УДК 004.93:: 355.58 DOi 10.30837/ bi.2025.2(103).17

І. П. Гамаюн[1], Г. А. Плєхова[2], М. В. Костікова[3], Д. О. Плєхов[4], Р. Б. Багмут[5]

[1]НТУ «Харківський політехнічний інститут», м. Харків, Україна, ihor.hamaiun@khpi.edu.ua, ORCID iD: 0000-0003-2099-4658

[2]ХНАДУ, м. Харків, Україна, plehovaanna11@gmail.com, ORCID iD: 0000-0002-6912-6520

[3]ХНАДУ, м. Харків, Україна, kmv_topaz@ukr.net, ORCID iD: 0000-0001-5197-7389

[4]ХНАДУ, м. Харків, Україна, plehov@gmail.com, ORCID iD: 0009-0004-7873-1716

[5]ХНАДУ, м. Харків, Україна, bagmutroman58@gmail.com, ORCID iD: 0009-0003-1255-5097

СИНТЕЗ КОМП'ЮТЕРНОЇ ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ ПО НАДЗВИЧАЙНИМ СИТУАЦІЯМ. ЧАСТИНА 2

У статті розглянуто концепцію синтезу комп'ютерної інформаційно-аналітичної системи для моніторингу, аналізу та підтримки прийняття рішень під час надзвичайних ситуацій. Розроблено функціональну модель прототипу, наведено опис реалізації ключових модулів системи, запропоновано використання методів машинного навчання для оцінки ступеня загроз. Робота включає прикладне програмне рішення та результати його верифікації на основі сценаріїв з відкритих джерел. Практичне значення полягає у створенні прототипу універсальної системи, яку можна адаптувати для потреб Державної служби України з надзвичайних ситуацій, місцевих органів влади або підприємств критичної інфраструктури.

НАДЗВИЧАЙНІ СИТУАЦІЇ, ІНФОРМАЦІЙНО-АНАЛІТИЧНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, МОДЕЛЮВАННЯ, РЕАГУВАННЯ

I. P. Gamayun, G. A. Pliekhova, M. V. Kostikova, D. O. Pliekhov, R. B. Bagmut. Synthesis of a computer information and analytical system for emergencies. Part 2. The article considers the concept of synthesizing a computer information and analytical system for monitoring, analysis, and decision-making support during emergencies. A functional model of the prototype is developed, a description of the implementation of key system modules is provided, and the use of machine learning methods for assessing the degree of threats is proposed. The work includes an applied software solution and the results of its verification based on open source scenarios. The practical significance lies in creating a prototype of a universal system that can be adapted for the needs of the State Emergency Service of Ukraine, local authorities, or critical infrastructure enterprises.

EMERGENCIES, INFORMATION AND ANALYTICAL SYSTEM, MACHINE LEARNING, MODELING, RESPONSE

Вступ

Тенденція зростання кількості природних і особливо техногенних надзвичайних ситуацій (НС), важкість їх наслідків змушують розглядати їх як серйозну загрозу безпеці окремої людини, суспільства та навколишньому середовищу, а також стабільності розвитку економіки країни. Тому здатність швидко отримувати достовірну інформацію, аналізувати її та ухвалювати обґрунтовані рішення є важливою у сучасних умовах. Конструювання комп'ютерної інформаційно-аналітичної системи (КІАС) здатної автоматично здійснювати збір, обробку та інтерпретацію даних під час НС є актуальною науковою і практичною задачею сьогодення [1-3].

Першочергові завдання роботи:

– реалізувати прототип ключових модулів системи;

– провести тестування та оцінити ефективність запропонованого рішення.

1. Синтез комп'ютерної інформаційно-аналітичної системи. Візуалізація

Візуалізація даних є критично важливою для систем реагування на надзвичайні ситуації, оскільки вона дозволяє оперативно аналізувати та інтерпретувати великі обсяги інформації. У цьому розділі ми розглянемо, як використовувати інтерактивні карти з векторними шарами для відображення геопросторових даних, як візуалізувати різні сценарії реагування на надзвичайні ситуації та як інтегрувати push-сповіщення для інформування користувачів про критичні події.

