Wadhifati: A Job Search Engine

Salha Al Said[1], Lilibeth Reales[2], Imran Ahmed[3], Afra Al Hadhrami[4]

20F20247@mec.edu.om[1], lilibeth@mec.edu.om[2], imran@mec.edu.om [3]

Dept. of Computing and Electronic, Engineering, Middle East College, Muscat, Oman[1, 2, 3, 4]

Abstract. The web application Wadhifati, meaning "my job" in Arabic, aims to assist job seekers in Oman. Many graduates face difficulties in finding suitable positions due to the confusion of available listings. Navigating several websites to find a posting that meets their qualifications further increases the difficulty, leaving many unemployed and unable to find a job suitable to their qualifications. Wadhifati uses web scraping to aggregate job postings from various websites, consolidating them into a single application to reduce the confusion and hassle of navigating multiple websites. The data-driven machine learning powered recommendation system uses Natural Language Processing with Sentence BERT to calculate semantic similarity using cosine similarity, providing a ranking of relevant jobs to users, enabling them to find jobs that align with their career path. As a result, the system would ease the job-search process and narrow down searches, potentially improving Oman's socioeconomic state.

Keywords: Machine Learning, Web Scraping, Job Searching, SBERT, NLP.

1 Introduction

The main objective of Oman's 2040 vision in the Economy and Development vision includes an employment system that encourages efficiency and innovation. Oman is projected to have a population growth of 2.4 million by 2040, with the percentage of Omanis in the working age rising to 69% [1]. Many job seekers struggle with finding roles that match their qualifications due to the overwhelming number of job listings scattered across different platforms, this issue will only increase with time [2].

The job searching application Wadhifati addresses this problem directly by web scraping job postings from various sources and gathering them in one place using Selenium for automated web scraping, reducing the need to check multiple sites. The user interface, developed using HTML, CSS, and JavaScript, is backed by a machine learning recommendation system. The algorithm utilizes Python, Sentence-BERT (SBERT) and Natural Language processing (NLP) to rank jobs based on user input skill and qualifications keywords, providing job recommendations to the user. Wadhifati aims to give employment seekers a tool to enhance their job searching capabilities with the hope of tackling one of the biggest issues faced currently and assisting in Oman's 2040 vision.

MECSRC 2025, May 01, Muscat, Oman
Copyright © 2026 EAI
DOI 10.4108/eai.1-5-2025.2361390

2 Literature Review

Studies emphasize the challenges and opportunities in improving job search. A case study from Sohar University reveals that Omani graduates rely on the internet to find jobs but face difficulties in navigating and finding the proper tools [3]. Research on Indeed.com proposes a machine learning enhanced solution to improve relevance by filtering out mismatched postings, vocabulary size is crucial to the improvement of these machine learning solutions [4]. Users favour the first page of searches and city specific searches, these trends can aid in the optimization of search engines [5]. For data storage, Firebase was identified as a powerful backend solution for web and mobile applications, offering real time data syncing and scalability, making it suitable for building data driven platforms [6].

3 Datasets, Preprocessing and Usage

The Wadhifati machine learning job recommendation system involved several occupational datasets sourced from the ONET (Occupational Information Network) framework [7]. These datasets were used to enhance keyword extraction and semantic matching between job listings and user submitted keywords. In addition to the ONET datasets, job listings were collected through a web scraper from various websites.

3.1 Job Listings Dataset (Web Scraping)

A real-world dataset is obtained through web scraping current job openings from Bayt.com [8], NaukriGulf.com [9], and PetroJobs.om [10]. The web scraper was implemented using Python and Selenium to extract the company name, job title, job description, source, and link to the listing. This data is not used for training, instead it contains the target jobs used by the recommendation system to rank after processing each users' keywords.

3.2 ONET Datasets

To extract relevant occupational keywords from user keywords and align them with job listings, six datasets from ONET were utilized. Each of the datasets below contains terminology that reflects job dimensions, attached with ONET-SOC (Occupational Information Network Standard Occupational) codes for job identification.

Occupation Data dataset: This dataset includes descriptions and phrases associated with job occupations, this dataset helped in identifying job specific terms within user keywords and job listings [11].

