ДОДАТОК А

Апробація результатів роботи

A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit

Vladyslav Lapin
Software Engineering Department
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
vladyslav.lapin@nure.ua

Kirill Smelyakov
Software Engineering Department
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
kyrylo.smelyakov@nure.ua

Anastasiya Chupryna
Software Engineering Department
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
anastasiya.chupryna@nure.ua

Zoia Dudar
Software Engineering Department
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
zoia.dudar@nure.ua

***Abstract*—Personalized recommendation systems play a crucial role in enhancing user experiences by providing tailored suggestions based on individual preferences and contextual factors. This paper presents a hybrid approach in developing a recommendation system for selecting locations to visit, integrating user-defined filters, contextual data, and collective user feedback. The proposed system leverages a deep neural network to analyze various inputs, including explicit user preferences (e.g., desired atmosphere, type of location, etc.), dynamic contextual factors (e.g., weather conditions, temperature, etc.), and historical user data (e.g., ratings, recommendation trends for similar preferences, etc.). By combining content-based filtering with collaborative filtering techniques, the model aims to improve the accuracy and relevance of recommendations. The system classifies locations as suitable or unsuitable based on the given criteria, providing users with adaptive and context-aware suggestions. The hybrid nature of the approach allows for a more comprehensive understanding of user needs while incorporating real-time environmental conditions. Experimental validation is conducted to assess the effectiveness of the model in generating accurate recommendations. The results highlight the advantages of integrating multiple data sources and deep learning techniques to enhance accuracy and achieve high-quality recommendations. This research contributes to the development of intelligent recommendation systems by proposing a scalable and adaptable framework for personalized location selection.**

***Index Terms*—recommendation systems, deep learning, hybrid approach, personalization, location-based services, contextual filtering**

I. Introduction

Recommendation systems have become an essential component of modern digital services, offering personalized suggestions based on user preferences and behavior. With the increasing availability of contextual and behavioral data, there is a growing need for more adaptive, intelligent, and personalized recommendation mechanisms, particularly in domains such as travel and location-based services.

Traditional recommendation approaches, such as collaborative filtering and content-based methods, have been widely employed [1].

Recent advancements have introduced deep learning techniques to enhance recommendation systems. Convolutional Neural Networks (CNNs), for instance, have been integrated to improve feature extraction and representation learning, particularly in domains like computer vision [2] and speech recognition. By incorporating CNN architectures [3], recommendation systems can process high-dimensional data more effectively, leading to more accurate and personalized suggestions.

Despite these advancements, challenges persist in the field of recommendation systems. Issues such as data sparsity, scalability, and the need for diversified recommendations remain prominent [4]. Addressing these challenges requires continuous exploration of innovative methodologies and the integration of multimodal data sources [5] to improve system performance.

This paper explores a hybrid approach [6] to developing a recommendation system aimed at personalized selection of locations for a visit. Unlike conventional systems that rely solely on static user preferences or historical ratings, the proposed method incorporates both content-based filtering and collaborative filtering, enriched with contextual data such as weather conditions, time of day, and user mood. The system also allows users to provide detailed filters, enabling a more fine-grained and situation-aware recommendation process [7].

II. Data Description

The development and evaluation of the proposed recommendation system are based on the Google Local Data (2021) dataset [8], which provides comprehensive information on various locations and their associated user reviews. This dataset includes both structured metadata and unstructured textual reviews, each linked by unique location identifiers.

A. Dataset Preparation

To ensure consistency and relevance, the dataset was preprocessed and segmented into training, validation, and test subsets. The training dataset comprises location data from the

Рисунок А.1 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 1)

states of California, Florida, and Texas, while the validation dataset consists of data from New York and Washington. This geographic partitioning allows the model to generalize across diverse regional patterns during evaluation.

As an initial step in data preparation, all available location categories were collected and their frequencies analyzed. Categories with fewer than 1,024 occurrences were removed to eliminate rare or poorly represented categories, thereby improving the reliability of learned patterns.

The core set of features extracted from the data includes: location name, category, average rating, number of reviews, geolocation coordinates (longitude and latitude).

In addition to structured data, a concatenated textual representation was constructed by merging the location description, miscellaneous metadata, and associated reviews. This composite text provides richer semantic context for content-based filtering.

B. Contextual Data Integration

A key enhancement in the proposed recommendation system is the integration of contextual data, which allows the model to provide more personalized and situation-aware suggestions. Rather than relying solely on static user preferences or generic location attributes, the system incorporates dynamic environmental and temporal factors that influence user decision-making in real-world scenarios.

Since the dataset contains timestamped reviews, it was possible to infer and associate contextual metadata with each user interaction. Specifically, for every review, the following contextual features were extracted or derived: time of day, seasonal information, temperature, weather conditions.

These contextual attributes were derived using historical weather data, synchronized with each review's timestamp and geolocation. The integration of these variables enables the recommendation system to consider situational suitability, for example, avoiding recommendations for outdoor locations during inclement weather or suggesting quiet indoor venues during late evening hours.

C. Test Dataset

To complement the training and validation phases, a dedicated test dataset was manually constructed to rigorously evaluate the model's performance under controlled and interpretable conditions. The test dataset consists of 70 carefully designed experimental scenarios, distributed across 10 distinct location categories, with 7 unique test cases per category. Each scenario was created to reflect a specific combination of input features, enabling a detailed assessment of the model's ability to handle varying data patterns and contextual conditions.

The test cases were structured as follows:

1) Average rating, average number of reviews, suitable context, suitable filters
2) Average rating, high number of reviews, suitable context, suitable filters
3) High rating, average number of reviews, unsuitable context, suitable filters
4) Low rating, average number of reviews, suitable context, suitable filters
5) High rating, average number of reviews, suitable context, unsuitable filters
6) Average rating, low number of reviews, suitable context, suitable filters
7) Average rating, average number of reviews, suitable context, suitable filters (distinct from Case 1)

This structured variation allows for a comprehensive evaluation of how well the model captures and responds to changes in key variables such as ratings, review volume, contextual appropriateness, and user-defined filters. By testing across multiple scenarios within each category, the system's sensitivity to different influencing factors can be measured, providing insights into its decision-making robustness, generalization ability, and contextual reasoning capabilities.

III. Hybrid Approach

To enhance the relevance and personalization of location recommendations, the proposed system employs a hybrid recommendation approach that combines the strengths of content-based filtering and collaborative filtering. This integration enables the system to simultaneously account for individual user preferences, real-world contextual conditions, and collective behavioral trends derived from historical user data.

A. Content-Based Filtering

Content-based filtering is a recommendation technique that focuses on analyzing the attributes of items and matching them to user-specific preferences. Instead of relying on the behavior or preferences of other users, this method evaluates the compatibility between a user's explicitly stated interests and the intrinsic characteristics of each item. In the context of recommendation systems, this approach enables personalized suggestions by identifying items whose features align closely with the user's profile or current needs.

In the proposed system, content-based filtering is implemented by incorporating a comprehensive set of user-defined filters and dynamic contextual factors directly into the recommendation process. The filters represent specific user requirements or desires, such as preferred atmosphere (e.g., quiet, lively), type of location (e.g., park, museum, cafe), or suitability for a particular mood, age group, or activity. These filters serve as key indicators of the user's intent and are encoded as input features to the model. Alongside these filters, the system integrates real-time contextual data including weather conditions, season, temperature, and time of day to refine the recommendation based on situational appropriateness. For instance, a user may express interest in visiting a park, but if the context indicates inclement weather or nighttime, the system will adjust its evaluation accordingly. This enables the model to make more nuanced decisions, considering not just what the user wants, but also whether the current environment supports that preference.

Рисунок А.2 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 2)

B. Collaborative Filtering

Collaborative filtering is a recommendation technique that generates suggestions based on the preferences, behaviors, and evaluations of other users. Rather than relying on the properties of items themselves, collaborative filtering identifies patterns in how users interact with items such as ratings, likes, or selections and uses this collective behavior to predict the preferences of similar users. The underlying assumption is that users with similar tastes in the past are likely to prefer similar items in the future.

In the proposed system, collaborative filtering is applied through the integration of historical user data, specifically focusing on two key aspects: location ratings and recommendation approvals or disapprovals. Each location in the dataset is associated with an aggregated average rating, reflecting general user satisfaction. Additionally, the system incorporates feedback on whether users found specific recommendations useful, particularly when provided under similar filter and context conditions. This helps capture not only how well a location is generally perceived, but also how effective it has been in satisfying users with comparable needs and situational constraints.

C. Hybrid Integration

Hybrid integration in recommendation systems [9] refers to the combination of multiple recommendation techniques to leverage the strengths of each and mitigate their individual limitations. By integrating both personalized attribute matching and collective behavioral insights, hybrid systems aim to produce more accurate, balanced, and context-aware recommendations. This approach helps overcome common challenges such as cold-start problems, overspecialization, and lack of adaptability to real-world conditions.

In the proposed system, hybrid integration is achieved by jointly incorporating all relevant data - user-defined filters, contextual variables, location attributes, historical ratings, and recommendation feedback into a unified model. Instead of treating content and collaborative signals separately, these diverse inputs are combined and processed simultaneously by a deep neural network, which learns complex relationships and interactions among features.

IV. Data Analysis Methods

To ensure optimal performance and compatibility with the deep learning model, extensive preprocessing and feature engineering were applied to the raw dataset. These steps aim to clean, transform, and encode the data into a structured numerical format suitable for training and inference.

A. Data Cleaning and Selection

The initial stage in the data processing pipeline involved data cleaning and selection [10], which is essential to eliminate noise, reduce redundancy, and retain only the most informative attributes for model training. The goal of this phase is to ensure that the dataset is well-structured, consistent, and aligned with the objectives of the recommendation system.