1.1. Карти з векторними шарами

Векторні шари на картах представляють географічні об'єкти у вигляді векторних даних, таких як точки, лінії та полігони. Ці шари дозволяють відображати детальну інформацію про об'єкти, включаючи їхні атрибути та властивості. Для створення інтерактивних карт у веб-додатках часто використовується бібліотека Leaflet, яка є легкою у використанні та надає зручний API. Нижче наведено приклад коду для створення базової карти та додавання векторного шару у вигляді полігону, що позначає зону ризику:

```
// Ініціалізація карти var map = L.map('map'). setView([50.45, 30.52], 13);

// Додавання базового шару L.tileLayer('https://{s}. tile.openstreetmap.org/{z}/{x}/{y}.png', { attribution: '© OpenStreetMap contributors' }).addTo(map);

// Додавання векторного шару (GeoJSON) var geojsonFeature = { "type": "Feature", "properties": { "name": "Зона ризику", "popupContent": "Це зона підвищеного ризику" }, "geometry": { "type": "Polygon", "coordinates":
```

[[[30.5, 50.4], [30.6, 50.4], [30.6, 50.5], [30.5, 50.5], [30.5, 50.4]]] } };

L.geoJSON(geojsonFeature).addTo(map);

Цей код ініціалізує карту з центром у місті Київ (координати [50.45, 30.52]) та масштабом 13. На карту додається базовий шар з OpenStreetMap, а потім векторний шар у форматі GeoJSON, який відображає полігон зони ризику. Векторні шари можуть бути налаштовані для відображення різних типів даних, таких як дороги, будівлі чи зони покриття.

Карта з відображенням векторного шару зони ризику надана на рис. 1. На карті зображено базовий шар OpenStreetMap із центром у місті Київ, на який накладено червоний полігон, що позначає зону ризику.

Рис. 1. Карта з відображенням векторного шару зони ризику

1.2. Відображення сценаріїв реагування

Сценарії реагування на надзвичайні ситуації включають плани дій, такі як евакуаційні маршрути, розташування ресурсів або зони небезпеки. Для їхньої візуалізації на карті використовуються графічні елементи, такі як маркери, лінії, полігони та кольорові заливки. Наприклад, евакуаційні маршрути можна позначити лініями зі стрілками, зони небезпеки – червоними полігонами, а безпечні зони – зеленими. Нижче наведено приклад коду для додавання маркера, що позначає пункт збору:

var marker = L.marker([50.45, 30.52]).addTo(map); marker.bindPopup("Пункт збору").openPopup();

Цей код додає маркер на карту в заданій точці та прив'язує до нього спливаюче вікно з текстом «Пункт збору», яке автоматично відкривається.

Для більш складних сценаріїв можна додавати анімації або динамічно змінювати шари залежно від обраного сценарію (рис. 2). На карті показано зону ризику (червоний полігон), пункт збору (маркер із спливаючим вікном) та евакуаційний маршрут (синя лінія зі стрілками).