Technology Skill dataset: This dataset contains technologies involved with each ONET-SOC occupation, examples of the various technology and tools include Adobe "Acrobat" or "ERP systems". This aided in classifying a user's expertise with an industry [12].

Skills dataset: This dataset included employability skills such as "Critical Thinking" and "Coordination", to measure a candidates fit to a job based on soft skills [13].

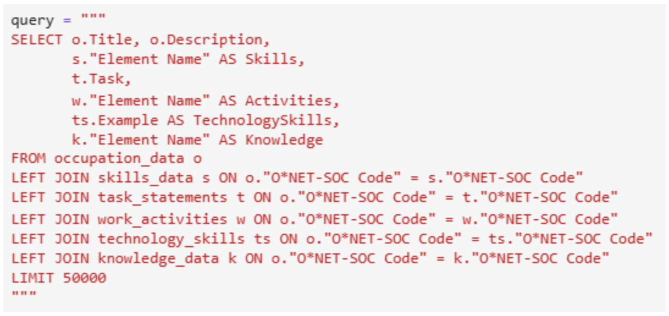
Knowledge dataset: The dataset includes knowledge domains associated with each job, such as "Economics and Accounting" and "Mechanical Engineering". This was useful for extracting academic domains from user keywords and match them to job [14].

Task Statements dataset: Includes task statements of occupations. This was used to create a link between functions described in user keywords and job listings [15].

Work Activities dataset: Represents the core work responsibilities, such as "Documenting Information" or "Interacting with Computers", to provide additional context for aligning user experience with job [16].

3.3 Data Integration

Figure 1 below is a code snippet show casing the integration of the ONET datasets their shared ONET-SOC codes to create a complete data representation of jobs.

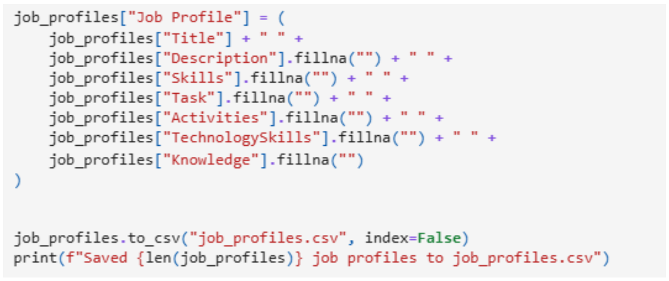


```
query = """
SELECT o.Title, o.Description,
       s."Element Name" AS Skills,
       t.Task,
       w."Element Name" AS Activities,
       ts.Example AS TechnologySkills,
       k."Element Name" AS Knowledge
FROM occupation_data o
LEFT JOIN skills_data s ON o."O*NET-SOC Code" = s."O*NET-SOC Code"
LEFT JOIN task_statements t ON o."O*NET-SOC Code" = t."O*NET-SOC Code"
LEFT JOIN work_activities w ON o."O*NET-SOC Code" = w."O*NET-SOC Code"
LEFT JOIN technology_skills ts ON o."O*NET-SOC Code" = ts."O*NET-SOC Code"
LEFT JOIN knowledge_data k ON o."O*NET-SOC Code" = k."O*NET-SOC Code"
LIMIT 50000
"""
```

Fig. 1. ONET Datasets Integration.

3.4 Data Preprocessing

Figure 2 below is the data preprocessing. The dataset is cleaned by removing null values and each ONET category is merged into a string called a "Job Profile" to create a data representation of each job for the machine learning model to interpret.

