https://doi.org/10.31891/2307-5732-2025-351-15
УДК 004.8

ГАВРИЛОВ МИКОЛА
Тернопільський національний технічний університет ім. Івана Пулюя
https://orcid.org/0009-0003-3199-129X
e-mail: gavrilovnikolay1999@gmail.com

АНАЛІЗ ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІДЕНТИФІКАЦІЇ ШКІДЛИВИХ APK-ФАЙЛІВ У СЕРЕДОВИЩІ ANDROID

В роботі наведено результати досліджень щодо застосування штучного інтелекту для ідентифікації шкідливих APK-файлів у середовищі Android. Для проведення аналізу було використано XGBClassifier та RandomForestClassifier, а також перетворення тестових даних у векторні представлення за допомогою моделі text-embedding-3-small від OpenAI.

Ключові слова: штучний інтелект, шкідливі APK-файли, Android, XGBClassifier, RandomForestClassifier, безпека мобільних застосунків.

HAVRYLOV MYKOLA
Ternopil Ivan Puluj National Technical University

ANALYSIS OF THE APPLICATION OF ARTIFICIAL INTELLIGENCE FOR THE IDENTIFICATION OF MALICIOUS APK FILES IN THE ANDROID ENVIRONMENT

This article presents a comprehensive examination of how artificial intelligence (AI) can be employed to detect malicious APK files within the Android environment. The research investigates machine learning algorithms, specifically XGBClassifier and RandomForestClassifier, which have demonstrated robust capabilities in analyzing diverse datasets and identifying harmful content. By leveraging these algorithms, the study aims to streamline the process of classifying APK files according to their threat level and thereby improve the overall security of mobile ecosystems. A crucial aspect of the proposed method involves the transformation of textual data into meaningful embeddings using the OpenAI text-embedding-3-small model, enabling the algorithms to capture semantic information and enhance the accuracy of threat detection.

The proposed workflow addresses several key challenges in the field of mobile security. Firstly, the feature extraction process is designed to incorporate code analysis and metadata inspection to ensure that all relevant signals are captured prior to training. Secondly, the use of ensemble methods underscores the importance of combining different machine learning techniques for improved prediction performance. By comparing the results of XGBClassifier and RandomForestClassifier, the research provides insights into the relative strengths of gradient boosting versus bagging approaches. Moreover, experimental evaluations are conducted on both benign and malicious APK files, drawn from open-source malware database, to validate the effectiveness and reliability of the classification pipeline. The outcomes offer clear indications that intelligent, AI-driven methodologies can significantly reduce false positives and false negatives, paving the way for more efficient, large-scale security solutions in Android environments. Beyond the empirical evaluation, the article delves into practical considerations for deploying these models in real-world applications, including scalability, model interpretability, and resource constraints. Finally, the research highlights future directions, such as exploring novel embedding techniques, refining hyperparameters for enhanced performance, and extending the training pipeline to encompass evolving malware patterns. These findings underscore the potential of AI-based solutions to play a pivotal role in safeguarding millions of Android users against ever-emerging cyber threats.

Keywords: artificial intelligence, malicious APK files, Android, XGBClassifier, RandomForestClassifier, mobile application security.

Стаття надійшла до редакції / Received 11.04.2025
Прийнята до друку / Accepted 26.04.2025

Постановка проблеми та її рішення

У сучасному цифровому середовищі мобільні пристрої стали невід'ємною частиною повсякденного життя і найбільш поширеною платформою для них є операційна система Android. Разом із постійним зростанням кількості Android-застосунків збільшується й ризик поширення шкідливих APK-файлів, здатних не лише шкодити пристроям та викрадати конфіденційну інформацію, але й порушувати стабільність усієї мобільної екосистеми. Традиційні методи антивірусного аналізу часто виявляються недостатньо ефективними для протидії все більш витонченим загрозам, які постійно адаптуються до нових механізмів захисту та маскуються під легітимні програми. У зв'язку з цим виникає гостра потреба у використанні новітніх підходів, зокрема методів штучного інтелекту (ШІ), здатних ефективніше ідентифікувати шкідливі APK-файли за їхніми поведінковими та семантичними ознаками.