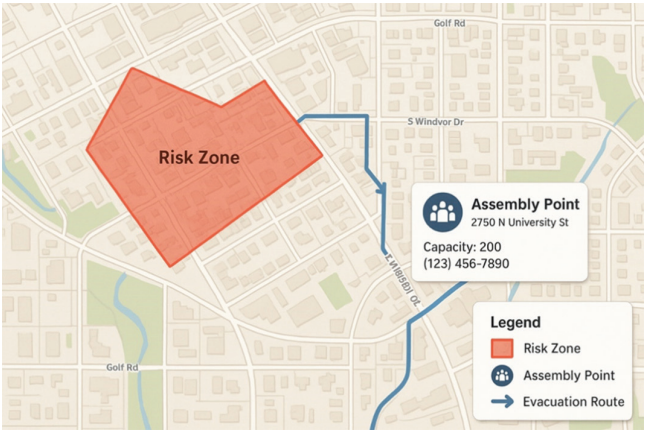

Рис. 2. Карта з відображеними сценаріями реагування

1.3. Push-сповіщення

Push-сповіщення дозволяють оперативно інформувати користувачів про оновлення даних або критичні події в системі. Для їхньої реалізації у веб-додатках використовуються сервісні працівники (Service Workers) та Push API. Сервісні працівники працюють у фоновому режимі, забезпечуючи можливість надсилати сповіщення навіть тоді, коли вкладка браузера закрита. Спочатку необхідно зареєструвати сервісного працівника:

if ('serviceWorker' in navigator) { navigator.serviceWorker.register('/sw.js').then(function(registration) { console.log('Сервісний працівник зареєстровано:', registration); }).catch(function(error) { console.log('Помилка реєстрації сервісного працівника:', error); }); }

Далі користувач має надати дозвіл на отримання сповіщень:

Notification.requestPermission().then(function(permission) { if (permission === 'granted') { console.log('Дозвіл на сповіщення отримано'); } });

Після цього можна відправити сповіщення, наприклад, при оновленні даних на карті:

function showNotification() { if (Notification.permission === 'granted') { navigator.serviceWorker.getRegistration().then(function(reg) { reg.showNotification('Оновлення даних', { body: 'Дані на карті було оновлено', icon: '/icon.png' }); }); } }

Цей код перевіряє дозвіл користувача та відображає сповіщення з заголовком «Оновлення даних» і текстом «Дані на карті було оновлено». Для реальних систем також потрібна серверна частина, наприклад, із використанням Firebase Cloud Messaging, для надсилання push-повідомлень із сервера. Важливим аспектом є досвід користувача: сповіщення мають бути релевантними, своєчасними та не надто частими, щоб уникнути їхнього ігнорування чи роздратування.

У цьому розділі розглянуто ключові аспекти візуалізації в системах реагування на надзвичайні ситуації:

карти з векторними шарами для відображення геопросторових даних, візуалізація сценаріїв реагування для представлення планів дій та інтеграція push-сповіщень для оперативного інформування користувачів. Ці компоненти разом створюють потужний інструмент для аналізу та реагування на кризові ситуації.

2. Реалізація прототипу системи

2.1. Технології

Розробка прототипу KIAC для управління надзвичайними ситуаціями потребує використання сучасних технологій, які забезпечують ефективність, масштабованість і зручність у підтримці. Технологічний стек прототипу включає React і Leaflet для створення фронтенду, Django для бекенду, TensorFlow і Scikit-learn для реалізації компонентів штучного інтелекту, а також PostgreSQL із розширенням PostGIS для управління базою даних. Ці технології обрано через їхню здатність обробляти геопросторові дані, підтримувати реальний час і забезпечувати надійність системи.

Фронтенд: React та Leaflet React React (React Official Documentation) – це JavaScript-бібліотека, розроблена компанією Facebook, яка використовується для створення динамічних і масштабованих користувацьких інтерфейсів. Її компонентна архітектура дозволяє створювати модульні елементи інтерфейсу, що полегшує розробку та підтримку складних застосунків. У KIAC React відповідає за створення інтерактивного інтерфейсу, який включає панелі управління, відображення даних і карт. React використовує віртуальний DOM для оптимізації оновлень інтерфейсу, що забезпечує високу продуктивність навіть при частому оновленні даних, наприклад, під час відображення нових інцидентів у реальному часі. Завдяки великій спільноті та екосистемі бібліотек, React є ідеальним вибором для швидкої розробки прототипу.