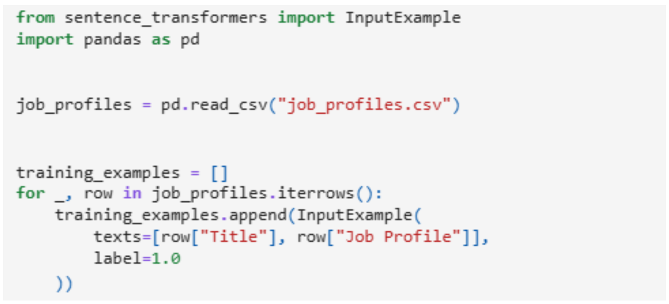


```
job_profiles["Job Profile"] = (
    job_profiles["Title"] + " " +
    job_profiles["Description"].fillna("") + " " +
    job_profiles["Skills"].fillna("") + " " +
    job_profiles["Task"].fillna("") + " " +
    job_profiles["Activities"].fillna("") + " " +
    job_profiles["TechnologySkills"].fillna("") + " " +
    job_profiles["Knowledge"].fillna("")
)


job_profiles.to_csv("job_profiles.csv", index=False)
print(f"Saved {len(job_profiles)} job profiles to job_profiles.csv")
```

Fig. 2. Data Preprocessing.

3.5 Data Usage

Figure 3 illustrates the usage of the job profiles to train the SBERT model. The job profiles and job titles are passed as pairs to the SBERT model as examples for training. This logic uses semantic similarity to indicate job titles alignment with descriptions, tools, tasks and domains to determine its fit, enabling the model to learn the context behind jobs and score the semantic similarity between a user's keywords and a job listing.



```python
from sentence_transformers import InputExample
import pandas as pd


job_profiles = pd.read_csv("job_profiles.csv")


training_examples = []
for _, row in job_profiles.iterrows():
    training_examples.append(InputExample(
        texts=[row["Title"], row["Job Profile"]],
        label=1.0
    ))
```

Fig. 3. Data Usage.

4 Implementation and Tools

Wadhifati was developed as a web application interface combining web scraping, machine learning, and a real time database. This section outlines the tools used for the implementation of the system.

4.1 Core Technologies

Python & Flask: Used to implement the client-server communication and integrate the backend with the frontend.

HTML/CSS/JavaScript: Used for designing the frontend user interface.

Sentence-BERT (SBERT): Used for the machine learning recommendation model and fine-tuned using paraphrase-MiniLM-L6-v2.

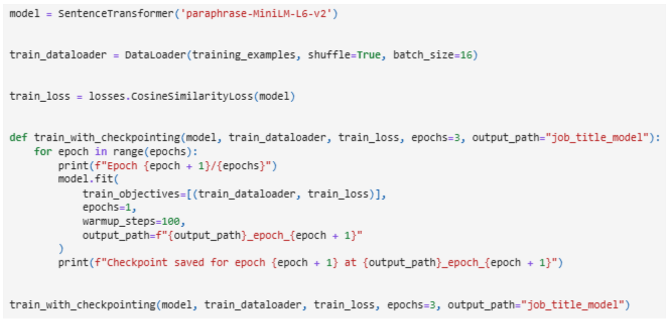
Firebase Firestore: A NoSQL database used to store job listings, user data, and recommendations.

Selenium and BeautifulSoup: To carry out the automated scraping of job listings from Bayt.com, NaukriGulf.com, and PetroJobs.om.

Visual Studio Code: The development environment for building the application.

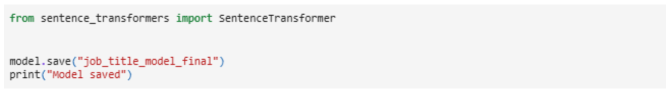
4.2 Model Training

Using the job profiles obtained through the data preprocessing of the ONET datasets, the pretrained SBERT model is fine-tuned with cosine similarity loss to focus on contextual meaning. The training is set to 3 epochs with a checkpoint implemented after each epoch to save progress to have better control over the performance stability. After training the model is saved locally for integration into the Flask application, as seen below in figures 4 and 5



```python
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')


train_dataloader = DataLoader(training_examples, shuffle=True, batch_size=16)


train_loss = losses.CosineSimilarityLoss(model)


def train_with_checkpointing(model, train_dataloader, train_loss, epochs=3, output_path="job_title_model"):
    for epoch in range(epochs):
        print(f"Epoch {epoch + 1}/{epochs}")
        model.fit(
            train_objectives=[(train_dataloader, train_loss)],
            epochs=1,
            warmup_steps=100,
            output_path=f"{output_path}_epoch_{epoch + 1}"
        )
        print(f"Checkpoint saved for epoch {epoch + 1} at {output_path}_epoch_{epoch + 1}")


train_with_checkpointing(model, train_dataloader, train_loss, epochs=3, output_path="job_title_model")
```

Fig. 4. Model Training.