Актуальність роботи зумовлена різними факторами: збільшення кількості кібератак та безупинна еволюція зловмисного програмного забезпечення вимагають оперативної розробки та впровадження нових рішень, що базуються на аналізі великих обсягів даних і здатні прогнозувати потенційні загрози у реальному часі. ШІ-алгоритми, зокрема XGBClassifier та RandomForestClassifier, уже продемонстрували високу точність у виявленні складних залежностей у даних, включно з біометричними, текстовими та мережевими ознаками. Їхнє застосування в Android-середовищі дозволяє не лише підвищити якість класифікації APK-файлів на «безпечні» та «шкідливі», а й знизити рівень помилкових спрацьовувань. Актуальність також зумовлена тим, що сучасні загрози все частіше використовують методи соціальної інженерії та маскування під легітимні застосунки, що ускладнює

ручне визначення шкідливої поведінки. Такий стан речей потребує автоматизованих систем, здатних швидко навчатися на нових загрозах, виявляти приховані закономірності та ефективно реагувати на появу невідомих шкідливих APK-файлів.

Метою даного дослідження є розробка та експериментальна перевірка методів виявлення шкідливих APK-файлів у середовищі Android із використанням алгоритмів машинного навчання XGBClassifier і RandomForestClassifier. Особливу увагу планується приділити трансформації текстових даних у вектори за допомогою моделі text-embedding-3-small від OpenAI, що має на меті покращити семантичне розпізнавання шкідливих залежностей у коді або метаданих APK-файлів.

Об'єкт дослідження - процес виявлення та класифікації шкідливих APK-файлів у середовищі Android.

Предмет дослідження - методи та алгоритми машинного навчання, а також засоби побудови векторів, які дозволяють підвищити точність і швидкість виявлення шкідливих APK-файлів.

Аналіз існуючих систем та методів

На сьогоднішній день існує низка підходів до виявлення шкідливого програмного забезпечення в середовищі Android, що базуються на статичному та динамічному аналізі вихідного коду, скануванні мережевої активності чи поведінкових характеристик застосунків. Традиційні антивірусні рішення проводять переважно сигнатурний аналіз, порівнюючи вміст APK-файлу зі списками уже відомих загроз [1]. Проте цей підхід має суттєві обмеження, оскільки зловмисники дедалі частіше застосовують методи обфускації та поліморфізму, що ускладнює виявлення нових загроз.

Задля підвищення ефективності, дослідники активно запроваджують методи машинного навчання та глибинних нейронних мереж [2]. Наприклад, у деяких роботах пропонується створення спеціалізованих класифікаторів, навчених на векторах ознак APK (маніфест-файл, дозволи, використані бібліотеки тощо). Інші системи базуються на обробці поведінкових логів запуску програм у віртуальному середовищі для виявлення прихованої шкідливої активності [3]. До переваг таких підходів належить здатність алгоритмів виявляти невідомі або модифіковані варіанти шкідливих застосунків.

Особливий інтерес викликають гібридні системи, що об'єднують статичний і динамічний аналіз з елементами штучного інтелекту [4]. У них застосовуються алгоритми (наприклад, RandomForestClassifier, XGBClassifier), а також техніки створення векторизованих представлень коду (embedding-моделі) для підвищення точності розпізнавання потенційно шкідливих дій. Однак повноцінне впровадження таких методів вимагає надійної та масштабованої інфраструктури, здатної обробляти великі обсяги даних у реальному часі.

