ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

COLLABORATIVE FILTERING ALGORITHMS FOR A JOB RECOMMENDATION SYSTEM BUILT WITH A MICROSERVICE ARCHITECTURE

Yurii Sosnovskyi, Yevdokym Fedorchuk

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
Authors' e-mails: *yurii.sosnovskyi.mnpzm.2023@lpnu.ua, yevdokym.n.fedorchuk@lpnu.ua*

https://doi.org/10.23939/acps2025.01.047

Submitted on 20.04.2025

***Abstract*: This article presents the development of a recommender system for recruiting personnel and vacancies to improve the efficiency of the hiring process. The proposed system has integrated collaborative and hybrid filtering methods to provide personalized job recommendations. Collaborative filtering model has analyzed historical data, identifying patterns by detecting connections between user information and job content. Research methodology has included literature analysis, dataset preparation, developing and training the Alternating Least Squares (ALS) model, and effectiveness evaluation of the implemented collaborative filtering algorithm by accuracy and performance metrics. Another part of the research is focused on integrating the recommendation module into an existing job and candidate search system built using microservices.**

***Index terms*: recommendation system, collaborative filtering, ALS model, microservices.**

I. INTRODUCTION

In the context of an ever-evolving business environment, the processes of job searching and professional talent acquisition continue to gain critical importance. As the demands of the labor market grow increasingly dynamic and complex, there is a clear need for the development of advanced software solutions and algorithmic frameworks that can adequately address current market requirements. These systems must not only eliminate the limitations of existing recruitment platforms but also introduce innovative functionality and intuitive tools that facilitate seamless interaction between employers and candidates.

The primary practical issue addressed by this research is the inefficiency of conventional methods for identifying relevant specialists and the prolonged hiring process in the context of today's labor market. This challenge is tackled through the implementation of a recommendation system based on machine learning algorithms, which can rapidly and accurately analyze large volumes of data and perform comparative analysis to identify the most relevant matches. This facilitates the formation of optimal candidate–position pairs, enhancing satisfaction levels among both employers and job seekers. The proposed system aims to resolve delays in recruitment by offering tailored job recommendations to candidates and suggesting potential applicants to recruiters, there by simplifying decision-making and improving interactions between hiring parties and professionals. The integration of artificial intelligence enables the development of a comprehensive candidate evaluation system that accounts for various skill sets and personal attributes [1]. The core business logic involves matching vacancies to the candidate, ranking them by relevance, and presenting the most suitable options. High scalability and accessibility of the software solution are ensured through the adoption of a microservice-based architecture. Thus, the project enhances the overall efficiency, speed, and accuracy of the recruitment process, providing a reliable software tool.

A key area of technological advancement in this domain is the use of collaborative filtering algorithms, which play a central role in generating personalized recommendations by analyzing user preferences and behavioral patterns. In particular, modern systems increasingly rely on implicit feedback – such as clicks, browsing time, application submissions, or profile views – rather than explicit user ratings, which are often unavailable in recruitment platforms [2]. Usually, implicit feedback is safer because unregulated feedback loops generate harmful side effects that ultimately impact the recommendation quality [3]. By leveraging large-scale historical interaction data, collaborative filtering enables the system to infer latent factors that describe the relationships between users and job postings, thereby providing recommendations that are both relevant and adaptive to user behavior over time.

The integration of collaborative filtering methods that utilize implicit feedback allows for the creation of more responsive and intelligent systems capable of continuously learning from user activity. This results in enhanced matching accuracy, greater user satisfaction, and improved hiring outcomes. Consequently, a thorough investigation into these approaches is essential for the development of effective recruitment recommendation systems that can operate efficiently in real-world conditions and at scale.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

The recruitment industry is no exception in the application of recommendation algorithms and smart search. In recent years, notable advancements in recommender system research have been documented across various academic publications and implemented in software solutions. A recommender system functions as an automated mechanism that delivers personalized suggestions of products or services to users [4]. It operates based on a defined set of users ($K_1...K_i$) and a collection of recommended items ($E_1...E_k$). Within this context, a user refers to any individual or entity interacting with the system, while an item represents a physical or digital entity – such as a job posting, user profile, or product – communicated to the user via email, text message, or graphical interface. These items are characterized by distinct attributes that form the basis of their comparison and recommendation production.