Only the most relevant features were retained, specifically:

- Category of the location
- Rating, representing the average user satisfaction
- Number of reviews, indicating the popularity
- Contextual information, including weather conditions, temperature, time of day, and season
- Summary information, which is a concatenation of location description, metadata, and user reviews

To standardize the text-based fields — category, contextual information, and summary information were first converted to lowercase to eliminate case-based inconsistencies. Next, all non-informative characters (punctuation, special symbols, etc.) were removed, followed by the elimination of stop words, which do not contribute meaningful semantic content. After normalizing the text, it was tokenized to split the text into individual meaningful terms (tokens), preparing it for further vectorization steps.

This comprehensive cleaning and selection process ensured that only high-quality, relevant, and interpretable data was passed to subsequent stages of transformation and modeling, thereby enhancing both the performance and reliability of the recommendation system.

B. Feature Transformation

To enhance the interpretability of the model and ensure numerical stability, several transformations were applied to key features.

The number of reviews varies significantly, ranging from very low to extremely high values. To reduce skewness and prevent large values from disproportionately influencing the model, a logarithmic transformation was applied:

$$TNR = log(1 + NR) \tag{1}$$

where TNR is the transformed number of reviews, NR is the original number of reviews. The addition of 1 prevents undefined values for locations with zero reviews.

To better capture the interplay between a location's rating and its popularity, a new feature - rating-reviews ratio was introduced:

$$RRR = (R - 3) \times TNR \tag{2}$$

where RRR is the rating-reviews ratio, R is the average rating of the location (ranging from 1 to 5), TNR is the transformed number of reviews. This feature is designed to reflect how the number of reviews affects the interpretation of a given rating. This transformation ensures that high ratings ($R > 3$) with many reviews increase the value of RRR, low ratings ($R < 3$) with many reviews decrease RRR, ratings close to 3 have minimal effect, preventing neutral ratings from dominating the feature distribution. Transformations help the model distinguish between highly rated, widely reviewed locations and polarizing locations with high engagement, enhancing its ability to make informed recommendations.

Рисунок А.3 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 3)

C. Feature Encoding and Scaling

To ensure compatibility with the deep learning model, categorical and numerical features were transformed into a standardized numerical format using encoding and scaling techniques [11], while text features were vectorized.

The category attribute was transformed using One-Hot Encoding. Given a total of N unique categories, each category was represented as a binary vector of length N:

$$C_i = [c_1, c_2, ..., c_N] \tag{3}$$

where C_i is the encoded vector of location i, $c_j = 1$ if the location belongs to category j, otherwise $c_j = 0$. Transformation ensures that categorical data is presented in a machine-readable format without introducing ordinal relationships.

Rating (R), log-transformed number of reviews (TNR), and rating-reviews ratio (RRR) were standardized using z-score normalization. The transformation is defined as:

$$X' = \frac{X - \mu}{\sigma} \tag{4}$$

where X' is the standardized value of the feature, X is the original value of the feature, μ is the mean of the feature, σ is the standard deviation of the feature. Process ensures that all numerical features have a mean of 0 and a standard deviation of 1, preventing features with large magnitudes from dominating the learning process.

Contextual and general information was vectorized using Term Frequency-Inverse Document Frequency (TF-IDF) [12]. TF-IDF score is computed as:

$$TF\text{-}IDF = TF(t, d) \times IDF(t) \tag{5}$$

$$TF(t, d) = No.\ of\ times\ term\ t\ appears\ in\ doc\ d \tag{6}$$

$$IDF(t) = log\frac{1 + n}{1 + df(t)} + 1 \tag{7}$$

where n is the total number of documents in the document set and $df(t)$ is the document frequency of t. The document frequency is the number of documents in the document set that contain the term t.

TF-IDF ensures that frequently occurring words within a document contribute more to its representation while reducing the impact of common words across all documents.

D. Dimensionality Reduction

To ensure that the deep neural network receives a fixed-size input while preserving the most important information, Principal Component Analysis was applied to all features simultaneously. This includes numerical attributes, categorical encodings, and TF-IDF vectorized text data.

This transformation results in a compact yet information-rich representation of each location, optimizing the feature space for the subsequent deep learning model.

V. Deep Neural Network

A FeedForward Neural Network [13] was implemented using PyTorch [14] to process the transformed features and make recommendations. The architecture was designed to balance performance and computational efficiency by varying the number of input neurons, hidden layers, and learning rates.

A. Architecture of Models

The number of neurons in the input layer varied across experiments, corresponding to the dimensionality of the data. The first hidden layer had twice as many neurons as the input layer, and each subsequent hidden layer halved the number of neurons until it reached the output layer. Each layer was followed by a LeakyReLU activation function and Dropout to prevent overfitting. A single neuron with a Sigmoid activation function was used to generate a probability score.

B. Training and Experimentation

A total of 27 models were trained using different configurations: input size - 256, 512, 1024; hidden layers - 3, 5, 7; learning rates - 0.01, 0.001, 0.0001.

Initial training was conducted for 10 epochs for all models. The models were evaluated based on accuracy (8) and F1-score (11), the results visualized in the accompanying graphs.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{11}$$

The accuracy of 27 different models illustrated in Fig. 1.

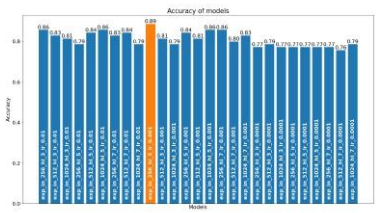


Fig. 1. Accuracy of 27 different models

The F1-Score of 27 different models illustrated in Fig. 2.

Рисунок А.4 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 4)

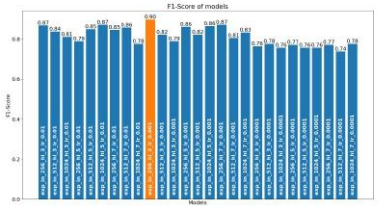


Fig. 2. F1-Score of 27 different models

The best-performing model achieved an accuracy of 0.89 and an F1-score of 0.90, demonstrating the effectiveness of the chosen hybrid recommendation approach.

The top 5 models were trained for 100 epochs to refine their performance.

The accuracy of top 5 models illustrated in Fig. 3.

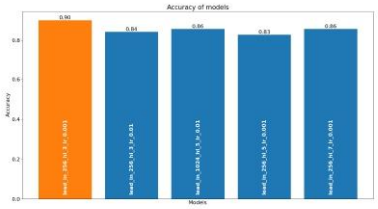


Fig. 3. Accuracy of top 5 models

The F1-Score of top 5 models illustrated in Fig. 4.

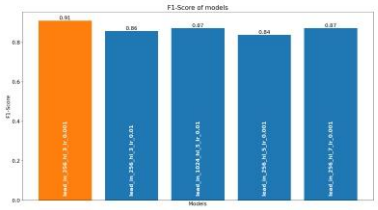


Fig. 4. F1-Score of top 5 models

Based on the accuracy and F1-score comparisons, the best model achieved the highest accuracy of 0.90 and the highest F1-score of 0.91, indicating that it provides the best balance between precision and recall.

VI. Experiments and Results

To evaluate the performance of the developed recommendation model, a series of experiments were conducted and the results were analyzed using multiple metrics, including accuracy, F1-score, the ROC-AUC curve, and the confusion matrix [15]. The results presented in Table I demonstrate the model's quality and reliability (T - true, P - predicted).

The AUC-ROC curve, illustrated in Fig. 5, demonstrates the model's ability to distinguish between classes. The Area Under the Curve is 0.91, indicating excellent performance.

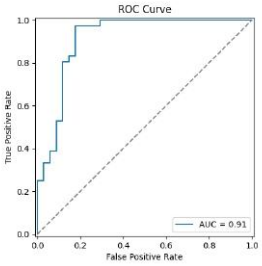


Fig. 5. AUC-ROC Curve

The confusion matrix, illustrated in Fig. 6, shows that the model correctly predicted 28 true negatives and 35 true positives, with only 6 false positives and 1 false negative. This suggests that the model is reliable for recommendation tasks.

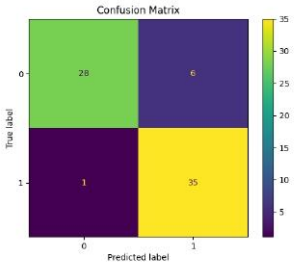


Fig. 6. Confusion matrix

One key strength of the model is its ability to understand contextual data. For example, in testing, it correctly avoided recommending parks and playgrounds in rainy weather, even when other suitability factors were met. This suggests that the model considers environmental conditions and does not make naive recommendations based solely on other attributes.

The model demonstrates an understanding of rating credibility. It does not overvalue high ratings when the number of reviews is extremely low, recognizing potential biases or artificial inflation. Conversely, it correctly penalizes low ratings even when they have an exceptionally high number of reviews, ensuring that frequently reviewed locations with consistently poor feedback are not recommended.

While the model performs well overall, there are cases where it struggles with filters. This may be due to similar terms appearing across different categories, leading to minor inaccuracies. However, the model still produces strong results, and its ability to continuously improve through user feedback will further enhance its accuracy and reliability.