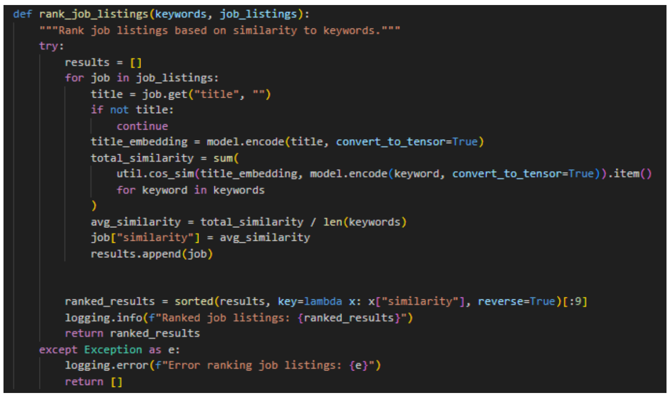
```python
from sentence_transformers import SentenceTransformer


model.save("job_title_model_final")
print("Model saved")
```

Fig. 5. Saving the Model.

4.3 Model Deployment

The SBERT model was saved locally and integrated into the Flask application, the model retrieves the users' submitted keywords and encodes it, comparing it to job titles from the Firestore database. It calculates the Cosine similarity between job titles and these keywords then stores the top nine results for the user in the database. These recommendations are stored for each user and accessible in the interface, presenting them with personalized recommendations.

The figure below represents the code used to deploy the ranking of job listings for users. The function takes two inputs, keywords and job listings, converting it into a vector using the SBERT model. The encoded data is used to calculate the cosine similarity between the inputs, computing the average similarity for each job and sorting it from highest to lowest.

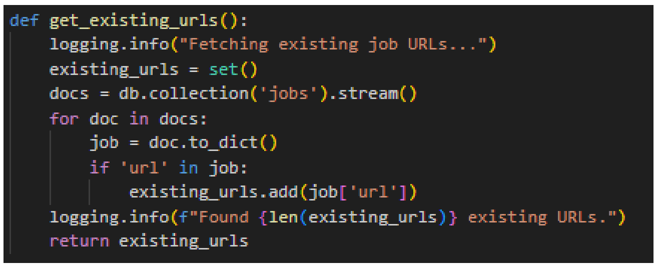


```python
def rank_job_listings(keywords, job_listings):
    """Rank job listings based on similarity to keywords."""
    try:
        results = []
        for job in job_listings:
            title = job.get("title", "")
            if not title:
                continue
            title_embedding = model.encode(title, convert_to_tensor=True)
            total_similarity = sum(
                util.cos_sim(title_embedding, model.encode(keyword, convert_to_tensor=True)).item()
                for keyword in keywords
            )
            avg_similarity = total_similarity / len(keywords)
            job["similarity"] = avg_similarity
            results.append(job)


        ranked_results = sorted(results, key=lambda x: x["similarity"], reverse=True)[:9]
        logging.info(f"Ranked job listings: {ranked_results}")
        return ranked_results
    except Exception as e:
        logging.error(f"Error ranking job listings: {e}")
        return []
```

Fig. 6. Rank job listing's function.

4.5 Job Scraping Pipeline

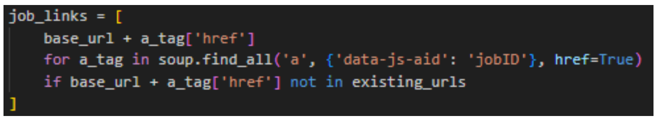
A custom scraping pipeline was developed for each website and integrated into the system. The scraper was implemented using Selenium for dynamic content and BeautifulSoup for scraping job titles, company names, descriptions, source websites, and URLS from Bayt.com, PetroJobs.om and GulfTalent.com. The code snippet in figure 7 shows the scraper checking for existing URLS in the Firestore database to create a list.



```python
def get_existing_urls():
    logging.info("Fetching existing job URLs...")
    existing_urls = set()
    docs = db.collection('jobs').stream()
    for doc in docs:
        job = doc.to_dict()
        if 'url' in job:
            existing_urls.add(job['url'])
    logging.info(f"Found {len(existing_urls)} existing URLs.")
    return existing_urls
```

Fig. 7. Get existing URLS function.