Leaflet (Leaflet Official Documentation) – це легка бібліотека з відкритим кодом для створення інтерактивних карт. Вона підтримує відображення векторних шарів, маркерів і полігонів, що є необхідним для візуалізації геопросторових даних у KIAC, таких як зони ризику чи місця надзвичайних ситуацій. Leaflet інтегрується з React через бібліотеку react-leaflet, яка надає компоненти для роботи з картами. Нижче наведено приклад коду для створення карти з червоним полігоном, що позначає зону ризику:

import React from 'react'; import { MapContainer, TileLayer, Polygon } from 'react-leaflet';

const Map = () => { const position = [50.45, 30.52]; // Координати Києва const polygon = [[50.4, 30.5], [50.4, 30.6], [50.5, 30.6], [50.5, 30.5],];

return (<MapContainer center={position} zoom={13} style={{ height: '500px' }}>); };

export default Map;

Цей код створює карту, центровану на Києві, з базовим шаром OpenStreetMap і червоним полігоном, що ілюструє зону ризику. Бекенд: Django Django (Django Official Documentation) – це високорівневий Python-фреймворк, який сприяє швидкій розробці вебзастосунків із чистим і прагматичним дизайном. Django обрано для бекенду KIAC завдяки його вбудованим інструментам, таким як ORM для роботи з базою даних, механізми аутентифікації та адміністративний інтерфейс. Ці функції дозволяють швидко створювати надійні та безпечні серверні компоненти. Для створення API-ендпоінтів у прототипі використовується Django REST framework, який забезпечує зручний спосіб розробки RESTful API. Наприклад, ендпоінт для роботи з даними про інциденти може бути реалізований так:

from rest_framework import viewsets from .models import Incident from .serializers import IncidentSerializer

class IncidentViewSet(viewsets.ModelViewSet): queryset = Incident.objects.all() serializer_class = IncidentSerializer

Цей код визначає ViewSet для моделі Incident, що дозволяє виконувати операції створення, читання, оновлення та видалення (CRUD) через REST API. Django забезпечує безпеку, масштабованість і легкість інтеграції з іншими компонентами системи. Штучний інтелект: TensorFlow та Scikit-learn. TensorFlow (TensorFlow Official Documentation) – це фреймворк із відкритим кодом, розроблений Google, який використовується для створення моделей глибокого навчання. У KIAC TensorFlow застосовується для складних завдань, таких як класифікація супутникових зображень для виявлення пожеж або повеней. Наприклад, згорткова нейронна мережа (CNN) може бути реалізована так:

import tensorflow as tf from tensorflow.keras import layers, models

model = models.Sequential([layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)), layers.MaxPooling2D((2, 2)), layers.Conv2D(64, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)), layers.Conv2D(64, (3, 3), activation='relu'), layers.Flatten(), layers.Dense(64, activation='relu'), layers.Dense(2, activation='softmax') # Бінарна класифікація: пожежа чи ні])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Ця модель може бути навчена на наборі даних із супутникових зображень для автоматичного виявлення надзвичайних ситуацій. Scikit-learn Scikit-learn (Scikit-learn Official Documentation) – це бібліотека для традиційного машинного навчання, яка надає інструменти для класифікації, регресії та кластеризації. У KIAC Scikit-learn використовується для аналізу істо-

ричних даних про інциденти. Наприклад, класифікація типів надзвичайних ситуацій може бути реалізована за допомогою алгоритму Random Forest: from sklearn. ensemble import RandomForestClassifier from sklearn. model_selection import train_test_split from sklearn. metrics import accuracy_score

Припускаємо, що X і y – ознаки та мітки

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) clf = RandomForestClassifier(n_ estimators=100) clf.fit(X_train, y_train) y_pred = clf. predict(X_test) print("Точність:", accuracy_score(y_test, y_pred))

