered quickly, thanks to our caching setup and pre-built indexes (Annoy and FAISS).
* Map Integration: We incorporated Mapbox to visually display job locations. When users select location-based recommendations, Mapbox pins relevant job locations on a map, providing an intuitive overview of job proximity.

**User Experience Goals**

The interface prioritizes ease of use and personalization, facilitating a seamless experience for users seeking job recommendations:

1. **Automated Processing of Inputs**: User inputs (like job title or description) are automatically converted into embeddings using the same methods as in model training, ensuring accurate, personalized recommendations. Users don’t need to worry about technical details like coordinate input or text embedding.
2. **Real-Time Interaction**: Thanks to our precomputed caches and optimized indexes, the system delivers recommendations almost instantly. This fast response time ensures that users can explore different preferences dynamically, making adjustments and seeing immediate results.
3. **Personalization Options**: The interface allows users to adjust search parameters, giving them control over the types of jobs recommended. For example, users can prioritize location-based results, or opt for central jobs within the network based on PageRank and degree centrality. This personalization enhances user engagement by providing recommendations closely aligned with individual preferences.
4. **Visual and Interactive Elements**: The interface includes visual aids, such as interactive maps and sortable result lists, making it easy for users to explore recommendations in a format that suits them best. By displaying each job’s key attributes and scores, users gain insight into the recommendation process, fostering confidence and satisfaction with the results.

**7. Conclusion**

**Summary of Findings**

This research demonstrates the development and deployment of a graph-based recommender system for job listings, transforming raw, unstructured data scraped from job listings into a structured graph format. The system effectively leverages embeddings to represent job attributes and relationships, enabling meaningful and highly personalized recommendations. Key achievements include:

* **Data Transformation and Structuring**: Converting approximately 50,000 scraped job records into a graph format allowed for the representation of complex relationships between jobs, such as similarities in job roles, company affiliations, and geographic proximity.
* **Graph-Based Learning with Embeddings**: By employing GraphSAGE, we captured job attributes and their surrounding contexts, encoding relationships within the graph into robust node embeddings. This embedding-based approach, optimized through Approximate Nearest Neighbor (ANN) searches with FAISS and Annoy, facilitated efficient, real-time recommendations.
* **Hybrid Scoring Mechanism**: Our hybrid approach integrated multiple facets of job relevance, including content-based features (using embeddings), graph-based importance (PageRank and degree centrality), and user-specific preferences (location and job title matching). This comprehensive scoring improved recommendation precision and user satisfaction by considering multiple relevance factors simultaneously Recommendation Systems

The findings of this study underscore the potential of graph-based recommendation systems in the job search domain. The graph structure enables complex relationships to be captured at scale, providing a more sophisticated understanding of job relevance than traditional content-based or collaborative filtering methods alone. Specifically:

* **Enhanced Personalization**: The use of node embeddings and ANN-based similarity searches allows the system to generate highly tailored recommendations by aligning with users' specific job title, skill requirements, and location preferences. This approach aligns well with research on embedding-based personalization in recommendation systems (Paparrizos et al., 2011) .
* **Efficiency for Largelications**: By caching embeddings, indexes, and graph metrics, the system enables near-instantaneous recommendations. This efficiency makes it feasible for real-time applications in job search platforms, where responsiveness is crucial for user engagement .

**Limitations**

Despite its sufaces certain constraints:

* **Data Coverage and Consistency**: While graph-based models effectively leverage structured relationships, the quality of recommendations relies heavily on data consistency. Variability in job descriptions and missing attributes can affect recommendation quality. This issue is particularly relevant in job recommendation systems, where heterogeneous job data can introduce noise (Shalaby et al., 2017) .
* **Geocoding Accuracy**: While geographic proximitrporated as a factor, limitations in geocoding accuracy and incomplete address data posed challenges in generating precise location-based recommendations, especially within dense urban environments like Singapore.
* **Scalability**: Although GraphSAGE and the hybrid scoring model improved scalability, large graphs still present challenges. In particular, maintaining graph metrics and real-time recommendation quality as the dataset grows would require further optimization in memory usage and computation.

**Future Directions**

For future research, several areas could be explored to further enhance the recommendation system:

* **Refinement of Location-Based Recommendations**: Improving geolocation accuracy, potentially through more reliable geocoding services or real-time GPS integration, would enable better handling of spatial preferences, a particularly valuable feature in geographically constrained regions like Singapore.
* **Enhanced Embedding Models**: Exploring other types of embeddings, such as transformer-based contextual embeddings, could capture even more nuanced job attributes. Techniques like contextualized language models (e.g., BERT-based embeddings) might enable better matching of job descriptions to candidate preferences (Kenthapadi et al., 2017) .
* **Expansion to Other Domains**: The graph-based recommender could be extended to other domains where content and relationships are similarly complex, such as academic publications, online courses, or real estate listings, where users would benefit from highly personalized recommendations based on multidimensional relationships.

**References**

* Balog, K., & Kenter, T. (2020). Personalized job recommendation: A deep learning approach. *ACM SIGIR Conference on Human Information Interaction and Retrieval*. doi:10.1145/3373509.3373548.
* Diaby, M., Viennet, E., & Launay, T. (2014). Toward the Next Generation of Recruitment Tools: An Online Social Network-Based Job Recommender System. *Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 369–372. doi:10.1109/ASONAM.2014.6921618.
* Trivedi, R., Faruqui, M., Sankar, A., & Srivastava, S. (2017). Representation Learning over Dynamic Knowledge Graphs. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 33–40. doi:10.18653/v1/D17-1005.
* Paparrizos, I., Cambazoglu, B. B., & Gionis, A. (2011). Machine Learned Job Recommendation. Proceedings of the Fifth ACM Conference on Recommender Systems, 325–328. doi:10.1145/2043932.2043996.
* Shalaby, W., Zadrozny, W., Vydiswaran, V., & Soliman, A. (2017). A Framework for Effective Job Matching Using Semantic Similarity and Graph Theory. Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 1975–1978. doi:10.1145/3132847.3133167.
* Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function Using NetworkX. Proceedings of the 7th Python in Science Conference (SciPy2008), 11–15.
* Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. Advances in Neural Information Processing Systems (NeurIPS), 1025–1035. doi:10.5555/3294771.3294869.
* Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. International Conference on Learning Representations (ICLR), arXiv preprint arXiv:1710.10903.
* Wang, H., Zhang, F., Zhao, M., Li, W., Xie, X., & Guo, M. (2019). Multi-Task Feature Learning in Job Recommendation. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 1541–1550. doi:10.1145/3292500.3330848.
* Kenthapadi, K., Papadimitriou, P., & Vassilvitskii, S. (2017). Personalized Job Recommendation at Scale: Current State, Challenges, and Future Directions. Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 1365–1366. doi:10.1145/3077136.3096469.