Рисунок А.5 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 5)

TABLE I
RESULTS OF THE EXPERIMENTS

Category	Rating	Reviews	Context	Summary	T	P
park	avg	avg	suitable	suitable	1	1
park	avg	high	suitable	suitable	1	1
park	high	avg	unsuitable	suitable	0	0
park	low	avg	suitable	suitable	0	0
park	high	avg	suitable	unsuitable	0	0
park	avg	low	suitable	suitable	0	0
park	avg	avg	suitable	suitable	1	1
restaurant	avg	avg	suitable	suitable	1	1
restaurant	avg	high	suitable	suitable	1	1
restaurant	high	avg	unsuitable	suitable	1	1
restaurant	low	avg	suitable	suitable	0	0
restaurant	high	avg	suitable	unsuitable	0	1
restaurant	avg	low	suitable	suitable	0	0
restaurant	avg	avg	suitable	suitable	1	1
bar	avg	avg	suitable	suitable	1	1
bar	avg	high	suitable	suitable	1	1
bar	high	avg	unsuitable	suitable	1	1
bar	low	avg	suitable	suitable	0	0
bar	high	avg	suitable	unsuitable	0	0
bar	avg	low	suitable	suitable	0	0
bar	avg	avg	suitable	suitable	1	1
club	avg	avg	suitable	suitable	1	1
club	avg	high	suitable	suitable	1	1
club	high	avg	unsuitable	suitable	0	0
club	low	avg	suitable	suitable	0	0
club	high	avg	suitable	unsuitable	0	1
club	avg	low	suitable	suitable	0	0
club	avg	avg	suitable	suitable	1	1
bakery	avg	avg	suitable	suitable	1	1
bakery	avg	high	suitable	suitable	1	1
bakery	high	avg	unsuitable	suitable	1	1
bakery	low	avg	suitable	suitable	0	0
bakery	high	avg	suitable	unsuitable	0	1
bakery	avg	low	suitable	suitable	0	0
bakery	avg	avg	suitable	suitable	1	1
attraction	avg	avg	suitable	suitable	1	1
attraction	avg	high	suitable	suitable	1	1
attraction	high	avg	unsuitable	suitable	0	0
attraction	low	avg	suitable	suitable	0	0
attraction	high	avg	suitable	unsuitable	0	1
attraction	avg	low	suitable	suitable	0	0
attraction	avg	avg	suitable	suitable	1	1
playground	avg	avg	suitable	suitable	1	1
playground	avg	high	suitable	suitable	1	0
playground	high	avg	unsuitable	suitable	0	0
playground	low	avg	suitable	suitable	0	0
playground	high	avg	suitable	unsuitable	0	0
playground	avg	low	suitable	suitable	0	0
playground	avg	avg	suitable	suitable	1	1
coffee shop	avg	avg	suitable	suitable	1	1
coffee shop	avg	high	suitable	suitable	1	1
coffee shop	high	avg	unsuitable	suitable	1	1
coffee shop	low	avg	suitable	suitable	0	0
coffee shop	high	avg	suitable	unsuitable	0	1
coffee shop	avg	low	suitable	suitable	0	0
coffee shop	avg	avg	suitable	suitable	1	1
salon	avg	avg	suitable	suitable	1	1
salon	avg	high	suitable	suitable	1	1
salon	high	avg	unsuitable	suitable	1	1
salon	low	avg	suitable	suitable	0	0
salon	high	avg	suitable	unsuitable	0	0
salon	avg	low	suitable	suitable	0	0
salon	avg	avg	suitable	suitable	1	1
cafe	avg	avg	suitable	suitable	1	1
cafe	avg	high	suitable	suitable	1	1
cafe	high	avg	unsuitable	suitable	1	1
cafe	low	avg	suitable	suitable	0	0
cafe	high	avg	suitable	unsuitable	0	1
cafe	avg	low	suitable	suitable	0	0
cafe	avg	avg	suitable	suitable	1	1

VII. CONCLUSIONS

In this work, a hybrid recommendation model aimed at improving the selection of locations based on contextual and user-defined filters was developed and evaluated. Experiments demonstrated that the model effectively understands the ratio between rating and number of reviews, contextual factors, filters, and adjusts recommendations accordingly.

The results of the evaluation metrics confirm that the model distinguishes well between relevant and irrelevant recommendations. These findings suggest that the model is a promising tool for generating reliable and context-aware recommendations, with potential for further optimization and refinement.

REFERENCES

[1] K. Benabbes, K. Housni, A. El Mezouary and A. Zellou, "Recommendation System Issues, Approaches and Challenges Based on User Reviews," in Journal of Web Engineering, vol. 21, no. 4, pp. 1017–1054, June 2022.

[2] A. Arsenov, I. Ruban, K. Smelyakov and A. Chupryna, "Evolution of convolutional neural network architecture in image classification problems," Selected Papers of XVIII International Scientific and Practical Conference "Information1, and 27, 2018. In CEUR Workshop Proceedings, Vol-2318, 2018, pp. 35–45.

[3] J. H. Yoon and B. Jang, "Evolution of Deep Learning-Based Sequential Recommender Systems: From Current Trends to New Perspectives," in IEEE Access, vol. 11, pp. 54265–54279, 2023.

[4] I. Saifudin and T. Widiyaningtyas, "Systematic Literature Review on Recommender System: Approach, Problem, Evaluation Techniques, Datasets," in IEEE Access, vol. 12, pp. 19827–19847, 2024.

[5] J. Lv, B. Song, J. Guo, X. Du and M. Guizani, "Interest-Related Item Similarity Model Based on Multimodal Data for Top-N Recommendation," in IEEE Access, vol. 7, pp. 12809–12821, 2019.

[6] Z. Huang, C. Yu, J. Ni, H. Liu, C. Zeng and Y. Tang, "An Efficient Hybrid Recommendation Model With Deep Neural Networks," in IEEE Access, vol. 7, pp. 137900–137912, 2019.

[7] M. Li, W. Zheng, Y. Xiao, K. Zhu and W. Huang, "Exploring Temporal and Spatial Features for Next POI Recommendation in LBSNs," in IEEE Access, vol. 9, pp. 35997–36007, 2021.

[8] J. McAuley, "Google Local Data (2021)," McAuley, https://mcauleylab.ucsd.edu/public_datasets/gdrive/googlelocal/ (accessed Mar. 27, 2025).

[9] A. Chaudhari, A. A. Hitham Seddig, A. Sarlan and R. Raut, "A Hybrid Recommendation System: A Review," in IEEE Access, vol. 12, pp. 157107–157126, 2024.

[10] "Data cleaning and preprocessing," Medium, https://medium.com/@sahilbansal480/data-cleaning-and-preprocessing-9680b71e00c3 (accessed Mar. 27, 2025).

[11] "Feature Creation and Transformation," Medium, https://medium.com/@AILearningHub/feature-creation-and-transformation-60818ad568e7 (accessed Mar. 27, 2025).

[12] "Demonstrating Calculation of TF-IDF From Sklearn," Medium, https://medium.com/analytics-vidhya/demonstrating-calculation-of-tf-idf-from-sklearn-4f9526e7e78b (accessed Mar. 27, 2025).

[13] A. Shrestha and A. Mahmood, "Review of Deep Learning Algorithms and Architectures," in IEEE Access, vol. 7, pp. 53040–53065, 2019.

[14] "Feedforward Neural Network with PyTorch," Deep Learning Wizard, https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neuralnetwork/ (accessed Mar. 27, 2025).

[15] "Evaluation Metrics for Classification," Medium, https://medium.com/@mlmind/evaluation-metrics-for-classification-fc770511052d (accessed Mar. 27, 2025).

Рисунок А.6 – Стаття на тему «A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit» (сторінка 6)