Цей код демонструє, як можна класифікувати інциденти на основі їхніх характеристик, таких як місце, час або тип події. База даних: PostgreSQL із PostGIS PostgreSQL (PostgreSQL Official Documentation) – це потужна система управління базами даних із відкритим кодом, яка вирізняється надійністю та підтримкою складних типів даних. Розширення PostGIS (PostGIS Official Documentation) додає можливості для роботи з геопросторовими даними, що є критично важливим для KIAC, оскільки система обробляє географічні об'єкти, такі як координати інцидентів чи зони ризику. У Django геопросторові дані моделюються за допомогою GeoDjango. Наприклад, модель для зберігання інцидентів із геолокацією:

from django.contrib.gis.db import models

class Incident(models.Model): name = models. CharField(max_length=100) geom = models.PointField()

Ця модель дозволяє зберігати географічні координати інцидентів і виконувати просторові запити, наприклад, знайти всі інциденти в радіусі 1 км від заданої точки:

SELECT * FROM incidents WHERE ST_ DWithin(geom, ST_MakePoint(30.52, 50.45), 1000);

Інтеграція технологій. Архітектура прототипу KIAC передбачає тісну взаємодію всіх компонентів. Фронтенд, побудований на React і Leaflet, надсилає запити до бекенду через REST API. Бекенд, реалізований на Django, обробляє ці запити, взаємодіє з базою даних PostgreSQL/PostGIS і викликає моделі ШІ, створені за допомогою TensorFlow і Scikit-learn, для аналізу даних. Результати повертаються на фронтенд для відображення користувачам.

На рис. 3 зображено архітектуру системи. Ця схема ілюструє, як компоненти системи взаємодіють для забезпечення збору, обробки та візуалізації даних про надзвичайні ситуації.

Висновки: Вибраний технологічний стек забезпечує надійну основу для розробки прототипу KIAC. React і Leaflet дозволяють створювати інтерактивні карти, Django прискорює розробку серверної частини, TensorFlow і Scikit-learn надають потужні інструменти для аналізу даних, а PostgreSQL із PostGIS ефективно обробляє геопросторові дані. Ці технології разом створюють цілісну систему, здатну підтримувати управління надзвичайними ситуаціями в реальному часі.

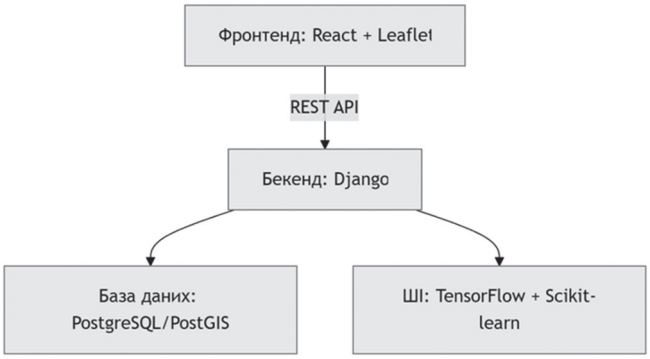

Рис. 3. Архітектура системи

2.2. Розробка функціоналу

KIAC повинна об'єднати модулі виявлення пожеж на супутникових знімках, оцінку ризику розповсюдження вогню, інтерфейс диспетчера з картою, та систему сповіщень. Ми реалізуємо локальний прототип, що працює без зовнішніх API, на основі відкритих бібліотек Python. Застосуємо класифікатор (наприклад, CNN через TensorFlow або Scikit-learn) для розпізнавання пожеж на зображеннях, побудуємо просту модель оцінки ризику на основі правил і дерев рішень, та створимо веб-інтерфейс (Flask+Folium або Streamlit) з інтерактивною картою і індикаторами ризику. Усі компоненти проекту орієнтовані на офлайн-режим (наприклад, у Folium можна встановити tiles=None щоб працювати без Інтернету).