У цілому, аналіз сучасних рішень свідчить про високу релевантність використання алгоритмів штучного інтелекту в завданні класифікації шкідливих APK-файлів. Разом з тим, розвиток загроз і технік обходу захисту стимулює дослідників до постійного вдосконалення підходів: від поліпшеної фільтрації даних та комбінування різних алгоритмів до впровадження нових архітектур нейронних мереж і механізмів розподіленого аналізу.

Аналіз обраного набору даних

Для подальшого проведення дослідження було обрано набір даних Androbin з платформи Kaggle. Набір даних містить статичні атрибути, витягнуті з Android-додатків за допомогою AndroZoo APK Analysis API. Таким чином, цей набір даних є підмножиною звітів Android-додатків, наданих AndroZoo, що складається з восьми текстових атрибутів (імена ресурсів, класи і методи вихідного коду, дозволи маніфесту тощо) і містить 213 928 безпечних і 70 340 шкідливих додатків. Нижче буде показано детальніший огляд цього набору даних та його характеристик. Оскільки дані дуже великого розміру (8.78 GB у стисненому вигляді), тому для дослідження було взято 11000 записів за допомогою наступного коду на рис. 1 та використання ресурсів Google Colab.

```python
class ParquetDataLoader:
    def __init__(
        self,
        input_file: str,
        columns: list[str],
        output_file_path: str,
        use_threads: bool = True,
        max_batches: int = 10,
        batch_size: int = 2000,
        *args,
        **kwargs
    ):
        self.input_file = input_file
        self.columns = columns
        self.output_file_path = output_file_path
        self.use_threads = use_threads
        self.max_batches = max_batches
        self.batch_size = batch_size
        self.parquets = []

    def process(self):
        self.load_data()
        self.save_to_csv()

    def load_data(self):
        parquet_file = pq.ParquetFile(self.input_file)
        for idx, batch in enumerate(
            parquet_file.iter_batches(
                batch_size=self.batch_size,
                columns=self.columns,
                use_threads=self.use_threads,
            )
        ):
            print("RecordBatch", idx)
            self.parquets.append(batch.to_pandas())
            if idx >= self.max_batches - 1:
                break

    def save_to_csv(self):
        data = pd.concat(self.parquets, ignore_index=True)
        data.to_csv(path_or_buf=self.output_file_path, index=False)
```

Рис. 1. Клас для вивантаження частини записів з набору даних

Обрані характеристики APK-файлів для подальшого аналізу:

1. **resource.entry**: містить список ресурсів, які використовуються додатком. Наприклад, це можуть бути імена кнопок, іконок, текстів, які відображаються в інтерфейсі.
2. **source.class.package**: вказує на пакети класів, які використовуються в додатку. Це програмні модулі, що імпортуються для забезпечення певного функціоналу (наприклад, робота з картою, рекламою, підтримка анімацій та інші можливості).
3. **manifest.permission**: список дозволів, які запитує додаток у файлі AndroidManifest.xml. Наприклад, доступ до інтернету, місцезнаходження, стану мережі тощо.
4. **manifest.activity**: список активностей, задекларованих у маніфесті додатка. Кожна активність представляє окремий екран або функціонал програми (наприклад, відображення реклами, відео, браузера тощо).
5. **manifest.action**: містить дії (intent actions), які підтримує додаток. Наприклад, android.intent.action.VIEW означає можливість відкривати URL-адреси.
6. **manifest.category**: список категорій (intent categories), що описують тип дій чи екранів додатка. Наприклад, android.intent.category.LAUNCHER означає, що активність може бути головною для запуску програми.
7. **manifest.feature**: вказує на додаткові функції, які додаток підтримує (наприклад, камера, сенсори). Якщо даних про функції немає, то це [None].
8. **source.method.name**: містить назви методів, які використовуються в коді додатка. Це конкретні функції, що виконують певну задачу, наприклад, getactivityconfidence, settax.
9. **label**: мітка, яка використовується для класифікації. У нашому випадку значення 0 вказує на те, що цей додаток вважається "не шкідливим" та 1 "шкідливий".
10. **meta.vt.date**: файл вперше було просканировано через систему VirusTotal.