ДОДАТОК Б

Код рекомендаційної системи

```
1.  import matplotlib.pyplot as plt
2.  import nltk
3.  import numpy as np
4.  import pandas as pd
5.  import torch
6.  import torch.nn as nn
7.  import torch.optim as optim
8.  from alive_progress import alive_bar, config_handler
9.  from config import Settings, default_settings
10. from network import NeuralNetwork
11. from nltk.corpus import stopwords
12. from nltk.stem import WordNetLemmatizer
13. from nltk.tokenize import word_tokenize
14. from preprocessing import CategoricalEncoder, Compressor,
NumericalScaler, TextVectorizer
15. from sklearn.metrics import (ConfusionMatrixDisplay, RocCurveDisplay,
accuracy_score, auc, confusion_matrix, f1_score,
16.                              precision_score, recall_score, roc_curve)
17. from utils import LocationDataLoader, LocationDataset
18.
19.
20. class RecommendationSystem:
21.
22.     _instance: Self | None = None
23.
24.     def __new__(cls, settings: Settings = default_settings) -> Self:
25.         if not isinstance(settings, Settings):
26.             raise TypeError("Invalid type of settings.")
27.
28.         if not isinstance(cls._instance, cls):
29.             cls._instance = super().__new__(cls)
30.
31.         return cls._instance
32.
33.     def __init__(self, settings: Settings = default_settings) -> None:
34.         config_handler.set_global(title_length=20,   monitor_end=False,
stats_end=False)
35.         torch.manual_seed(seed=42)
36.
37.         self._settings = settings
38.
39.         self._types = {
40.             "name": str,
41.             "category": str,
42.             "rating": np.uint8,
43.             "num_of_reviews": np.uint32,
44.             "latitude": np.float16,
45.             "longitude": np.float16,
46.             "context": str,
47.             "summary": str,
48.             "recommend": np.int8
49.         }
50.
51.         nltk.data.path.append("data/nltk")
52.
53.         if not os.path.exists("data/nltk"):
54.             nltk.download("punkt_tab", download_dir="data/nltk")
```

```
55.                 nltk.download("wordnet", download_dir="data/nltk")
56.                 nltk.download("stopwords", download_dir="data/nltk")
57.
58.             if not os.path.exists(f"models/{self._settings.model_name}"):
59.                 os.makedirs(f"models/{self._settings.model_name}")
60.
61.             self._device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
62.
63.             self._preprocessors = {
64.                 "encoders": {
65.                     "category": CategoricalEncoder(max_categories=8192),
66.                     "rating": NumericalScaler(),
67.                     "num_of_reviews": NumericalScaler(),
68.                     "rating_reviews_ratio": NumericalScaler(),
69.                     "context": TextVectorizer(max_features=4096),
70.                     "summary": TextVectorizer(max_features=16384)
71.                 },
72.                 "compressor":
Compressor(n_components=self._settings.model_dims[0])
73.             }
74.
75.             if
os.path.exists(f"models/{self._settings.model_name}/preprocessors.pkl"):
76.                 self._load_preprocessors()
77.             else:
78.                 self._prepare_preprocessors()
79.
80.             self._model                                                    =
NeuralNetwork(*self._settings.model_dims).to(self._device)
81.
82.             if
os.path.exists(f"models/{self._settings.model_name}/model.pth"):
83.                 self._load_model()
84.             else:
85.                 if                                                     not
os.path.exists(f"models/{self._settings.model_name}/output"):
86.
os.makedirs(f"models/{self._settings.model_name}/output")
87.                 else:
88.                     output_dir                                             =
os.listdir(f"models/{self._settings.model_name}/output")
89.
90.                     if len(output_dir) > 0:
91.                         for file in output_dir:
92.
os.remove(f"models/{self._settings.model_name}/output/{file}")
93.
94.                 self._prepare_model()
95.
96.         @staticmethod
97.         def compare_models(models: list[tuple[tuple[int, list[int], int],
float, int]], prefix: str) -> None:
98.             if not os.path.exists(f"output/{prefix}"):
99.                 os.makedirs(f"output/{prefix}")
100.            else:
101.                output_dir = os.listdir(f"output/{prefix}")
102.
103.                if len(output_dir) > 0:
104.                    for file in output_dir:
105.                        os.remove(f"output/{prefix}/{file}")
106.
107.            def save_output(x: list[str], y: list[float], params: dict) ->
None:
```

```
            best_val = max(y)
            best = [i for i, val in enumerate(y) if val == best_val]

            output = " | ".join([
                f"Best-{params["y_label"]}: {best_val:.4f}",
                f"Best-Models: {", ".join([x[val] for val in best])}"
            ])

            with open(f"output/{prefix}/log.txt", "a") as file:
                file.write(f"{output}\n")

            plt.rcParams.update({"font.size": 14})
            plt.figure(figsize=(16, 9))

            plt.bar(x, y, color=["C1" if i in best else "C0" for i in
range(len(x))])

            plt.title(params["title"])
            plt.xlabel(params["x_label"])
            plt.ylabel(params["y_label"])

            plt.xticks([])
            for i, val in enumerate(y):
                plt.text(i,     val,     f"{val:.2f}",     ha="center",
va="bottom")
                plt.text(i,   0.025   *   max(y),   x[i],   ha="center",
va="bottom", color="w", fontweight="bold", rotation=90)

            plt.tight_layout()

            plt.savefig(f"output/{prefix}/{params["filename"]}")

            plt.cla()
            plt.clf()
            plt.close()

        results = []

        for (model_dims, learning_rate, num_of_epochs) in models:
            model_name                                                  =
f"{prefix}_in_{model_dims[0]}_hl_{len(model_dims[1])}_lr_{learning_rate}"

            settings = Settings(
                model_name=model_name,
                model_dims=model_dims,
                learning_rate=learning_rate,
                num_of_epochs=(num_of_epochs, 1),
                confidence_threshold=0.5,
                train_data="data/train_data.csv",
                val_data="data/val_data.csv",
                test_data="data/test_data.csv"
            )

            recommendation_system = RecommendationSystem(settings)

            last_log = recommendation_system._load_logs()[-1]

            accuracy = float(m.group(1)) if (m := re.search(r"Test-
Accuracy: ([0-9.]+)", last_log)) else 0.0
            f1_score = float(m.group(1)) if (m := re.search(r"Test-F1-
Score: ([0-9.]+)", last_log)) else 0.0

            results.append((model_name, accuracy, f1_score))