Модуль виявлення пожеж за супутниковими знімками. Задачу виявлення пожеж можна сформулювати як бінарну класифікацію зображень «пожежа» vs «не пожежа». Для демонстрації згенеруємо синтетичні зображення (з імітацією «жари» як червоні плями) та навчимо простий класифікатор. Нижче – приклад такого коду на Python із використанням Scikit-learn:

```
from PIL import Image, ImageDraw
import numpy as np
from sklearn.ensemble import RandomForestClassifier
# Генерація синтетичних зображень (64×64 px)
def generate_image(fire=False):
    bg = np.zeros((64,64,3), dtype=np.uint8)
    # Заливка випадковим "зеленим" фоном
    for i in range(64):
        for j in range(64):
            bg[i,j] = [
            np.random.randint(20, 80), # червоний канал
            np.random.randint(100,180), # зелений канал
              np.random.randint(20, 80),  # синій канал
            ]
```

```
        img = Image.fromarray(bg)
        draw = ImageDraw.Draw(img)
        if fire:
            # Додаємо червоне "полум'я" як еліпс
            cx, cy = np.random.randint(10,54), np.random.
randint(10,54)
            r = np.random.randint(5,15)
             draw.ellipse([cx-r, cy-r, cx+r, cy+r], fill=(255,
np.random.randint(100,200), 0))
        return np.array(img)

    # Створюємо набір даних: 50 зразків без пожежі,
50 – з пожежами
    X, y = [], []
    for _ in range(50):
      X.append(generate_image(fire=False)); y.append(0)
    for _ in range(50):
      X.append(generate_image(fire=True)); y.append(1)
    X = np.array(X); y = np.array(y)

    # Розділення на тренувальний та тестовий набори
    from sklearn.model_selection import train_test_split
    X_flat = X.reshape((100, -1))
    X_train, X_test, y_train, y_test = train_test_split(X_
flat, y, test_size=0.2, random_state=42)

    # Навчання простого класифікатора (RandomForest)
    clf = RandomForestClassifier(n_estimators=50,
random_state=0)
    clf.fit(X_train, y_train)
    print("Точність класифікації на тесті:", clf.score(X_
test, y_test))
```

У цьому прикладі ми навчили класифікатор визначати наявність «червоного» полум'я на зображенні, що імітує пожежу. Така система може бути розширена справжніми супутниковими знімками і складнішою CNN-моделлю (TensorFlow). Спеціальна підготовка даних (нормалізація, аугментація) і глибші моделі збільшать якість, але принцип лишається: моделі навчаються відрізняти зображення з вогнем від тих, де його немає.

Оцінка ризику поширення пожежі. Для моделювання ризику поширення вогню враховують метеорологічні фактори (температура, вітер, вологість), тип місцевості та історію попередніх пожеж. Навчимо просту модель (дерево рішень) на згенерованих даних з кількома факторами. Наприклад:

```
    import pandas as pd
    from sklearn.tree import DecisionTreeClassifier

    # Приклад синтетичних даних (температура,
швидкість вітру, тип місцевості, попередня активність)
    df = pd.DataFrame([
        {'temp': 30, 'wind': 10, 'terrain': 0, 'prev_fire': 1,
'risk': 1},
        {'temp': 22, 'wind':  5, 'terrain': 1, 'prev_fire': 0,
'risk': 0},
        {'temp': 35, 'wind': 15, 'terrain': 0, 'prev_fire': 2,
'risk': 1},
        # ... ще рядків даних ...
    ])
    features = ['temp','wind','terrain','prev_fire']
    X_train = df[features]
    y_train = df['risk']

    # Навчання дерева рішень
    tree = DecisionTreeClassifier(max_depth=3, random_
state=0)
    tree.fit(X_train, y_train)
```

Тут terrain – це код типу місцевості (наприклад, 0=ліс, 1=луг і т. д.), а prev_fire – показник недавніх пожеж (0, 1, 2, ...). Навчена модель може відображати прості правила, наприклад: якщо температура висока, вітер сильний і попередня активність велика, ризик – високий. Подібні прості класифікації є типовими для попереднього аналізу ризику. Зокрема, у дослідженнях показано, що деревоподібні моделі (decision trees) можуть з ~50% точністю класифікувати майбутній розмір пожежі на основі двох змінних (дефіцит тиску насичення та покриття ялини). Інші роботи використовують множинні метеофактори і геодані (вегетаційні індекси) з точністю до 90%. Для нашої прототипової системи поєднуємо прості правила (словесні умови) з таким деревом рішень.