Among the various approaches employed in recommender systems, collaborative filtering (CF) stands out as one of the most prevalent and widely applied methods [5]. CF techniques are generally categorized into two main types: memory-based and model-based. Model-based collaborative filtering leverages machine learning models trained on historical interaction data to uncover latent patterns and make informed predictions about user preferences. In contrast, memory-based methods rely directly on the user-item interaction matrix – either in full or sampled form – to compute recommendations in real time. This latter approach is further divided into user-based and item-based filtering [6]. User-based filtering generates recommendations by identifying users with similar behavioral histories, whereas item-based filtering focuses on identifying correlations between items based on user evaluations. Comparative studies have shown that item-based filtering often yields more accurate but less tailored recommendations than user-based approaches due to features of item similarity metrics [7].

A number of studies focus on improving intelligent recommender systems. Authors use collaborative algorithms to optimize recommendations in this specific domain [8]. There are two different aspects to measuring accuracy. The quantitative aspect is based on indicators such as the mean absolute error and the root mean square error, which is the root of the mean error of all estimates obtained by the algorithm used [9]. And the qualitative aspect depends on the direct result of the recommendations, and we evaluate it by reviewing the generated recommendation.

Another challenge faced by collaborative filtering is the sparsity problem, where there are not enough interactions between users and items to generate reliable recommendations, which can occur in large datasets or for specialized products that are not widely used. Several approaches have been proposed to address this problem, such as matrix factorization models [10], vector decomposition, clustering, and graph-based algorithms. For example, singular vector decomposition has been used to compact the original user-item matrix, and latent semantic models have been used to cluster users and items. However, these approaches have the disadvantage that the decomposition needs to be updated every time a new user or rating is added to the matrix. A more advanced approach is based on the analysis of prediction errors to improve the accuracy of user-based filtering. The drawback of this approach is that computing the errors of all estimates during training is quite resource-intensive [11]. In addition, the above methods must be modified to avoid the cold start problem. The simplest CF process is based on three steps, as shown in Fig. 1.

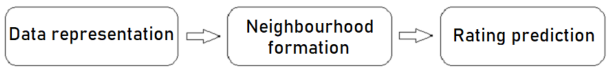


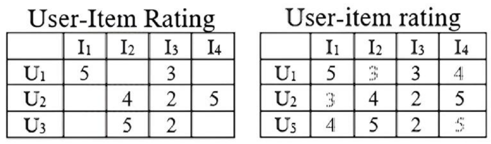
Fig. 1. The process of collaborative recommendation generation

Before starting the collaborative filtering process, the first step is to collect data from users who need recommendations [12]. This data (in our case, candidate and job ratings) serves as a query to the algorithm. The second step of the CF is to build a rating matrix and fill it in. Since users do not rate items regularly, the most common technique in CF is to replace the empty cells of the matrix with the average user ratings. In this step, we look for the neighborhood of the most similar users using a similarity metric. There are different measures for obtaining similarity, but the most common are Pearson's correlation coefficient and cosine similarity [13]. This is considered the standard way to measure correlation. Values obtained using the equation below range from –1 to 1 (–1 means that there is an opposite similarity between two objects, 0 means that there is no correlation between those, and 1 means that they are identical).

$$sim\left(P_{u_i}, V_{i_j}\right) = \frac{\sum_{l=1}^{k} w_l v_l}{\sqrt{\sum_{l=1}^{k} w_l^2} \sqrt{\sum_{l=1}^{k} v_l^2}}. \quad (1)$$

The cosine similarity (1) between the vectors P_{ui} and V_{ij} is used to determine the relevance of element i_j to user u_i, where w_l and v_l are the coordinates of these vectors, i, j and l are counters that lie in the range from 1 to k.

At the last step, after selecting the closest k-neighbors for the active user [14], using the degree of similarity based on the score matrix (Fig. 2), the CF generates predictions for previously missing objects.


User-Item Rating

	I_1	I_2	I_3	I_4
U_1	5		3	
U_2		4	2	5
U_3		5	2	

User-item rating

	I_1	I_2	I_3	I_4
U_1	5	3	3	4
U_2	3	4	2	5
U_3	4	5	2	5

Fig. 2. An example of a score matrix for data representation

A number of researchers in their works conclude that microservices architecture with autonomous databases is well suited for systems with a high user load, which is our product. According to this hypothesis, breaking the system into small and autonomous microservices will allow for reliability enhancement, efficient implementation of new features, and easier further software support [15].