```

```
166.          name, accuracy, f1_score = zip(*results)
167.
168.          accuracy_params = {
169.              "title": "Accuracy of models",
170.              "x_label": "Models",
171.              "y_label": "Accuracy",
172.              "filename": "accuracy_comparison.png"
173.          }
174.          save_output(name, accuracy, accuracy_params)
175.
176.          f1_score_params = {
177.              "title": "F1-Score of models",
178.              "x_label": "Models",
179.              "y_label": "F1-Score",
180.              "filename": "f1_score_comparison.png"
181.          }
182.          save_output(name, f1_score, f1_score_params)
183.
184.      def  _save_train_figures(self,  data_x:  list[list[int]],  data_y:
list[list[float]], params: dict) -> None:
185.          plt.figure(figsize=(16, 9))
186.
187.          for i, (x, y) in enumerate(zip(data_x, data_y)):
188.              plt.plot(x,          y,          label=params["labels"][i],
color=params["colors"][i], linestyle="-")
189.
190.          plt.title(params["title"])
191.          plt.xlabel(params["x_label"])
192.          plt.ylabel(params["y_label"])
193.          plt.legend()
194.
195.
plt.savefig(f"models/{self._settings.model_name}/output/{params["filename"]}")
196.
197.          plt.cla()
198.          plt.clf()
199.          plt.close()
200.
201.      def    _save_test_figures(self,    figure:    RocCurveDisplay    |
ConfusionMatrixDisplay, params: dict) -> None:
202.          figure.plot()
203.
204.          if isinstance(figure, RocCurveDisplay):
205.              plt.plot([0,  1],  [0,  1],  linestyle="--",  color="gray",
label="No Skill")
206.
207.          plt.title(params["title"])
208.
209.
plt.savefig(f"models/{self._settings.model_name}/output/{params["filename"]}")
210.
211.          plt.cla()
212.          plt.clf()
213.          plt.close()
214.
215.      def _save_logs(self, output: str) -> None:
216.          with
open(f"models/{self._settings.model_name}/output/logs.txt", "a") as file:
217.              file.write(f"{output}\n")
218.
219.      def _load_logs(self) -> list[str]:
220.          with
open(f"models/{self._settings.model_name}/output/logs.txt", "r") as file:
221.              return file.readlines()
```

```
222.
223.      def _save_preprocessors(self) -> None:
224.          with
open(f"models/{self._settings.model_name}/preprocessors.pkl", "wb") as file:
225.              pickle.dump(self._preprocessors, file)
226.
227.      def _load_preprocessors(self) -> None:
228.          with
open(f"models/{self._settings.model_name}/preprocessors.pkl", "rb") as file:
229.              self._preprocessors = pickle.load(file)
230.
231.      def _save_model(self) -> None:
232.          model_state = self._model.state_dict()
233.          torch.save(model_state,
f"models/{self._settings.model_name}/model.pth")
234.
235.      def _load_model(self) -> None:
236.          model_state                                                    =
torch.load(f"models/{self._settings.model_name}/model.pth")
237.          self._model.load_state_dict(model_state)
238.          self._model.eval()
239.
240.      def _prepare_data(self, df: pd.DataFrame) -> pd.DataFrame:
241.          df = df.drop(["name", "latitude", "longitude"], axis=1)
242.
243.          lemmatizer = WordNetLemmatizer()
244.          stop_words = set(stopwords.words("english"))
245.
246.          def clean_text(text: str) -> str:
247.              text = text.lower()
248.              text = re.sub(r"[^A-Za-z0-9_]+", " ", text)
249.              text = text.strip()
250.
251.              tokens = word_tokenize(text)
252.              tokens = [
253.                  lemmatizer.lemmatize(word) for word in tokens
254.                  if word not in stop_words
255.              ]
256.
257.              if not tokens:
258.                  result = "missing information"
259.              else:
260.                  result = " ".join(tokens)
261.
262.              return result
263.
264.          df["num_of_reviews"] = np.log1p(df["num_of_reviews"])
265.          df["rating_reviews_ratio"]    =    (df["rating"]    -    3)    *
df["num_of_reviews"]
266.
267.          df.loc[:, "category"] = df["category"].apply(clean_text)
268.          df.loc[:, "context"] = df["context"].apply(clean_text)
269.          df.loc[:, "summary"] = df["summary"].apply(clean_text)
270.
271.          return df
272.
273.      def _fit_encoders(self, df: pd.DataFrame) -> None:
274.          for key in self._preprocessors["encoders"]:
275.              self._preprocessors["encoders"][key].fit(df[[key]])
276.
277.      def _extract_features(self, df: pd.DataFrame) -> np.ndarray:
278.          features = np.hstack(
279.              [
```

```
280.
self._preprocessors["encoders"][key].transform(df[[key]])
281.                     for key in self._preprocessors["encoders"]
282.                 ],
283.                 dtype=np.float32
284.             )
285.
286.             return features
287.
288.         def _fit_compressor(self, features: np.ndarray) -> None:
289.             self._preprocessors["compressor"].fit(features)
290.
291.         def _compress_features(self, features: np.ndarray) -> np.ndarray:
292.             features                                                    =
self._preprocessors["compressor"].transform(features)
293.             return features
294.
295.         def _preprocess(self, df: pd.DataFrame) -> np.ndarray:
296.             features = self._extract_features(df)
297.             preprocessed = self._compress_features(features)
298.             return preprocessed
299.
300.         def _prepare_preprocessors(self) -> None:
301.             total = math.ceil(len(pd.read_csv(self._settings.train_data)) /
self._settings.model_dims[0])
302.
303.             with alive_bar(total=total * 2, title="Preprocessing...") as
bar:
304.
305.                 with pd.read_csv(
306.                     self._settings.train_data,
307.                     dtype=self._types,
308.                     chunksize=self._settings.model_dims[0],
309.                 ) as reader:
310.                     for chunk in reader:
311.                         df = self._prepare_data(chunk)
312.                         self._fit_encoders(df)
313.
314.                         bar()
315.
316.                 with pd.read_csv(
317.                     self._settings.train_data,
318.                     dtype=self._types,
319.                     chunksize=self._settings.model_dims[0],
320.                 ) as reader:
321.                     for chunk in reader:
322.                         df = self._prepare_data(chunk)
323.                         features = self._extract_features(df)
324.                         self._fit_compressor(features)
325.
326.                         bar()
327.
328.             self._save_preprocessors()
329.
330.         def _train(self, df_train: pd.DataFrame, df_val: pd.DataFrame) ->
None:
331.             num_epochs = self._settings.num_of_epochs[0]
332.
333.             with alive_bar(total=num_epochs, title="Training...") as bar:
334.
335.                 X_train,  y_train  =  df_train.drop("recommend",  axis=1),
df_train["recommend"].to_numpy()
336.
337.                 train_dataset = LocationDataset(X_train, y_train)
```

```
            train_dataloader = LocationDataLoader(
                train_dataset,
                preprocess=self._preprocess,
                batch_size=self._settings.model_dims[0],
                shuffle=True
            )

            X_val, y_val = df_val.drop("recommend", axis=1), df_val["recommend"].to_numpy()

            val_dataset = LocationDataset(X_val, y_val)
            val_dataloader = LocationDataLoader(
                val_dataset,
                preprocess=self._preprocess,
                batch_size=self._settings.model_dims[0],
                shuffle=False
            )

            criterion = nn.BCELoss()
            optimizer = optim.Adam(self._model.parameters(), lr=self._settings.learning_rate)

            min_val_loss = np.inf

            loss_train, loss_val = [0.0 for _ in range(num_epochs)], [0.0 for _ in range(num_epochs)]
            accuracy_train, accuracy_val = [0.0 for _ in range(num_epochs)], [0.0 for _ in range(num_epochs)]

            for epoch in range(num_epochs):
                self._model.train()

                train_loss = 0.0
                y_true, y_pred = [], []

                for X_batch, y_batch in train_dataloader:
                    optimizer.zero_grad()

                    outputs = self._model(X_batch.to(self._device))

                    loss = criterion(outputs, y_batch.to(self._device))
                    loss.backward()

                    optimizer.step()

                    train_loss += loss.item()

                    true = y_batch.cpu().numpy().flatten()
                    pred = (outputs >= self._settings.confidence_threshold).float().cpu().numpy().flatten()

                    y_true.extend(true)
                    y_pred.extend(pred)

                train_loss /= len(train_dataloader)
                train_accuracy = accuracy_score(y_true, y_pred)

                self._model.eval()

                val_loss = 0.0
                y_true, y_pred = [], []

                with torch.no_grad():
                    for X_batch, y_batch in val_dataloader:
```

```
397.                         outputs = self._model(X_batch.to(self._device))
398.
399.                         loss            =            criterion(outputs,
y_batch.to(self._device))
400.
401.                         val_loss += loss.item()
402.
403.                         true = y_batch.cpu().numpy().flatten()
404.                         pred          =          (outputs          >=
self._settings.confidence_threshold).float().cpu().numpy().flatten()
405.
406.                         y_true.extend(true)
407.                         y_pred.extend(pred)
408.
409.                 val_loss /= len(val_dataloader)
410.                 val_accuracy = accuracy_score(y_true, y_pred)
411.
412.                 if val_loss < min_val_loss:
413.                     min_val_loss = val_loss
414.                     self._save_model()
415.
416.                 output = " | ".join([
417.                     f"Epoch                  [{epoch                  +
1:>{len(str(self._settings.num_of_epochs[0]))}}/{num_epochs}]",
418.                     f"Train-Loss: {train_loss:.4f}",
419.                     f"Validation-Loss: {val_loss:.4f}",
420.                     f"Train-Accuracy: {train_accuracy:.4f}",
421.                     f"Validation-Accuracy: {val_accuracy:.4f}"
422.                 ])
423.                 self._save_logs(output)
424.
425.                 loss_train[epoch], loss_val[epoch] = float(train_loss),
float(val_loss)
426.                 accuracy_train[epoch],      accuracy_val[epoch]      =
float(train_accuracy), float(val_accuracy)
427.
428.                 bar()
429.
430.         epochs = list(range(1, self._settings.num_of_epochs[0] + 1))
431.
432.         loss_params = {
433.             "title": "Training / Validation Loss",
434.             "x_label": "Epoch",
435.             "y_label": "Loss",
436.             "labels": ["Training Loss", "Validation Loss"],
437.             "colors": ["C0", "C1"],
438.             "filename": "train_val_loss.png"
439.         }
440.         self._save_train_figures([epochs,     epochs],     [loss_train,
loss_val], loss_params)
441.
442.         accuracy_params = {
443.             "title": "Training / Validation Accuracy",
444.             "x_label": "Epoch",
445.             "y_label": "Accuracy",
446.             "labels": ["Training Accuracy", "Validation Accuracy"],
447.             "colors": ["C0", "C1"],
448.             "filename": "train_val_accuracy.png"
449.         }
450.         self._save_train_figures([epochs,   epochs],   [accuracy_train,
accuracy_val], accuracy_params)
451.
452.         self._load_model()
453.
```

```
    def _test(self, df_test: pd.DataFrame) -> None:
        X, y = df_test.drop("recommend", axis=1), df_test["recommend"].to_numpy()

        test_dataset = LocationDataset(X, y)
        test_dataloader = LocationDataLoader(
            test_dataset,
            preprocess=self._preprocess,
            batch_size=self._settings.model_dims[0],
            shuffle=False
        )

        self._model.eval()

        y_true, y_pred, y_score = [], [], []

        with torch.no_grad():
            for X_batch, y_batch in test_dataloader:
                outputs = self._model(X_batch.to(self._device))

                true = y_batch.cpu().numpy().flatten()
                pred = (outputs >= self._settings.confidence_threshold).float().cpu().numpy().flatten()
                score = outputs.float().cpu().numpy().flatten()

                y_true.extend(true)
                y_pred.extend(pred)
                y_score.extend(score)

        test_accuracy = accuracy_score(y_true, y_pred)
        test_recall = recall_score(y_true, y_pred)
        test_precision = precision_score(y_true, y_pred)
        test_f1_score = f1_score(y_true, y_pred)

        output = " | ".join([
            f"Test-Accuracy: {test_accuracy:.4f}",
            f"Test-Recall: {test_recall:.4f}",
            f"Test-Precision: {test_precision:.4f}",
            f"Test-F1-Score: {test_f1_score:.4f}"
        ])
        self._save_logs(output)

        fpr, tpr, _ = roc_curve(y_true, y_score)
        roc_auc = auc(fpr, tpr)
        auc_roc_curve_figure = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)

        auc_roc_curve_params = {
            "title": "ROC Curve",
            "filename": "auc_roc_curve.png"
        }
        self._save_test_figures(auc_roc_curve_figure, auc_roc_curve_params)

        cm = confusion_matrix(y_true, y_pred)
        confusion_matrix_figure = ConfusionMatrixDisplay(confusion_matrix=cm)

        confusion_matrix_params = {
            "title": "Confusion Matrix",
            "filename": "confusion_matrix.png"
        }
        self._save_test_figures(confusion_matrix_figure, confusion_matrix_params)
```

```
512.
513.      def _prepare_model(self) -> None:
514.          df_train                                                    =
self._prepare_data(pd.read_csv(self._settings.train_data))
515.          df_val                                                      =
self._prepare_data(pd.read_csv(self._settings.val_data))
516.          self._train(df_train, df_val)
517.
518.          df_test                                                     =
self._prepare_data(pd.read_csv(self._settings.test_data))
519.          self._test(df_test)
520.
521.      def _predict(self, X: np.ndarray) -> torch.Tensor:
522.          X_tensor                    =                   torch.tensor(X,
dtype=torch.float32).to(self._device)
523.
524.          self._model.eval()
525.
526.          with torch.no_grad():
527.              return self._model(X_tensor).squeeze()
528.
529.      def _do_recommend(self, data: dict) -> bool:
530.          df = self._prepare_data(pd.DataFrame.from_records([data]))
531.          preprocessed = self._preprocess(df)
532.
533.          prediction = self._predict(preprocessed)
534.
535.          return prediction.item() >= self._settings.confidence_threshold
536.
537.      def fine_tune(self, data: dict) -> None:
538.          df = self._prepare_data(pd.DataFrame.from_records([data]))
539.
540.          num_epochs = self._settings.num_of_epochs[1]
541.
542.          X, y = df.drop("recommend", axis=1), df["recommend"].to_numpy()
543.
544.          X_train_tensor       =       torch.tensor(self._preprocess(X),
dtype=torch.float32)
545.          y_train_tensor = torch.tensor(y, dtype=torch.float32).view(-1,
1)
546.
547.          criterion = nn.BCELoss()
548.          optimizer        =        optim.Adam(self._model.parameters(),
lr=self._settings.learning_rate)
549.
550.          self._model.train()
551.
552.          while num_epochs > 0:
553.              optimizer.zero_grad()
554.
555.              outputs = self._model(X_train_tensor.to(self._device))
556.
557.              loss = criterion(outputs, y_train_tensor.to(self._device))
558.              loss.backward()
559.
560.              optimizer.step()
561.
562.              num_epochs -= 1
563.
564.          self._save_model()
565.
566.      def get_recommendations(self, data: list[dict]) -> set[int]:
567.          recommendations = set()
568.
```

```
569.          for item in data:
570.              if not self._do_recommend(item):
571.                  continue
572.
573.              recommendations.add(item["id"])
574.
575.          return recommendations
```