Далі буде показано розподіл шкідливих додатків та їхніх характеристик відповідно до колонки label.

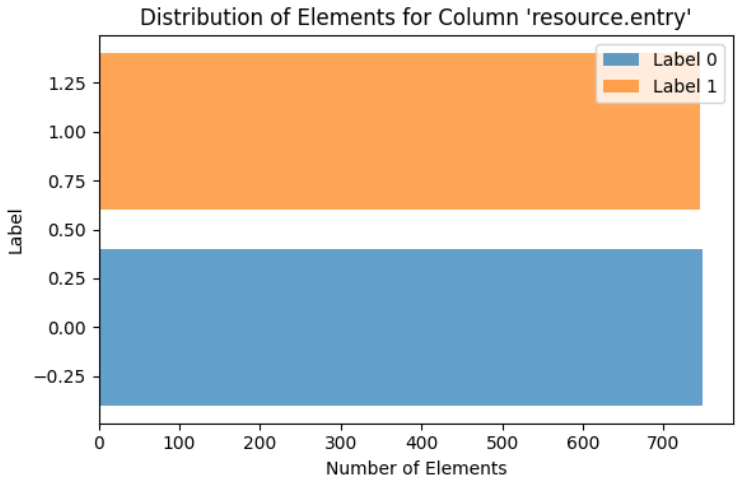

Рис. 2. Розподіл APK-файлів відносно колонки resource.entry

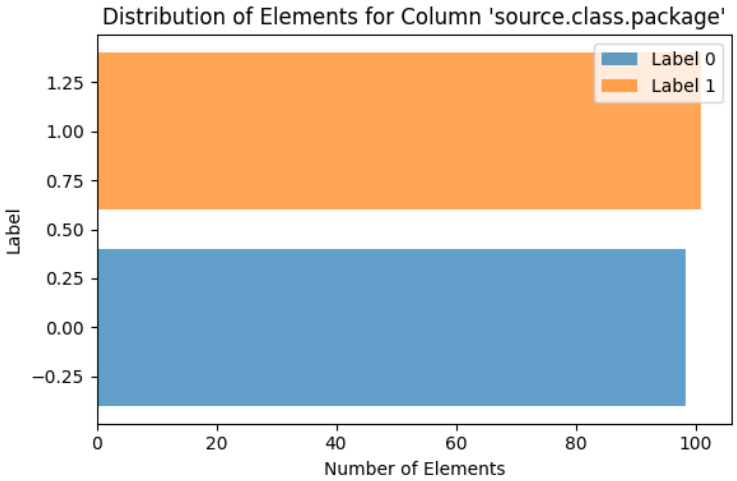

Рис. 3. Розподіл APK-файлів відносно колонки source.class.package

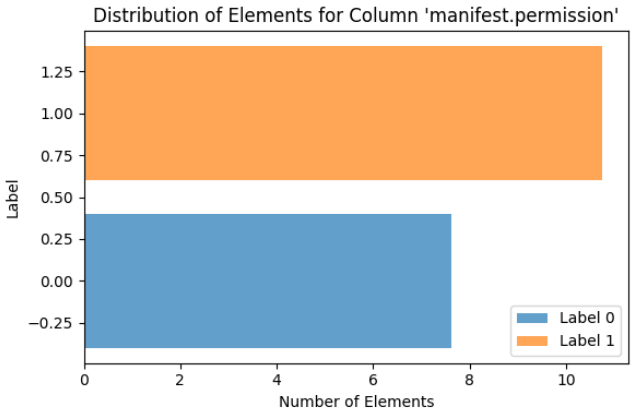

Рис. 4. Розподіл APK-файлів відносно колонки manifest.permission

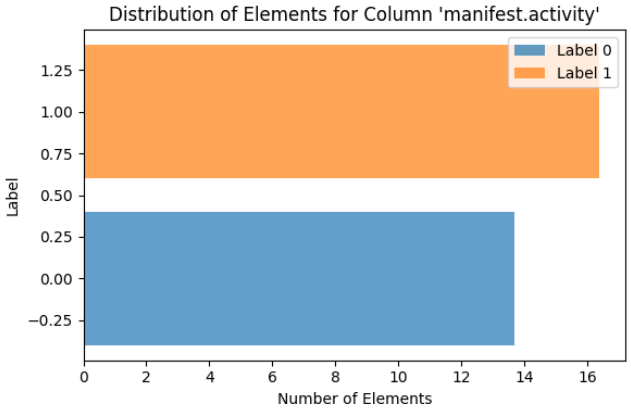

Рис. 5. Розподіл APK-файлів відносно колонки manifest.activity

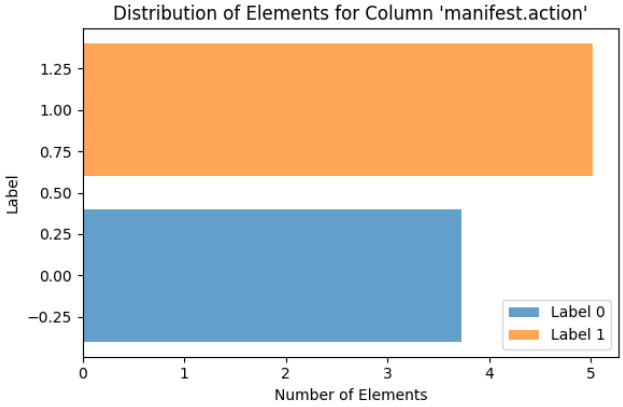