III. SCOPE OF WORK AND OBJECTIVES

The main purpose of the work is to improve the process of selecting candidates and vacancies by using collaborative filtering algorithms. Development of a recommendation system based on the best practices of microservice architecture. To achieve this objective, the following research tasks have been carried out:

- Dataset preparation.
- Building and training an ALS model for generating recommendations.
- Designing and creating a software implementation of a recommender module as part of a microservice platform for job search and recruitment.
- Conducting an experiment, testing the developed software system, and fixing defects.
- Analyzing the results obtained according to the criteria of relevance (RMSE – Root Mean Squared Error, MAE – Mean Absolute Error) and performance (computing resource consumption).
- The final stage involves integrating the selected recommendation strategy into the complete software solution.

The expected practical significance of the research is to increase the probability of successful hiring, as well as to optimize the time and material costs of selecting a specialist or a place of work through the implementation of personalized recommendations. The potential social result of the software product is to reduce pressure on the labor market, reduce the unemployment rate and provide specialists with an effective new means of employment.

Potential users of the system are candidates looking for a job or are in a passive job search, students, freelancers who are open to new opportunities, along with recruiters, hiring managers, and HR professionals.

The research methodology is based on a comprehensive approach that includes the analysis of relevant academic sources, market analysis of existing systems, development of a functional prototype, and experimental validation of the proposed solution. Special emphasis is placed on the model optimizing and data preparation, which facilitates better accuracy indicators and reduces computational costs.

IV. SYSTEM ARCHITECTURE DESIGN

Modern high-load recommendation systems designed for matching candidates with job opportunities require a high level of flexibility, scalability, and efficient integration of various components. One of the most effective design paradigms for building such systems is the microservices architecture. The specified approach enables the development of independent software services, each responsible for a well-defined and limited set of functionalities. These services (or modules) can be deployed independently and scaled individually based on specific performance demands or load characteristics.

By decomposing the overall system into smaller, modular components, the development and maintenance of each service can be conducted in isolation, often by separate teams. This leads to enhanced flexibility in implementing changes, reduces the risk of unintended disruptions due to updates, and ensures more efficient allocation and utilization of computing resources. Additionally, the adoption of a microservices-based architecture improves system resilience, as a failure in one service does not necessarily compromise the functionality of the entire platform.

In the proposed system, the inter-service communication is realized through Hypertext Transfer Protocol (HTTP) requests within the framework of Representational State Transfer (REST) design. This method provides a standardized and technology-agnostic interface for service interaction, enabling seamless integration and interoperability regardless of the specific implementation stack used across different components. The utilization of message brokers or alternative message bus architectures is deemed suboptimal due to the predominantly synchronous nature of operations, obviating the necessity for an event-driven paradigm. A detailed representation of the recommender subsystem, an integral component of the web platform, is provided in the deployment diagram illustrated in Fig. 3.