ДОДАТОК В

Слайди презентації

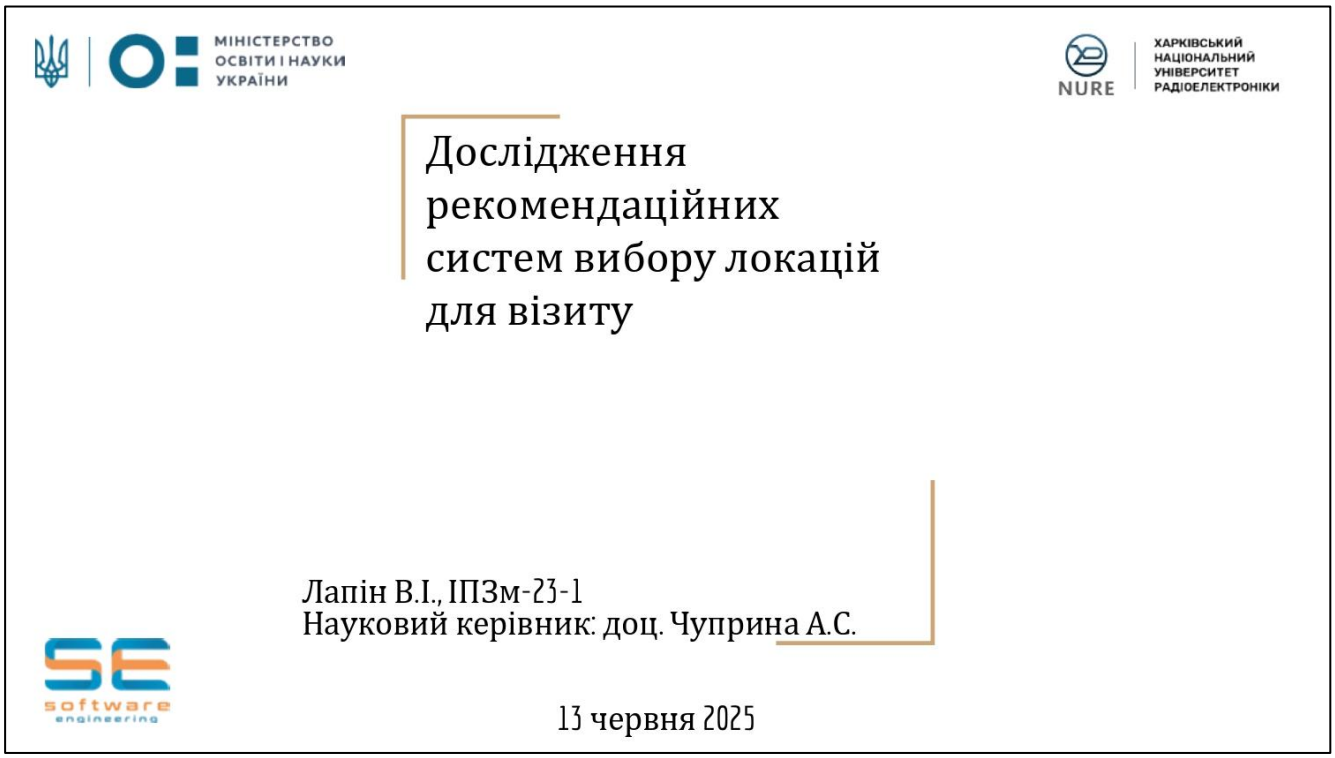


Рисунок В.1 – Титульний слайд

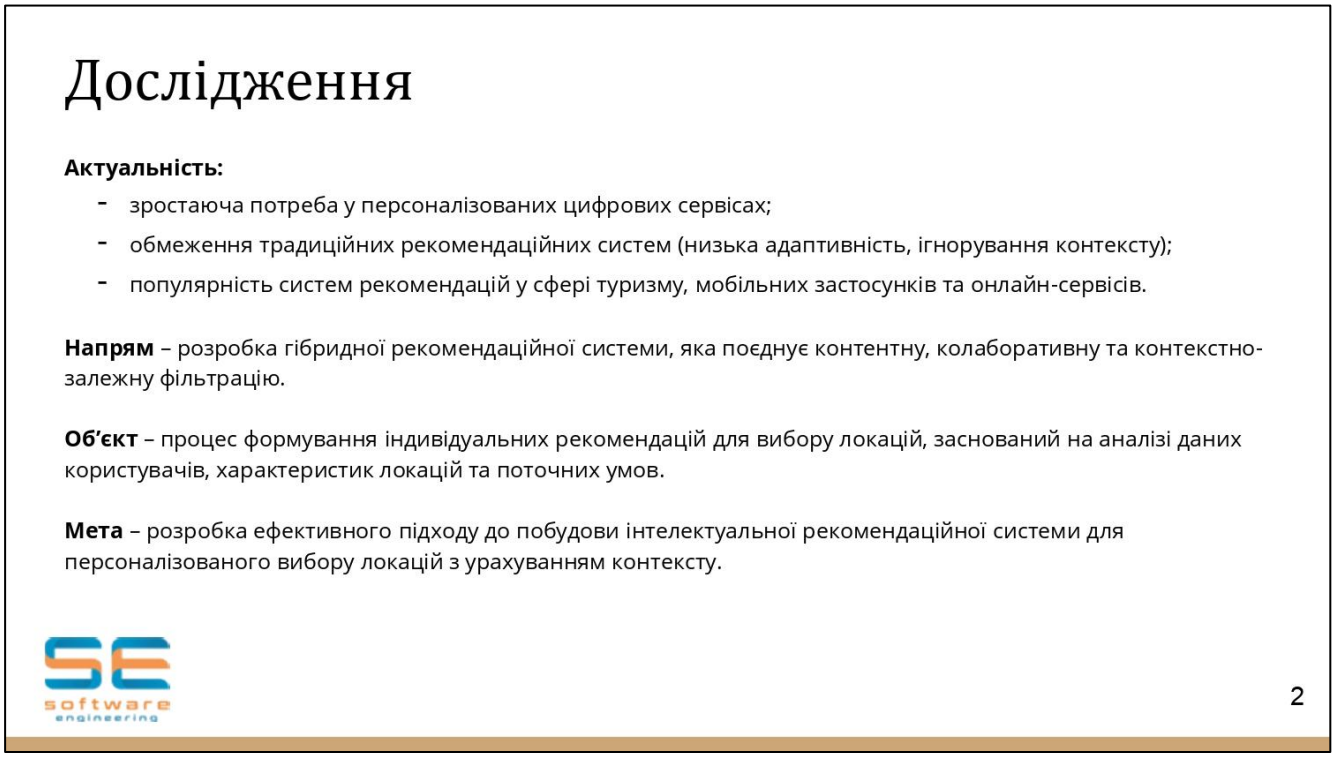


Рисунок В.2 – Слайд «Дослідження»

Огляд літератури

Основні джерела:

- A. Chaudhari, A. A. Hitham Seddig, A. Sarlan and R. Raut, "A Hybrid Recommendation System: A Review," in IEEE Access, vol. 12, pp. 157107-157126, 2024;
- K. A. Fararni, F. Nafis, B. Aghoutane, A. Yahyaouy, J. Riffi and A. Sabri, "Hybrid recommender system for tourism based on big data and AI: A conceptual framework," in Big Data Mining and Analytics, vol. 4, no. 1, pp. 47-55, March 2021;
- Z. Huang, C. Yu, J. Ni, H. Liu, C. Zeng and Y. Tang, "An Efficient Hybrid Recommendation Model With Deep Neural Networks," in IEEE Access, vol. 7, pp. 137900-137912, 2019;
- P. Yochum, L. Chang, T. Gu and M. Zhu, "Linked Open Data in Location-Based Recommendation System on Tourism Domain: A Survey," in IEEE Access, vol. 8, pp. 16409-16439, 2020;
- K. Benabbes, K. Housni, A. El Mezouary and A. Zellou, "Recommendation System Issues, Approaches and Challenges Based on User Reviews," in Journal of Web Engineering, vol. 21, no. 4, pp. 1017-1054, 2022.

3

Рисунок В.3 – Слайд «Огляд літератури»

Постановка задачі

Формулювання проблеми – існуючі рекомендаційні системи часто не враховують контекст, особисті фільтри користувача та мають проблеми з новими або малопопулярними локаціями.

Поставлені задачі:

- визначити структуру даних, необхідних для аналізу;
- обґрунтувати вибір методів і алгоритмів;
- спроєктувати архітектуру системи;
- реалізувати та інтегрувати програмні компоненти;
- провести експерименти та тестування для оцінки якості роботи системи.

Очікувані результати:

- створення гібридної рекомендаційної системи, що поєднує кілька методів фільтрації;
- забезпечення персоналізованих рекомендацій з урахуванням поточного контексту;
- підвищення точності, адаптивності та зручності використання системи.

4

Рисунок В.4 – Слайд «Постановка задачі»

Методологія

Використані методи дослідження:

- теоретичний аналіз наукових джерел;
- системний підхід до формалізації задач;
- математичне моделювання процесу рекомендацій;
- експериментальне моделювання;
- комп'ютерне моделювання з використанням машинного навчання.

Інструменти та технології:

- PyTorch – реалізація моделі;
- pandas, NumPy – обробка даних;
- scikit-learn – векторизація, PCA, метрики;
- matplotlib, seaborn – візуалізація результатів;
- Google Local Dataset – джерело даних.

NumPy Google pandas scikit learn seaborn PyTorch matplotlib

SE software engineering

5

Рисунок В.5 – Слайд «Методологія»

Архітектура системи для проведення експериментального дослідження

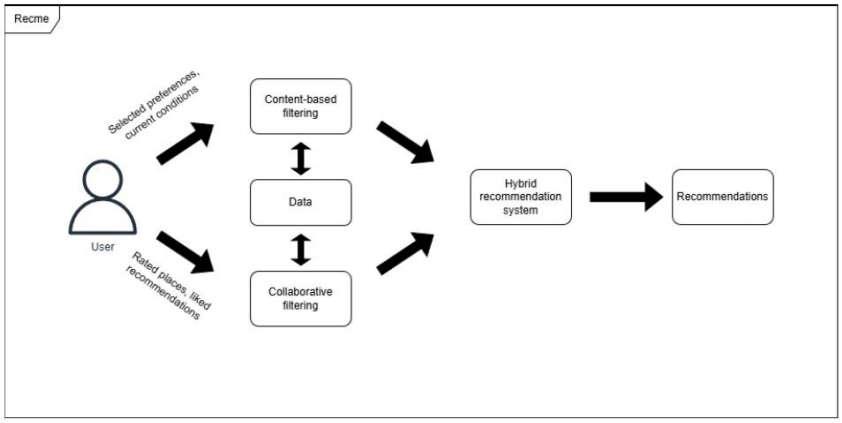


Схема роботи гібридної рекомендаційної системи

SE software engineering

6

Рисунок В.6 – Слайд «Архітектура системи для проведення експериментального дослідження»

Опис програмного забезпечення, що було використано у дослідженні

Процес розробки:

- проєктування клієнт-серверної архітектури;
- реалізація окремих модулів (інтерфейс, API, база даних);
- тестування функціональності та точності моделі;
- інтеграція компонентів у єдину систему.