Рис. 6. Розподіл APK-файлів відносно колонки manifest.category

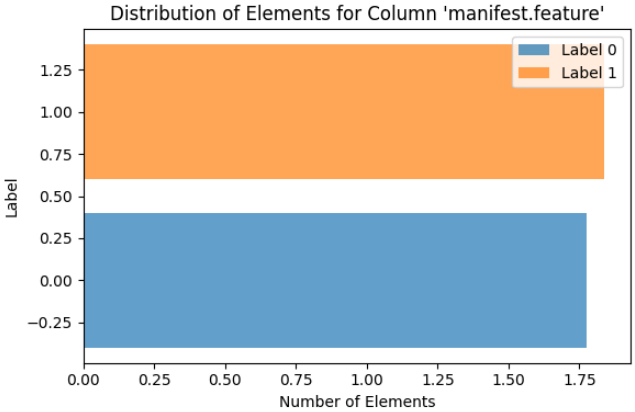

Рис. 7. Розподіл APK-файлів відносно колонки manifest.feature

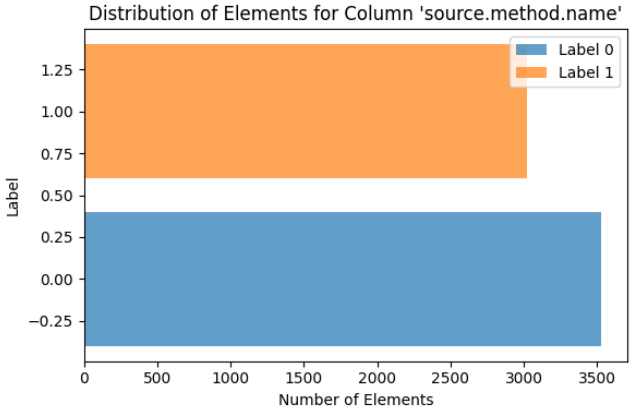

Рис. 8. Розподіл APK-файлів відносно source.method.name

З вище показаних графіків на рис. 2-8 можна зробити висновок, що група manifest в середньому має більше елементів для шкідливих APK-файлів у порівнянні з безпечними застосунками. Подальше дослідження буде фокусуватися на роботі з цими характеристиками групи manifest.