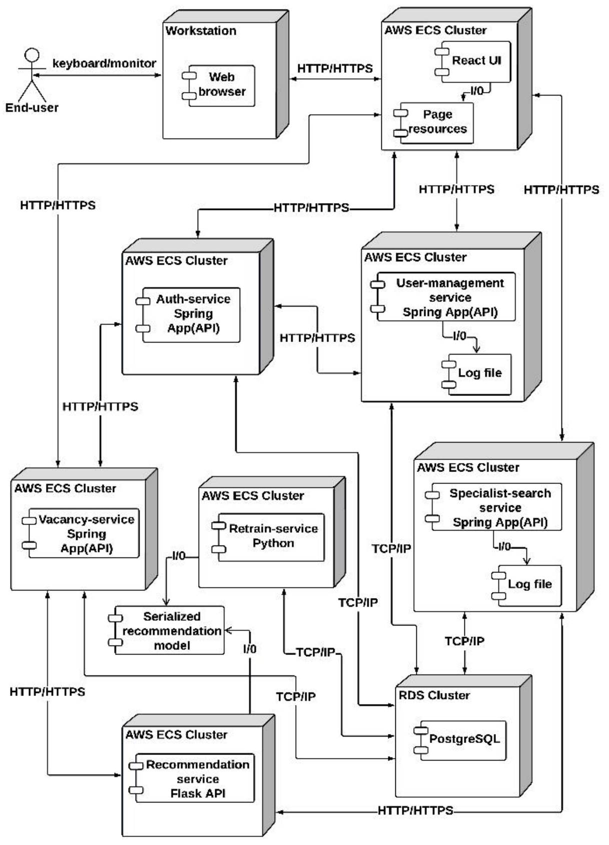


Fig. 3. Microservices web platform deployment diagram

A typical usage scenario involves the Vacancy-Service receiving user requests, subsequently either performing direct vacancy searches or dispatching a request to the Recommendation-Service endpoint for personalized suggestions. Concurrently, the Recommendation-Service processes HTTP GET requests, leverages a pre-trained model to generate a recommendation list, and transmits these recommendations back to the Vacancy-Service. The Vacancy-Service then augments these identifiers with supplementary vacancy details for presentation on the user interface. Notably, the recommendation microservice operates without direct database interaction. An analogous workflow is employed by the Specialist-search-service to procure candidate recommendations. The deployment diagram (see Fig. 3) delineates the system's physical architecture, exemplified by the Amazon Web Services (AWS) Elastic Container Service (ECS) cluster and a Relational Database Service (RDS) instance for the PostgreSQL database, illustrating the distribution and interrelation of software and hardware components. The serialized model file is persisted in AWS Simple Storage Service (S3). Furthermore, the Retrain-service is a candidate for migration from AWS ECS to the AWS Lambda serverless compute service, with its execution scheduled via the integrated Cron job mechanism, triggered by periodic events from Amazon CloudWatch Events.

V. DATASET PREPARATION AND PROCESSING

The development of the recommendation system begins with the preparation of a dataset about candidates – for this, we will use the existing Structured Query Language (SQL) database of the web platform for searching specialists. The Spring Data toolkit was used to read the data from tables. We have access to repository abstraction, which allows us to easily work with domain entities using various data access technologies, in particular the Hibernate framework, the JPA interface and tools for SQL databases. For aggregation, a utility has been written that writes flatly structured data about specialists to a CSV file, which is easy to work with Python Pandas Data frame.

The same data persistence sequence was also implemented for existing vacancy records and view history. During the dataset analysis, categorical and numerical data were identified. Empty or incorrect values were detected and handled. If the percentage of such columns for a record exceeded 40%, that entry was removed from the resulting dataset. Among the obtained dataset of candidate-vacancy interactions, outliers were identified in the "reviews" column (number of views), which we will use to generate recommendations based on implicit feedback. A total of 615 such rows were found, which is approximately 5.5 % of all data (Fig. 4). Such a quantity is considered uncritical, and these records are removed to improve model accuracy.

For the analysis of user behavior on the job search platform, the data are structured as a collection of distinct files, each encapsulating unique aggregated information. These files undergo continuous updates during the periodic model retraining performed by the Retrain-service, wherein novel information is appended and obsolete, non-pertinent records are purged.

The underlying data source is a PostgreSQL database; complex aggregated queries directed at this relational database can exhibit comparatively protracted execution durations. Therefore, a decision was made to execute these queries asynchronously in a background process, thereby circumventing the latency associated with synchronous, real-time result retrieval characteristic of a memory-based recommendation system.

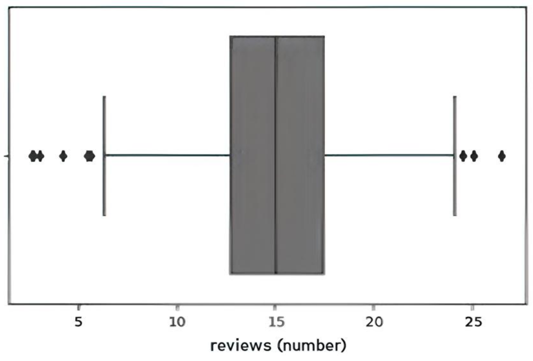


Fig. 4. Boxplot for the "reviews" category

VI. ALS MODEL TRAINING

A highly prevalent and efficacious method for the realization of collaborative filtering is the Alternating Least Squares (ALS) algorithm. This algorithm optimizes the factorization of the interaction matrix by iteratively updating the latent parameters of users and vacancies. The implementation of this approach within the recommendation system ensures a high level of precision in forecasting pertinent vacancies for candidates, a cardinal objective of the present investigation.