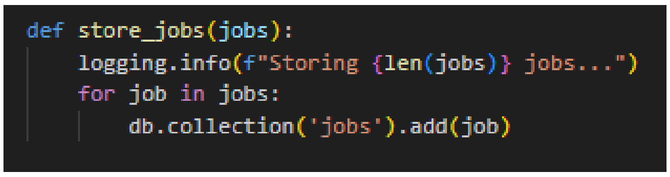
This list of existing URLS is used in the scraping function in the figure below to avoid duplication by skipping existing URLS.



```python
job_links = [
    base_url + a_tag['href']
    for a_tag in soup.find_all('a', {'data-js-aid': 'jobID'}, href=True)
    if base_url + a_tag['href'] not in existing_urls
]
```

Fig. 8. Remove existing URLS.

The figure below is a code snippet to save collected jobs to the firebase jobs collection.



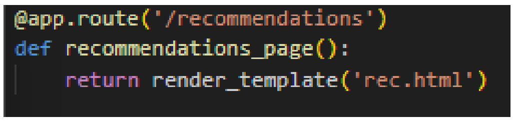
```python
def store_jobs(jobs):
    logging.info(f"Storing {len(jobs)} jobs...")
    for job in jobs:
        db.collection('jobs').add(job)
```

Fig. 9. Store Jobs function.

5 Web Application deployment

The application was deployed as a Python Flask web service using Visual Studio Code to link the frontend and backend processes. API routes were used to communicate between the frontend and backend, trigger the backend logic, and begin automation tasks.

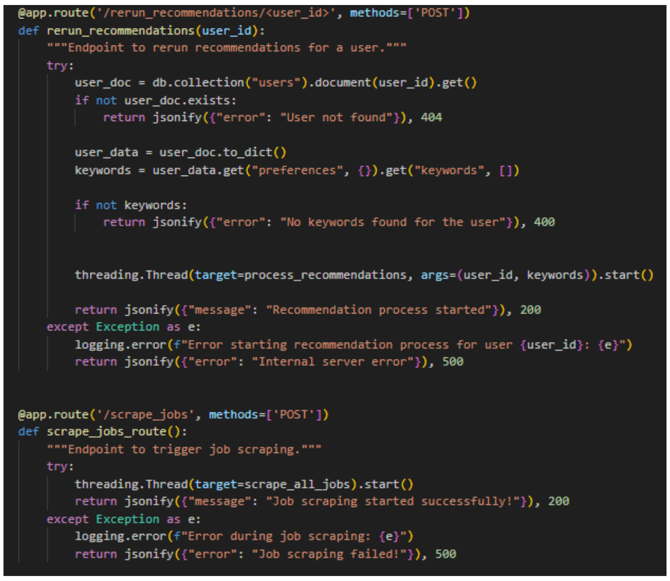
The code snippet in figure 10 shows an example of the HTML pages are rendered using flask routes to link it to the backend.



```python
@app.route('/recommendations')
def recommendations_page():
    return render_template('rec.html')
```

Fig. 10. HTML Routing.