Конвертування текстового представлення у векторне

Було проведено виокремлення характеристик групи manifest та з'єднано в єдине текстове представлення для подальшого перетворення у вектори, рис. 9-11.

```python
def concatenate_columns(
    dataframe: pd.DataFrame, exclude_columns: list[str]
) -> pd.Series:
    """
    Concatenate all columns of each row into a single string, excluding specified columns.
    """
    return dataframe.drop(columns=exclude_columns, errors="ignore").apply(
        lambda row: ", ".join(
            " ".join(map(str, val)) if isinstance(val, list) else str(val)
            for val in row
            if val is not None
        ),
        axis=1,
    )


exclude_columns = [
    "label",
    "meta.vt.date",
    "resource.entry",
    "source.class.package",
    "source.method.name",
]

data["full_string"] = concatenate_columns(data, exclude_columns)
```

Рис. 9. Перетворення характеристик групи в єдине текстове представлення

```python
def compute_embeddings(texts: pd.Series, batch_size: int = 1) -> list:
    """Generates embeddings for a list of texts using OpenAI's API."""
    embeddings = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i : i + batch_size].tolist()
        try:
            response = client.embeddings.create(
                input=batch, model="text-embedding-3-small"
            )
            batch_embeddings = [item.embedding for item in response.data]
            embeddings.extend(batch_embeddings)
        except Exception as e:
            print(f"Error generating embeddings for batch {i}: {e}")
    return embeddings
```

Рис. 10. Перетворення тексту у векторне представлення

```python
data[["openai_embedding", "label"]].to_csv("apk_data_with_embeddings.csv", index=False)
```

Рис. 11. Збереження векторів та маркерів в окремий файл

Тренування моделей

XGBClassifier

Проведено тренування класифікатора на наборі даних з крос-валідацією та різними наборами параметрів, рис 12. Представлено результати тренування на рис. 13-14.

```python
X = pd.DataFrame(data_with_embeddings["openai_embedding"].to_list())
y = data_with_embeddings["label"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [4, 6],
    "learning_rate": [0.01, 0.1],
}

grid_search = GridSearchCV(
    estimator=XGBClassifier(random_state=42),
    param_grid=param_grid,
    scoring="accuracy",
    cv=3,
    verbose=1,
)

grid_search.fit(X_train, y_train)

best_xgb = grid_search.best_estimator_

y_pred_best_xgb = best_xgb.predict(X_test)
```

Рис. 12. Тренування XGBClassifier

	precision	recall	f1-score	support
0	0.84	0.93	0.88	1725
1	0.86	0.71	0.77	1025
accuracy			0.85	2750
macro avg	0.85	0.82	0.83	2750
weighted avg	0.85	0.85	0.84	2750