The ALS algorithm is predicated on the principle of matrix factorization, which entails the decomposition of the interaction matrix into two lower-rank matrices [16]. One matrix embodies the latent factors characterizing users, while the other represents the latent factors associated with vacancies. This methodology facilitates the identification of latent interrelationships between candidates and employment opportunities, even in instances of limited historical interaction data. In its conventional form, this algorithm optimizes parameters such as the dimensionality of the latent space, regularization coefficients, and the number of iterative steps to incrementally improve predictive accuracy.

The principal merit of the adopted algorithm resides in its ability to efficiently handle sparse datasets, a prevalent issue in recruitment platforms where a substantial proportion of user interactions remains implicitly represented. In contradistinction to gradient descent algorithms, which concurrently update all parameters, ALS employs an alternating update scheme for the latent factors of candidates and vacancies, solving a system of linear equations at each iterative step. This approach guarantees robust convergence, even in the presence of a large cardinality of users and vacancies. Prior to the initiation of model training, a rigorous preprocessing and cleansing of the input data is imperative – procedures elucidated in the preceding section. Subsequent to successful data generation, the training phase of the neural network model can be undertaken. For this purpose, Python version 3.11, the pip3 package manager, and the Implicit, Scipy, and Scikit libraries were deployed.

A pivotal stage comprises the formulation of a candidate-vacancy interaction matrix, incorporating data on views, applications, and other parameters indicative of user engagement. Recognizing the prevalence of implicit feedback in recruitment platforms, an implicit collaborative filtering strategy was implemented. Consequently, each user query, vacancy view, or application submission was transformed into a corresponding numerical rating, allowing the ALS algorithm to precisely quantify the relevance of a vacancy for a given candidate.

Given the substantial dimensions and inherent sparsity of the original matrix, the Implicit library was utilized for its manipulation. This library offers an efficient implementation of the ALS algorithm, providing scalability for extensive datasets and adhering to a Model-Based paradigm. All computations were executed in a sparse matrix format via the Sparse submodule of the Scipy library, thereby significantly reducing computational resource demands and expediting model training.

Model training was conducted in a multi-stage process. The initial phase involved the determination of the optimal latent dimensionality, representing the number of latent factors within the factorized matrices. Empirical studies demonstrated that selecting a value within the range of 50 to 100 factors achieves a judicious equilibrium between model performance and recommendation precision. The subsequent phase entailed the calibration of the regularization coefficient, which serves to prevent model overfitting and enhance its generalization capacity.

The model was trained on historical user-vacancy interaction data (number of views), and its predictive efficacy was assessed on a distinct test dataset. Standard recommendation system evaluation metrics, specifically Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), were employed to quantify the accuracy of the predictions.

VII. MODEL SERIALIZATION AND ITS INTEGRATION INTO A MICROSERVICE PLATFORM

Model serialization and its deployment via an Application Programming Interface (API) are foundational elements of modern machine learning architectures. These processes enable the efficient persistence, transmission, and retrieval of models across heterogeneous execution environments. The primary objective is to facilitate the seamless deployment of a model trained in one environment to another, thereby obviating the necessity for redundant retraining. This ensures solution agility, scalability, and accessibility for a wide array of applications and end-users, including other software system modules.

Serialization entails the transformation of a model object into a byte stream, which can be readily stored on persistent storage, transmitted across network infrastructure, or archived in cloud-based repositories. This process is of paramount importance for distributed systems wherein machine learning models necessitate deployment across disparate servers or environments (development, quality assurance, production). For the preservation of the trained model's state to enable its subsequent utilization, Pickle, a standard Python serialization utility, was adopted. Periodic model retraining is proposed on a bi-hourly schedule to maintain the currency of the recommendation repository.

Following serialization, the model can be deployed as an integral module of a web application, accessible via an API that functions as an interface between the machine learning model and end-users. In this specific implementation, communication is mediated through a REST API leveraging the Flask framework. RESTful API constitutes an architectural paradigm for the development of web services, predicated on the HTTP protocol and employing its methods (GET, POST, PUT, DELETE, et cetera) for interaction with server-side resources. Each resource is uniquely identified by a Uniform Resource Identifier (URI), enabling client access.