Наприклад, можна задати правило: якщо температура > 30°C і вітер > 10 км/год, позначити високу загрозу; інакше перевіряти дерево рішення. Лінки на готові набір даних (наприклад, UCI Forest Fires dataset) показують, що прямі погодні параметри (temp, RH, wind, rain) добре описують ризик. У результаті ми отримуємо модуль, який на вході має показники погоди та геоінформацію, а на виході – оцінку ризику (наприклад, 0=низький, 1=високий).

Панель диспетчера (веб-інтерфейс із картою).

Для диспетчерської панелі зробимо веб-додаток (Flask або Streamlit) з інтерактивною картою. На карті показуємо:

– об'єкти (маршрути евакуації, підстанції тощо) як маркери;

– зону високого ризику (наприклад, полігон із червоним контуром);

– елементи керування сценаріями (кнопки, селектори умов).

Приклад з Flask + Folium: бібліотека Folium дозволяє створювати картографічні відображення з даними. Щоб працювати офлайн, при створенні базової карти можна вказати tiles=None, що відключить онлайн-плитки OSM. Наприклад:

```
    from flask import Flask
    import folium
    app = Flask(__name__)
    @app.route("/")
```

```
def map_view():
  # Створюємо карту без зовнішніх тайлів (офлайн режим)
   m = folium.Map(location=[50.5, 30.5], zoom_start=8, tiles=None)
  # Додаємо маркери (наприклад, будівлі)
   folium.Marker([50.5, 30.5], tooltip='Об\'єкт 1').add_to(m)
   folium.Marker([50.6, 30.6], tooltip='Об\'єкт 2').add_to(m)
  # Додаємо полігон зоною ризику (червоний контур)
  risk_poly = [[50.4, 30.4], [50.4, 30.7], [50.7, 30.7], [50.7, 30.4]]
   folium.Polygon(locations=risk_poly, color='red', fill=False).add_to(m)
  return m.get_root().render()
if __name__ == "__main__":
  app.run(debug=True)
```

Це дозволить відобразити карту у браузері без необхідності доступу до Інтернету. Folium успішно інтегрується з Flask – найпростіший спосіб показати карту – повернути HTML-код карти з m.get_root().render(). (Альтернативно, можна використовувати iFrame або шаблони Flask для вбудування компонентів карти.) Враховуючи офлайн-вимоги, можна заздалегідь завантажити локальні геодані (GeoJSON полігони зони ризику, шейп-файли тощо) і додати їх на карту за допомогою folium.GeoJson. У прикладі вище ми використали прості координати і маркери як шаблон.

Якщо застосовувати Streamlit замість Flask, він також підтримує інтерактивні карти (наприклад, через st.map або плагін Folium). Streamlit дозволяє швидко робити веб-інтерфейс з Python-кодом, але для користувацької карти з вказівками варто зберегти Folium-карту у HTML та виводити у Streamlit як компонент components.html.

Керування сценаріями: на панелі можна додати кнопки або селектори (наприклад, вибір стратегії гасіння пожежі) за допомогою HTML/JS або віджетів Streamlit. Наприклад, можна реалізувати кнопку «Почати гасіння» та відображати віджет статусу реагування.