Рис. 13. Класифікаційний репорт тренування XGBClassifier

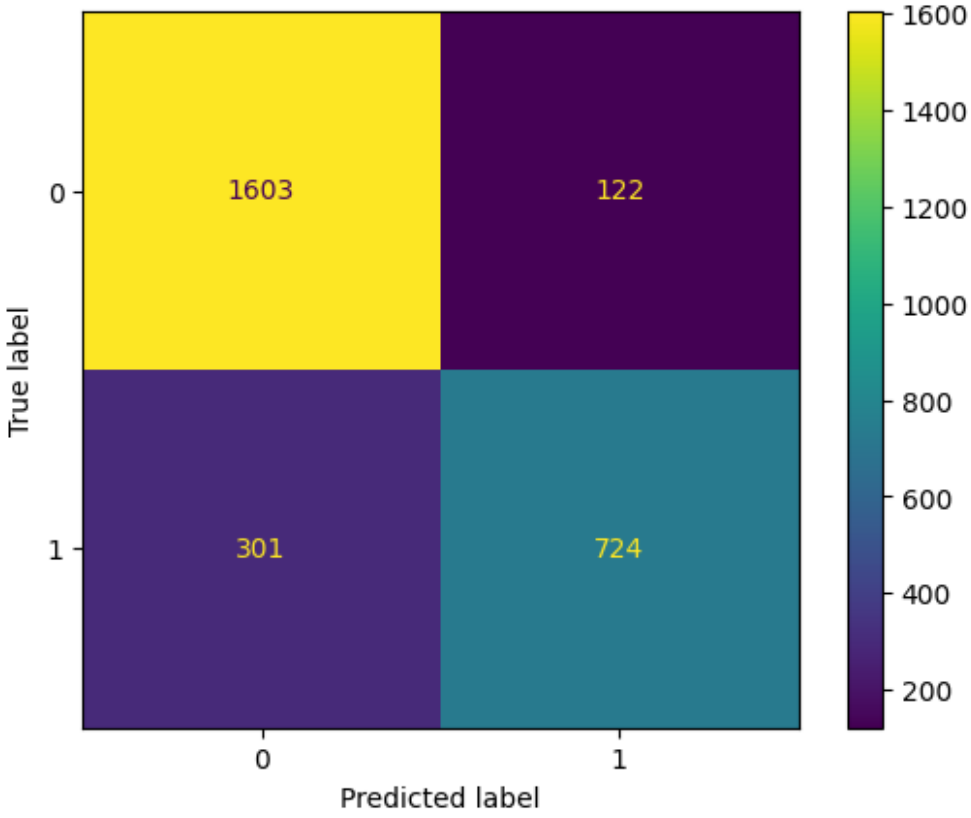

Рис. 14. Confusion matrix результатів класифікатора XGBClassifier

RandomForestClassifier

Проведено тренування класифікатора на наборі даних, рис. 15. Представлено результати тренування на рис. 16-17.

```
# Split the dataset into train (75%) and test (25%) sets
X = pd.DataFrame(data_with_embeddings["openai_embedding"].to_list())
y = data_with_embeddings["label"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# Train a simple classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict class for the test set
y_pred = clf.predict(X_test)
```

Рис. 15. Тренування RandomForestClassifier

	precision	recall	f1-score	support
0	0.81	0.94	0.87	1725
1	0.87	0.64	0.73	1025
accuracy			0.83	2750
macro avg	0.84	0.79	0.80	2750
weighted avg	0.83	0.83	0.82	2750