A key attribute of RESTful APIs is their stateless nature. This implies that each request encapsulates all necessary information for its processing, and the server does not maintain any client-specific state between successive requests. This characteristic facilitates system scalability and enhances reliability. Integration with the developed recommendation retrieval API was implemented using Feign Client, a library facilitating simplified interaction with external REST-based web services. It is a constituent of the Spring Cloud ecosystem and is realized through a declarative programming paradigm.

VIII. TESTING OF THE RECOMMENDATION MODULE AND ANALYZING THE RESULTS

A critical phase in the implementation and evaluation process of the developed predictive model is validation. The initial step involves a preliminary functional verification of the model's capacity to return a valid list of recommendations, commonly referred to as Smoke-testing. This aims to assess the end-to-end functionality of the software application, focusing on core business logic and essential operational capabilities.

The subsequent phase entails rigorous API testing. This stage is paramount in verifying the correct operational behavior of the Recommendation-Service and its seamless integration with other system components, notably the Vacancy-Service and Specialist-Search-Service. Automated REST API testing was conducted utilizing Postman and pytest in conjunction with the Python's Requests Library. Key validation procedures included assessing the accurate processing of recommendation retrieval requests, HTTP method adherence, parameter validation, conformity of input and output data formats (JSON schema compliance), as well as the handling of boundary conditions, such as requests with absent or malformed parameters. Test scenarios confirmed the API's ability to correctly process client requests, returning personalized recommendations with a mean response latency of 160ms under standard load conditions.

An analytical evaluation of the developed software module was performed based on accuracy metrics (Mean Absolute Error, Mean Squared Error, Root Mean Squared Error) and performance metrics (computational resource utilization). During evaluation on a sequestered test dataset (approximately 2760 vacancies), the error metric values obtained were *MAE* = 0.831, *RMSE* = 1.063, and *MSE* = 1.129. These results indicate a high degree of congruence between the model's predictions and empirical data, although residual deviations suggest potential avenues for further optimization. The findings demonstrate that the adapted collaborative filtering methodology employing the Alternating Least Squares algorithm yielded a consistent improvement ranging from 1.8 % to 3.2 % across varying training and testing set sizes for MAE, and from 2.0 % to 4.1 % for RMSE.

Laboratory-based efficiency testing of the model was executed on a compute-optimized medium AWS c7i.2xlarge instance featuring 16 GB of Random Access Memory (RAM) and 8 virtual Central Processing Units (vCPUs), powered by an Intel Xeon Platinum 8488C processor with a base clock frequency of 2.4 GHz (3.8 GHz peak), operating under the Amazon cloud computing infrastructure. Analysis of performance indicators (RAM and CPU utilization) revealed a significant advantage of the model-based recommendation algorithm. Local testing on an Intel Core i7-9750H processor at 2.6 GHz with 16 GB of DDR4 RAM yielded substantially similar results. Peak memory and central processing unit resource consumption reached 91 % and 84 %, respectively, for the memory-based approach, whereas the model-based approach exhibited maximum utilization levels of 67 % for RAM and 58 % for CPU (Fig. 5). Load testing was conducted utilizing Apache JMeter software with 50 concurrent threads, at an aggregate throughput of 200 requests per second over a duration of 100 seconds. Recommendations were generated for a randomly selected candidate from a cohort of 11,000 existing candidates for each request.

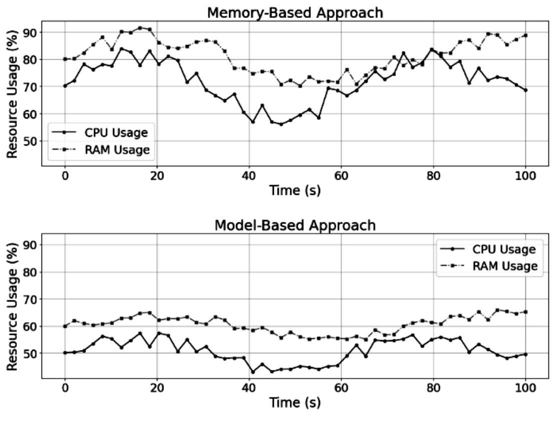


Fig. 5. Performance comparison graphs of memory-based and model-based approaches

The service demonstrated stable performance with a mean response latency of 215 ms, which remains within an acceptable threshold for real-time systems. Analysis of the service's log data confirmed the absence of critical errors or anomalous operational delays in the API, thereby validating its suitability for scalable web service deployment.