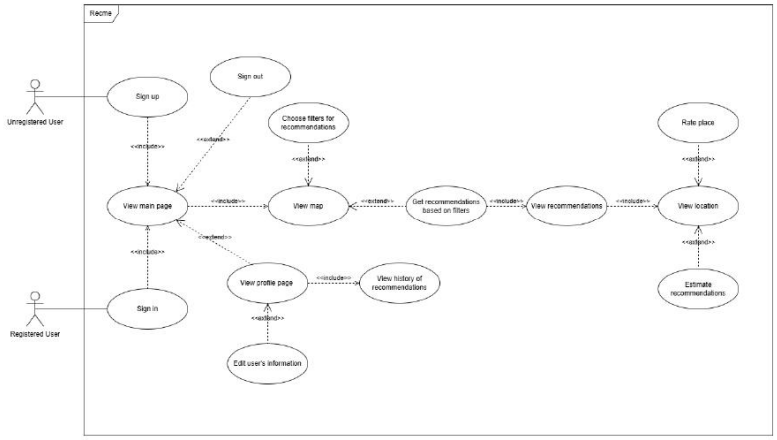
Використані технології:

- клієнтська частина – TypeScript, React;
- серверна частина – Python, Django;
- бази даних – PostgreSQL, SQLite;
- редактор коду – Visual Studio Code;
- середовище розробки – GIT, Linux.

7

Рисунок В.7 – Слайд «Опис програмного забезпечення, що було використано у дослідженні»

Моделювання програмного забезпечення

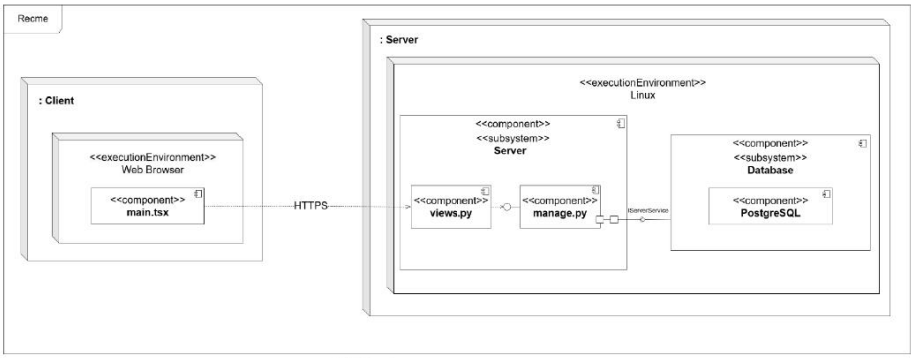


UML діаграма прецедентів користувачів

8

Рисунок В.8 – Слайд «Моделювання програмного забезпечення»

Архітектура програмного забезпечення

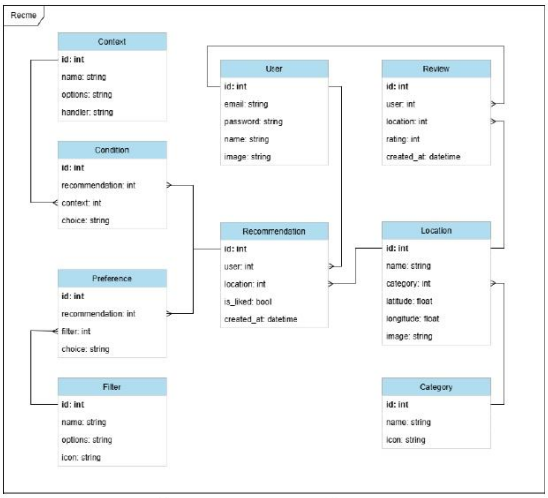


UML діаграма розгортання

9

Рисунок В.9 – Слайд «Архітектура програмного забезпечення»

Структура зберігання даних



ER-діаграма структури бази даних

10

Рисунок В.10 – Слайд «Структура зберігання даних»

Зміст проведених експериментів

Методи:

- експериментальне моделювання;
- класифікація результатів за допомогою нейронної мережі;
- контрольоване тестування на заздалегідь підготовлених сценаріях.

Вхідні дані:

- категорія локації;
- рейтинг;
- кількість відгуків;
- фільтри користувача;
- контекстуальні фактори.

Критерії ефективності:

- Accuracy;
- F1-score;
- AUC-ROC curve;
- Confusion matrix.

Послідовність:

Побудова тестових сценаріїв → формування вхідних даних → генерація рекомендацій → аналіз відповідності очікуваним результатам.

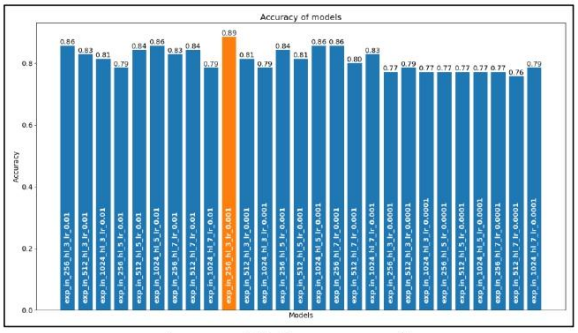
Вимірювання:

Автоматичне порівняння очікуваної та фактичної класифікації, побудова метрик для кожного сценарію та візуалізація результатів у вигляді графіків.

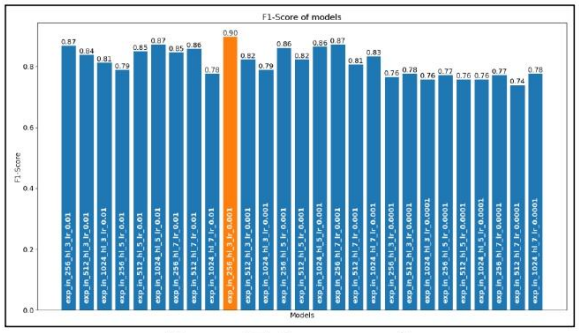
11

Рисунок В.11 – Слайд «Зміст проведених експериментів»

Результати експериментів



Accuracy 27 різних моделей

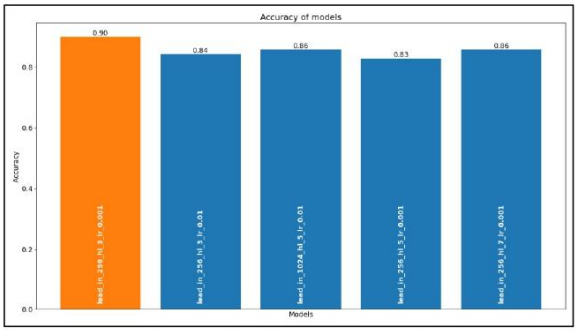


F1-score 27 різних моделей

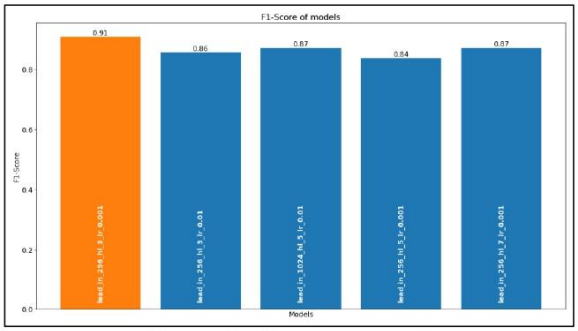
12

Рисунок В.12 – Слайд «Результати експериментів» (перша частина)

Результати експериментів



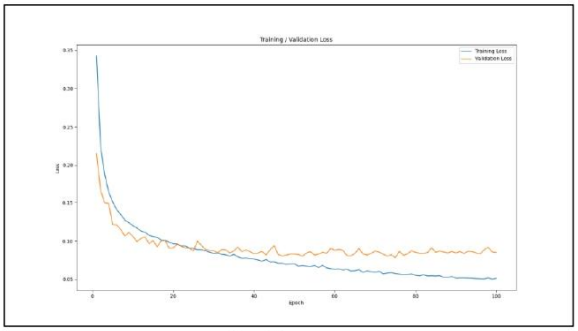
Accuracy 5 найкращих моделей



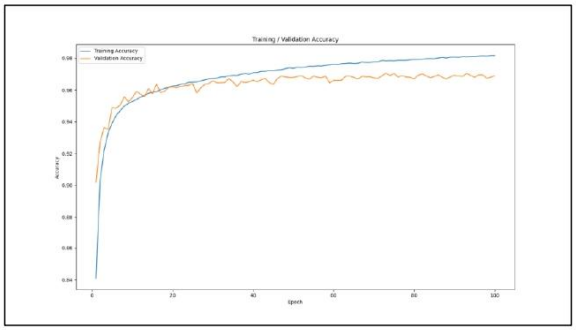
F1-score 5 найкращих моделей

Рисунок В.13 – Слайд «Результати експериментів» (друга частина)

Аналіз отриманих результатів



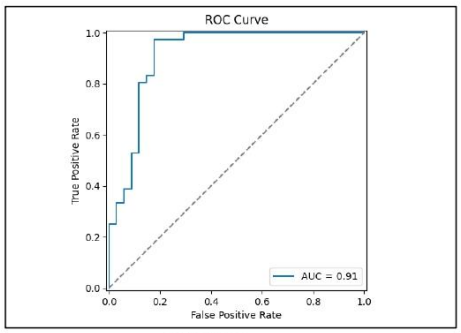
Графік втрат



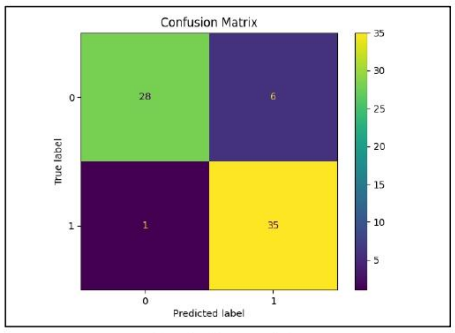
Графік точності

Рисунок В.14 – Слайд «Аналіз отриманих результатів» (перша частина)

Аналіз отриманих результатів



AUC-ROC curve



Confusion matrix

15

Рисунок В.15 – Слайд «Аналіз отриманих результатів» (друга частина)

Публікація результатів

Стаття на тему:
«A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit»

Була представлена на:
9th Open International Conference "Electrical, Electronic and Information Sciences" eStream 2025, формат IEEE.