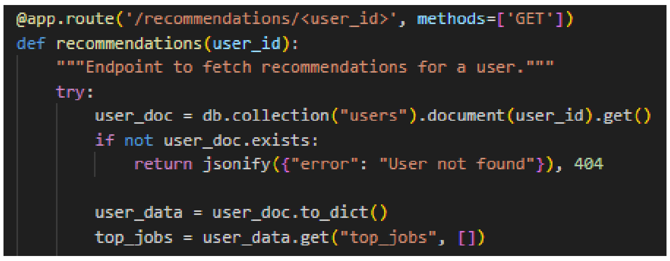
Figure 11 is the routes to trigger the job recommendation and web scraping.



```python
@app.route('/rerun_recommendations/<user_id>', methods=['POST'])
def rerun_recommendations(user_id):
    """Endpoint to rerun recommendations for a user."""
    try:
        user_doc = db.collection("users").document(user_id).get()
        if not user_doc.exists:
            return jsonify({"error": "User not found"}), 404

        user_data = user_doc.to_dict()
        keywords = user_data.get("preferences", {}).get("keywords", [])

        if not keywords:
            return jsonify({"error": "No keywords found for the user"}), 400


        threading.Thread(target=process_recommendations, args=(user_id, keywords)).start()

        return jsonify({"message": "Recommendation process started"}), 200
    except Exception as e:
        logging.error(f"Error starting recommendation process for user {user_id}: {e}")
        return jsonify({"error": "Internal server error"}), 500


@app.route('/scrape_jobs', methods=['POST'])
def scrape_jobs_route():
    """Endpoint to trigger job scraping."""
    try:
        threading.Thread(target=scrape_all_jobs).start()
        return jsonify({"message": "Job scraping started successfully!"}), 200
    except Exception as e:
        logging.error(f"Error during job scraping: {e}")
        return jsonify({"error": "Job scraping failed!"}), 500
```

Fig. 11. Trigger functions.

Figure 12 is an example of the routing to fetch the personalized recommendations for users to access through the interface.



```
@app.route('/recommendations/<user_id>', methods=['GET'])
def recommendations(user_id):
    """Endpoint to fetch recommendations for a user."""
    try:
        user_doc = db.collection("users").document(user_id).get()
        if not user_doc.exists:
            return jsonify({"error": "User not found"}), 404

        user_data = user_doc.to_dict()
        top_jobs = user_data.get("top_jobs", [])
```

Fig. 12. Fetch recommendations.

6 Results and Evaluation

For the performance evaluation of the learning recommendation model, a set of diverse test cases were selected to simulate the keyword of 10 different job seeking users. These keywords were paired with expected job titles they align with to compare to the models predicted job titles. This evaluation measures the model's ability to return relevant job titles from a pool of 50 job roles.

6.1 Evaluation Metrics

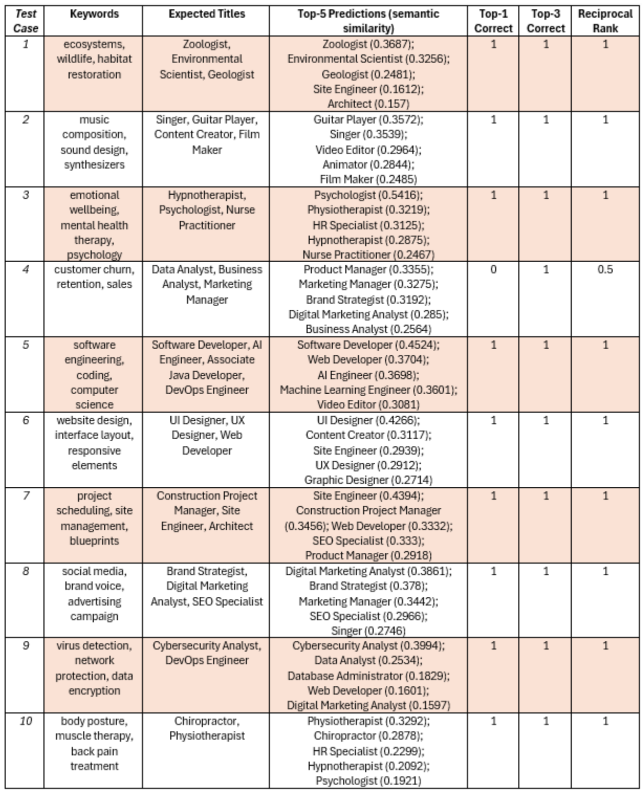
Top 1 Accuracy: This evaluates the model by verifying whether the top ranked job title in the prediction list matches the expected outcomes. It reflects how often the system identifies the most relevant job title as its top suggestion; a higher accuracy indicates that the model semantically calculating the appropriate roles.