These results substantiate the rationale for a subsequent experimental investigation aimed at evaluating the effectiveness of the developed recommendation system within authentic operational environments. Specifically, the efficacy of the generated recommendations, system performance under realistic load conditions, and the potential for enhancing the recruitment process through automated candidate and vacancy matching may be assessed in the scope of further research.

The investigation will be conducted on a live, deployed system integrated within a functional job search web platform. The experimental cohort will consist of two distinct user categories: recruiters, who will receive automatically generated lists of prospective candidates, and candidates, who will receive personalized vacancy recommendations predicated on their skill sets, professional experience, and historical interactions with the platform.

Over a longitudinal period of two months, statistical data pertaining to user engagement with the recommendation system will be collected, and a comprehensive analysis of its effectiveness will be performed. The accuracy of the generated recommendations will be quantified using Click-Through Rate (CTR) and Conversion Rate (CVR) metrics. It is hypothesized that the CTR for recommended vacancies will exhibit a statistically significant increase of 15–20 % in comparison to vacancies identified through standard search functionalities. Furthermore, the CVR (defined as the number of applications submitted for recommended vacancies) is projected to increase by 10–15 %. The mean time expenditure by users on manual vacancy searches will also be quantified and compared against the time allocated to interacting with recommended vacancies. A statistically significant reduction of 25–30 % in the mean vacancy search time is anticipated. Load testing will be executed under authentic operational conditions, simulating an estimated daily active user base of 10,000. Analysis of computational resource utilization (CPU, RAM, API response latency) will be conducted. The anticipated mean response latency of the Recommendation-Service is within the range of 150–300 milliseconds.

IX. CONCLUSION

This research investigated several pivotal facets in the creation of effective recommendation systems leveraging collaborative filtering, model serialization, and API integration. These components are critically significant for the design of modern, scalable, and performant machine learning systems that can be deployed across a diverse range of services.

Particular attention was directed towards the Alternating Least Squares (ALS) method, a widely adopted iterative algorithm for the decomposition of the interaction matrix into the product of two lower-rank matrices representing user factors and item factors. It was functioned by minimizing the mean squared error of ratings to achieve dimensionality reduction and the extraction of latent factors from the data. In the context of this study, this enabled the modeling of relationships between users and vacancies, even in the absence of direct observed interactions.

It was determined that the development of the software product employing a microservice architecture ensures high scalability and fault tolerance. This architectural paradigm facilitated the facile updating and extension of the system, thereby enhancing its performance and reliability characteristics. A demonstration of machine learning model deployment via a REST API was conducted, enabling users to interact with the model without requiring knowledge of its internal structure or implementation specifics. The concluding phase of the research involved the validation of the developed model, the analysis of relevant performance metrics, and experimentation. In summary, the proposed system exhibits potential for implementation in the automation of personnel and vacancy matching processes, which could exert a positive impact on the contemporary labor market.

References

[1] Huamán, A., Rebaza, G., & Subauste, D. (2024). Hybrid Job Recommendation Model Based on Professional Profile Using Data from Job Boards and Machine Learning Libraries. *Proceedings of the 18th International Multi-Conference on Society, Cybernetics and Informatics: IMSCI 2024,* pp. 72–79. DOI: https://doi.org/10.54808/IMSCI2024.01.72.

[2] Liu, Y., Xiao, Y., Wu, Q., Miao, C., Zhang, J., Zhao, B., & Tang, H. (2020). Diversified Interactive Recommendation with Implicit Feedback. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(4), 4932–4939. DOI: https://doi.org/10.1609/aaai.v34i04.5931.

[3] Krauth, K., Wang, Y., & Jordan, M. (2025). Breaking Feedback Loops in Recommender Systems with Causal Inference. *ACM Transactions on Recommender Systems*. DOI: https://doi.org/10.1145/3728372.