Рис. 16. Класифікаційний репорт тренування RandomForestClassifier

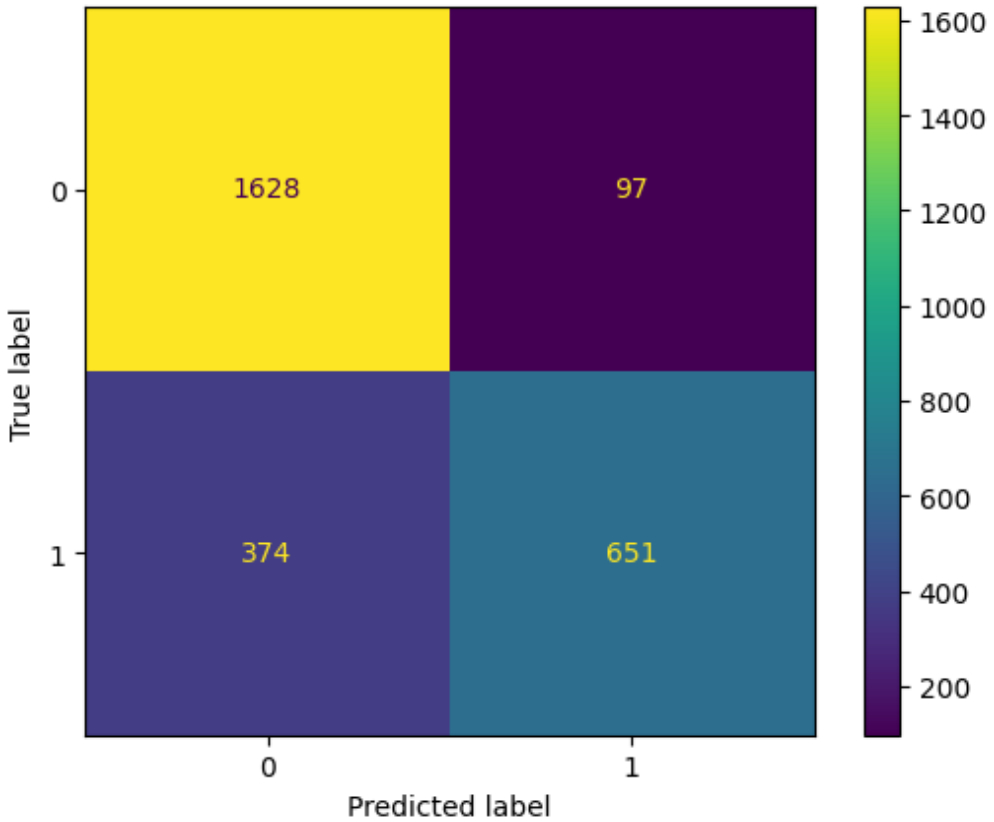

Рис. 17. Confusion matrix результатів класифікатора RandomForestClassifier

Висновки з даного дослідження і перспективи подальшого розвитку в даному напрямі

Експериментальним шляхом досліджено можливість використання штучного інтелекту для виявлення шкідливих APK-файлів у середовищі Android. Найкращі результати показав класифікатор XGBClassifier. Правильно виявлено 724 шкідливі файли та 301 не виявлено. Помилково ідентифіковано 122 доброякісні програми. В той час як класифікатор RandomForestClassifier правильно ідентифікував 651 шкідливий файл та 374 не виявлено, а 97 помилково ідентифіковано як недоброзичливу програму.

Для покращення результатів можливе використання кількох варіантів:

- Використання кращої моделі для перетворення текстового представлення у векторне, наприклад, text-embedding-3-large від OpenAI або тренувати власний трансформер.
- Збільшення кількості записів для тренування.
- Комбінувати різні набори характеристик.
- Розробити нову Deep Learning модель, яка зможе краще виявляти шаблони поведінки таких програм.

Література

1. Arp D. Drebin: Effective and explainable detection of Android malware in your pocket / D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck // Network and Distributed System Security Symposium (NDSS). – 2014.

2. Sahs J. A machine learning approach to Android malware detection / J. Sahs, L. Khan // 2012 European Intelligence and Security Informatics Conference (EISIC). – 2012. – P. 141–147. – ISBN 978-1-4673-2358-1.

3. Yan L.K. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis / L.K. Yan, H. Yin // Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). – 2012. – P. 569–584. – ISBN 978-931971-95-9.

4. Bokolo B. Hybrid Analysis Based Cross Inspection Framework for Android Malware Detection / B. Bokolo, G. Sur, Q. Liu, F. Yuan, F. Liang // 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA). – 2022. – ISBN 978-1-6654-8351-3.

References

1. Arp D. Drebin: Effective and explainable detection of Android malware in your pocket / D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck // Network and Distributed System Security Symposium (NDSS). – 2014.

2. Sahs J. A machine learning approach to Android malware detection / J. Sahs, L. Khan // 2012 European Intelligence and Security Informatics Conference (EISIC). – 2012. – P. 141–147. – ISBN 978-1-4673-2358-1.

3. Yan L.K. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis / L.K. Yan, H. Yin // Proceedings of the 21st USENIX Security Symposium (USENIX Security 12). – 2012. – P. 569–584. – ISBN 978-931971-95-9.

4. Bokolo B. Hybrid Analysis Based Cross Inspection Framework for Android Malware Detection / B. Bokolo, G. Sur, Q. Liu, F. Yuan, F. Liang // 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA). – 2022. – ISBN 978-1-6654-8351-3.