Top 3 Accuracy: It considers the top three predictions. This will verify whether the model is correctly predicting the rest of the outcomes, this is more realistic as not only the top recommendation is being considered.

Mean Reciprocal Rank (MRR): This method is useful for weighting decision making criteria. It will determine the model's relevance by calculating the average position of jobs.

6.2 Test Cases

The Test cases and results for each test cases are in figure 13. The top 1 Accuracy demonstrates that 9 out of 10 cases were correct in their first ranked predictions, test case 4 was the only case with an incorrect top 1. The top 3 accuracy has a perfect score indicating all the test cases had an expected job title within the top 3 ranked jobs. The MRR is a 0.95 indicating the model rankings are relevant.

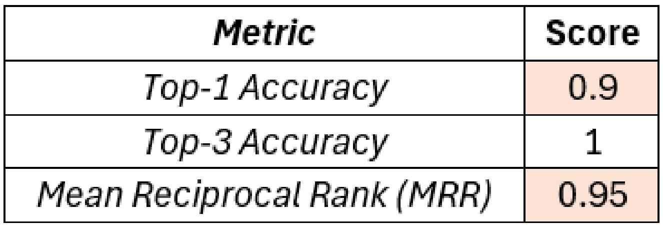


Test Case	Keywords	Expected Titles	Top-5 Predictions (semantic similarity)	Top-1 Correct	Top-3 Correct	Reciprocal Rank
1	ecosystems, wildlife, habitat restoration	Zoologist, Environmental Scientist, Geologist	Zoologist (0.3687); Environmental Scientist (0.3256); Geologist (0.2481); Site Engineer (0.1612); Architect (0.157)	1	1	1
2	music composition, sound design, synthesizers	Singer, Guitar Player, Content Creator, Film Maker	Guitar Player (0.3572); Singer (0.3539); Video Editor (0.2964); Animator (0.2844); Film Maker (0.2485)	1	1	1
3	emotional wellbeing, mental health therapy, psychology	Hypnotherapist, Psychologist, Nurse Practitioner	Psychologist (0.5416); Physiotherapist (0.3219); HR Specialist (0.3125); Hypnotherapist (0.2875); Nurse Practitioner (0.2467)	1	1	1
4	customer churn, retention, sales	Data Analyst, Business Analyst, Marketing Manager	Product Manager (0.3355); Marketing Manager (0.3275); Brand Strategist (0.3192); Digital Marketing Analyst (0.285); Business Analyst (0.2564)	0	1	0.5
5	software engineering, coding, computer science	Software Developer, AI Engineer, Associate Java Developer, DevOps Engineer	Software Developer (0.4524); Web Developer (0.3704); AI Engineer (0.3698); Machine Learning Engineer (0.3601); Video Editor (0.3081)	1	1	1
6	website design, interface layout, responsive elements	UI Designer, UX Designer, Web Developer	UI Designer (0.4266); Content Creator (0.3117); Site Engineer (0.2939); UX Designer (0.2912); Graphic Designer (0.2714)	1	1	1
7	project scheduling, site management, blueprints	Construction Project Manager, Site Engineer, Architect	Site Engineer (0.4394); Construction Project Manager (0.3456); Web Developer (0.3332); SEO Specialist (0.333); Product Manager (0.2918)	1	1	1
8	social media, brand voice, advertising campaign	Brand Strategist, Digital Marketing Analyst, SEO Specialist	Digital Marketing Analyst (0.3861); Brand Strategist (0.378); Marketing Manager (0.3442); SEO Specialist (0.2966); Singer (0.2746)	1	1	1
9	virus detection, network protection, data encryption	Cybersecurity Analyst, DevOps Engineer	Cybersecurity Analyst (0.3994); Data Analyst (0.2534); Database Administrator (0.1829); Web Developer (0.1601); Digital Marketing Analyst (0.1597)	1	1	1
10	body posture, muscle therapy, back pain treatment	Chiropractor, Physiotherapist	Physiotherapist (0.3292); Chiropractor (0.2878); HR Specialist (0.2299); Hypnotherapist (0.2092); Psychologist (0.1921)	1	1	1

Fig. 13. Test Cases.