[4] Jain, A., & Gupta, C. (2018). Fuzzy logic in recommender systems. *Fuzzy logic augmentation of neural and optimization algorithms: Theoretical aspects and real applications*, 255–273. DOI: https://doi.org/10.1007/978-3-319-71008-2_20.

[5] Liu, D. (2018, December). A Study on Collaborative Filtering Recommendation Algorithms. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)* (pp. 2256–2261). IEEE. DOI: https://doi.org/10.1109/compcomm.2018.8780979.

[6] Chornous, G., Nikolskyi, I., Wyszyński, M., Kharlamova, G., & Stolarczyk, P. (2021). A hybrid user-item-based collaborative filtering model for e-commerce recommendations. *Journal of International Studies*, *14*(4). 157–173. DOI: https://doi.org/10.14254/2071-8330.2021/14-4/11.

[7] Permana, K. E. (2024). Comparison of User Based and Item Based Collaborative Filtering in Restaurant Recommendation System. *Mathematical Modelling of Engineering Problems*, *11*(7), 1922–1928. DOI: https://doi.org/10.18280/mmep.110723.

[8] Xu, Y., Zhang, Y., Zhang, Y., & Li, H. (2023). Optimization of intelligent recommendation of innovation and entrepreneurship projects based on collaborative filtering algorithm. *Intelligent Decision Technologies*, *17*(1), 1–13. DOI: https://doi.org/10.3233/idt-230313.

[9] Sitkar, S., Teslyuk, V., Yanchuk, I., Antonyuk, D., & Kohut, I. (2022). The intellectual system of movies recommendations based on the collaborative filtering. *Journal of Education, Health and Sport*, *12*(3), 115–127. DOI: https://doi.org/10.12775/jehs.2022.12.03.010.

[10] Nguyen, L. V., Vo, Q.-T., & Nguyen, T.-H. (2023). Adaptive KNN-Based Extended Collaborative Filtering Recommendation Services. *Big Data and Cognitive Computing*, *7*(2), 106. DOI: https://doi.org/10.3390/bdcc7020106.

[11] Nguyen, L. V., Nguyen, T.-H., Vo, Q.-T., & Le, T. A. (2020). Cognitive Similarity-Based Collaborative Filtering Recommendation System. *Applied Sciences*, *10*(12), 4183. DOI: https://doi.org/10.3390/app10124183.

[12] El Fazziki, A., Lachhab, A., & Mouloudi, A. (2022). Employing opposite ratings users in a new approach to collaborative filtering. *Indonesian Journal of Electrical Engineering and Computer Science*, *25*(1), 450–459. DOI: https://doi.org/10.11591/ijeecs.v25.i1.pp450-459.

[13] Muhammad, M. (2021). Recommendation System Using User-Based Collaborative Filtering and Spectral Clustering. In *Proceedings of the 1st International Seminar on Teacher Training and Education, ISTED 2021* (pp. 516–521). EAI. DOI: https://doi.org/10.4108/eai.17-7-2021.2312409.

[14] Kwieciński, R., Filipowska, A., Górecki, T., & Dubrov, V. (2023). Job recommendations: benchmarking of collaborative filtering methods for classifieds. *arXiv e-prints*, arXiv-2301. DOI:https://doi.org/10.48550/arXiv.2301.07946.

[15] Melnyk, K., Borysova, N., Kochuieva, Z., & Huliieva, D. (2021). Towards Designing of Recommendation System for Recruiting of Software Development Teams. *Computer Modeling and Intelligent Systems*, *2864*, 226–237. DOI: https://doi.org/10.32782/cmis/2864-20.

[16] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE. DOI: https://doi.org/10.1109/icdm.2008.22

Yurii Sosnovskyi was born in Lviv. Starting from 2023, he is a student of the Master's educational and scientific program there. Since 2021, he has been working as a Java Software Engineer, his current position is Senior Software Developer at ELEKS. Yurii's areas of interest are backend engineering, enterprise systems, cloud computing, machine learning and recommender systems.

Yevdokym Fedorchuk graduated from the Faculty of Physics of Ivan Franko National University of Lviv, majoring in Radiophysics and Electronics. Since 2000 and up to the present time, he has been working as an Associate Professor at the Department of Software Engineering of Lviv Polytechnic National University. He is the author of over 50 scientific publications and educational-methodical developments