A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit

A Hybrid Approach in Developing a Recommendation System for Personalized Selection of Locations for a Visit

16

Рисунок В.16 – Слайд «Публікація результатів»

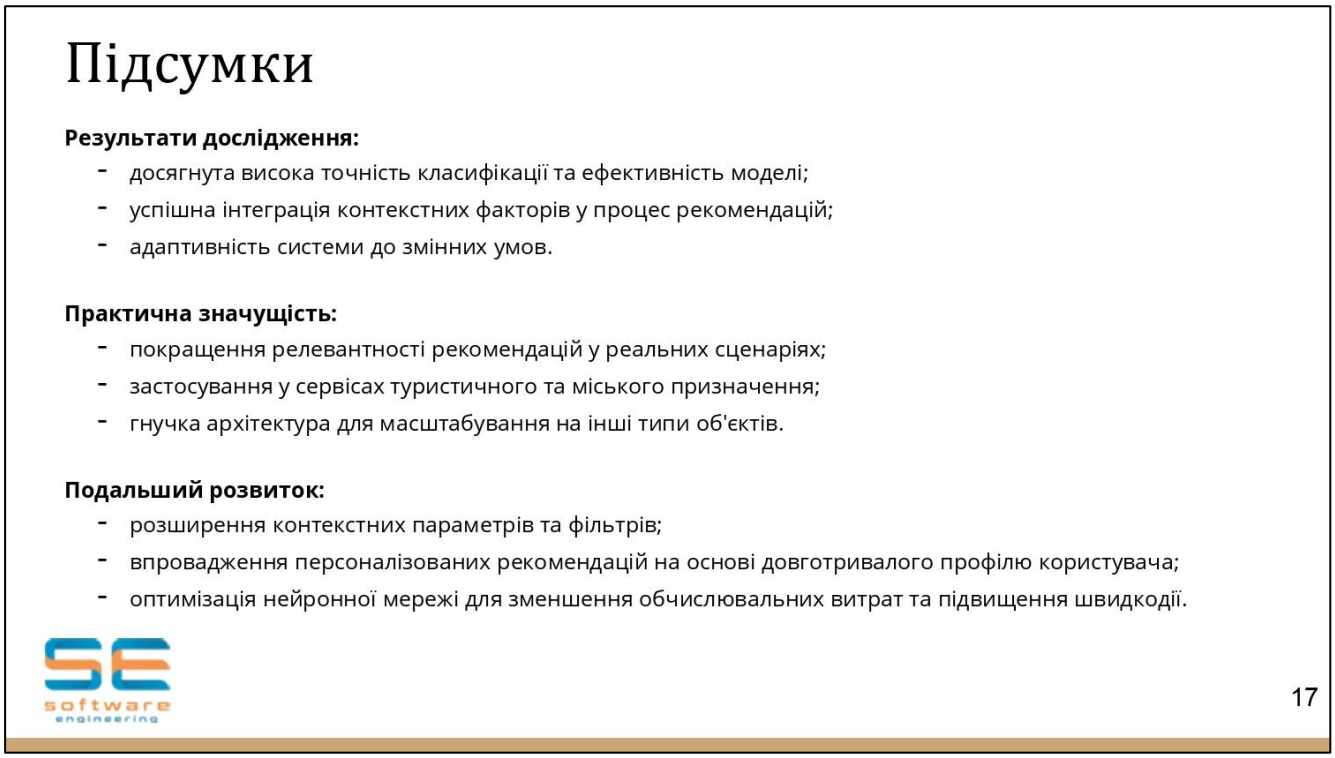
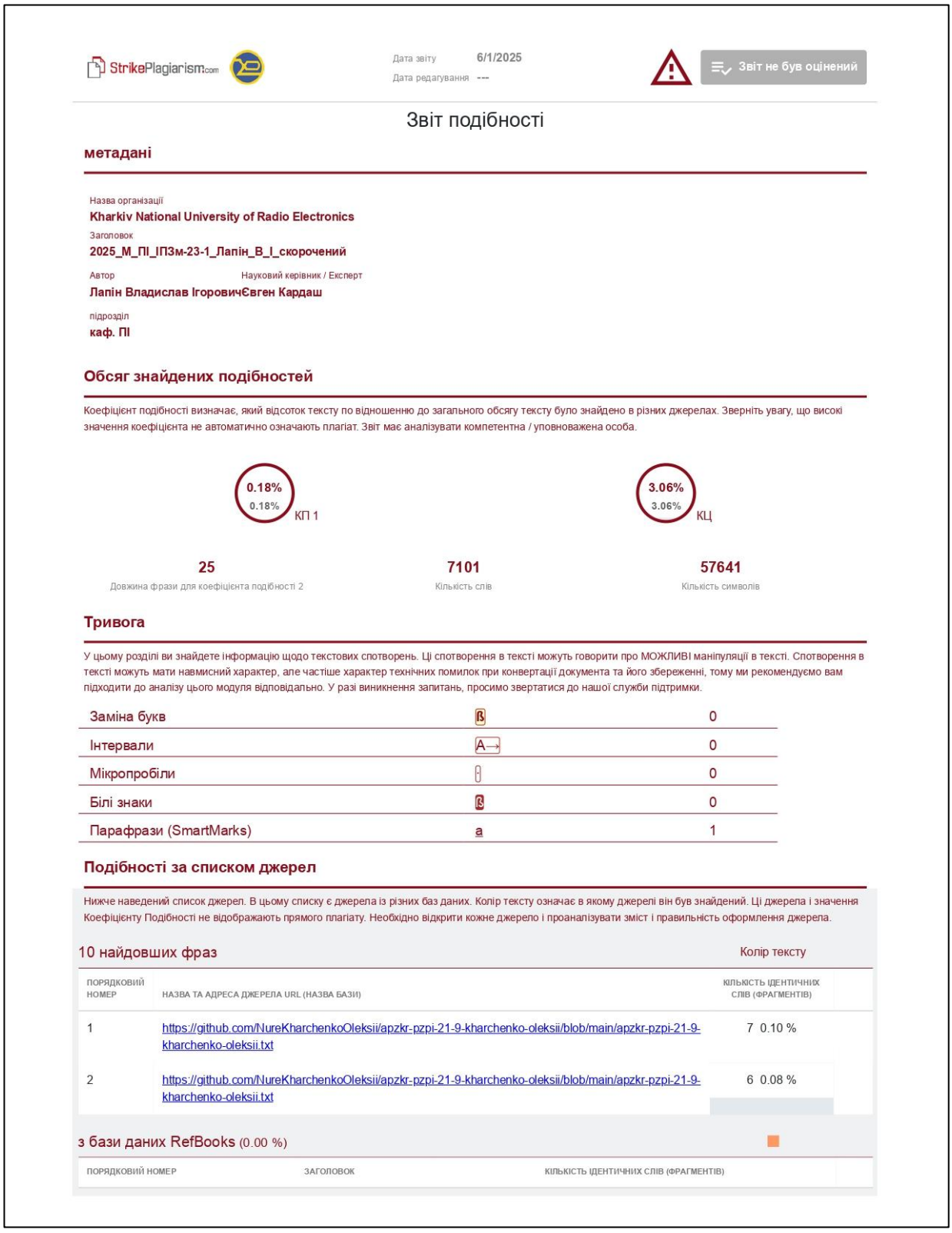


Рисунок В.17 – Слайд «Підсумки»

ДОДАТОК Г

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism.com

Дата звіту 6/1/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_М_ПІ_ІПЗм-23-1_Лапін_В_І_скорочений
Автор
Лапін Владислав ІгоровичЄвген Кардаш
Науковий керівник / Експерт
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.18% 0.18% КП 1

3.06% 3.06% КЦ

25	7101	57641
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)		1

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://github.com/NureKharchenkoOleksii/apzkr-pzpi-21-9-kharchenko-oleksii/blob/main/apzkr-pzpi-21-9-kharchenko-oleksii.txt	7 0.10 %
2	https://github.com/NureKharchenkoOleksii/apzkr-pzpi-21-9-kharchenko-oleksii/blob/main/apzkr-pzpi-21-9-kharchenko-oleksii.txt	6 0.08 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

Рисунок Г.1 – Результат перевірки роботи на плагіат

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-1 (група)

Владислав ЛАПІН
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

6/3/2025

Рисунок Д.1 – Експертний висновок результатів перевірки на відповідність оформлення вимогам ДСТУ 3008:2015