6.3 Overall results

The overall performance results of the model demonstrate a high level of precision as can be seen in figure 14. The top 1 Accuracy demonstrates that 9 out of 10 cases were correct in their first ranked predictions. The top 3 accuracy has a perfect score indicating all the test cases had an expected job title within the top 3 ranked jobs. The MRR is a 0.95 indicating the model is highly effective in ranking relevant job titles.



Metric	Score
Top-1 Accuracy	0.9
Top-3 Accuracy	1
Mean Reciprocal Rank (MRR)	0.95

Fig. 14. Overall Results.

7 Conclusion

This research presented Wadhifati, a job recommendation web application designed to support job seekers by integrating web scraping, machine learning, and cloud-based storage. The system offers personalized job recommendations based on user submitted keywords with very promising results. It can understand the deep context behind the meanings of words across different categories with various keywords, such as relating the "music composition" keyword with the "singer" job title. This suggests that the system can aid job seekers in finding job roles targeting various industries and categories.

In addition to the machine learning, Wadhifati's web scraping pipeline successfully automated the process of aggregating job listings from websites and stored them on Firebase, eliminating the noise of irrelevant listings and reducing the need to check each website individually. Adding more websites to the scraping algorithm would increase the pool of jobs and enhance the job seeking experience as the model performs better with a larger variety of jobs, a limited pool results in irrelevant listings being placed to the top of rankings.

Overall, Wadhifati bridges a critical gap in the job search process as intended. Wadhifati holds the potential to become an impactful solution within the Omani labour market, which would ultimately aid in Oman's 2040 vision.

References

[1] Oman Vision 2040 Implementation Follow-up Unit. (2023). Oman Vision 2040. https://www.oman2040.om/uploads/publication/20231105221146-2023-11-05publication221143_.pdf

[2] National Center for Statistics and Information. (2015). Population projections in the Sultanate of Oman (2015–2040). https://www.ncsi.gov.om/Elibrary/LibraryContentDoc/ben_population%20projections%20In%20Sultanate%20of%20Oman_fa17fe2c-34fe-4d6b-93fd-1f7746e7b23e.pdf

[3] Al Jahwari, N., Alkhalidi, M., AlBalushi, S., Al Ghanami, M., Al Omairi, Z., & Khan, F. (2020). Job-Search behaviour of Omani Graduates: Sohar University - A Case study. International Journal of Research in Entrepreneurship & Business Studies, 1(2), 24–32. https://doi.org/10.47259/ijrebs.123

[4] Marin, I., & Amel, H. (2023). Web Platform for Job Recommendation Based on Machine Learning. Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering. https://doi.org/10.5220/0011993600003464

[5] Kim, Y. (2013). Analysis of Query Entries of a Job Search Engine. Design, User Experience, and Usability. Web, Mobile, and Product Design, 203–211. https://doi.org/10.1007/978-3-642-39253-5_22

[6] Gaikwad, A., Yadav, V., & Chougale, P. (2022). FIREBASE -OVERVIEW AND USAGE. International Research Journal of Modernization in Engineering Technology and Science, 3(12). https://doi.org/2582-5208K .

[7] National Center for O*NET Development. O*NET 29.2 Database. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/database.html

[8] Bayt. (n.d.). Jobs in Oman. Bayt. https://www.bayt.com/en/oman/jobs/

[9] NaukriGulf. (n.d.). Jobs in the Gulf. https://www.naukrigulf.com

[10] PetroJobs Oman. (n.d.). PetroJobs Oman. https://www.petrojobs.om

[11] National Center for O*NET Development. Occupation Data - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/occupation_data.html
[12] National Center for O*NET Development. Technology Skills - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/technology_skills.html
[13] National Center for O*NET Development. Skills - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/skills.html
[14] National Center for O*NET Development. Knowledge - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/knowledge.html
[15] National Center for O*NET Development. Task Statements - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/task_statements.html
[16] National Center for O*NET Development. Work Activities - O*NET 29.2 Data Dictionary. O*NET Resource Center. Retrieved April 18, 2025, from https://www.onetcenter.org/dictionary/29.2/excel/work_activities.html