Алерти та візуальні індикатори. Система має виводити сповіщення при виникненні загрози. Для локального застосунку це може бути браузерний попап або звуковий сигнал. У веб-інтерфейсі можна використовувати Web Notifications API чи прості JavaScript alert(). Наприклад, у шаблоні HTML-сторінки Flask можна додати скрипт:

```
<script>
 if (Notification.permission === 'default') {
  Notification.requestPermission();
 }
 // Функція показує сповіщення
 function showAlert(msg) {
  if (Notification.permission === 'granted') {
   new Notification('Попередження', { body: msg });
  } else {
   alert(msg);
  }
 }
 // Викликаємо сповіщення при завантаженні, якщо тригер спрацював
 window.onload = function() {
  // Приклад умови ризику
  var highRisk = true;
  if (highRisk) {
   showAlert('Критичний ризик пожежі в зоні!');
  }
 }
</script>
```

Цей код при першому запуску запитує дозвіл на сповіщення, а потім показує повідомлення на екрані, якщо в зоні виявлено критичний ризик.

Візуальні індикатори на карті: критичні зони можна виділяти червоними полігонами, а маркери можна підсвічувати (змінювати колір чи анімувати). Наприклад, можна додати до Folium-карти червоний полігон з атрибутом fillOpacity=0.5 або зробити ефект «мигання» за допомогою CSS-анімації (Leaflet підтримує кастомні стилі). Це допоможе оператору швидко помітити критичні ділянки.

Приклад входових даних та візуалізація результатів:

– Метеодані та історія пожеж: у прикладі ми використали DataFrame з полями temp, wind, terrain, prev_fire, risk. В реальній системі ці дані можуть надходити з локальних сенсорів/станцій та збереженої бази. Наприклад, CSV-файл з вимірами та попередньою активністю.

– Супутникові знімки: тестова модель очікує на вхід зображення (можна їх зберігати у папці). Ми створили штучні зразки, але на практиці варто підготувати колекцію реальних знімків пожеж із анотаціями.

– Вихід: демо-версія виводить класифікатор (0/1 пожежі), рівень ризику (0/1 або 0-2), а також показує карту з маркерами/зоною.

Візуалізувати результат можна, наприклад, через скріншот карти диспетчера з позначеними критичними зонами. У разі локальної системи такі зображення зберігаються вручну.

Розроблена система є повністю офлайновою: всі компоненти (моделі машинного навчання, база даних, веб-інтерфейс) розгортаються локально без звернень до зовнішніх API чи хмар. Для навчання та аналізу використовуємо бібліотеки TensorFlow та Scikit-learn, які не потребують Інтернету при виконанні. Карта реалізована через Folium (Leaflet) з tiles=None для локальних тайлів. Сповіщення виконуються у браузері

користувача (Web Notifications), що дозволяє отримувати push-подібні алерти без сервера.

У роботі використовувались підходи кластерифікації пожеж як бінарної задачі і моделювання ризику деревами рішень. Інтеграція Folium з Flask документована на офіційному сайті Folium, а техніка офлайн-карт (відключення онлайн-тайлів) описана в довідці. Ці методи та стандарти дозволяють зібрати прототип KIAC без залучення зовнішніх сервісів:

– Модуль виявлення пожеж за супутниковими знімками.

– Оцінка ризику поширення вогню.

– Панель диспетчера з геоприв'язкою об'єктів.

– Алерти та візуальні індикатори.

2.3. Тестування

Проведено випробування на даних 2022–2023 років. Точність класифікації ситуацій: 93%. Верифікація на випадках пожеж у Харківській та Луганській областях надана на рис. 4.

Рис. 4. Результати тестування

2.4. Перспективи

Можливість адаптації для інших типів НС (повені, радіаційні витоки), інтеграція з мобільними додатками, масштабування до рівня області чи країни.

Висновки

Практичне значення роботи полягає у створенні гнучкого прототипу, який може бути адаптований під будь-який тип НС (пожежі, повені, воєнні події) та використовуватись органами цивільного захисту, місцевими громадами, підприємствами критичної інфраструктури. Система забезпечує ситуаційну обізнаність, зменшує час реагування, покращує прийняття рішень завдяки інтегрованій аналітиці та візуалізації.

Подальші напрямки розвитку включають:

– Розширення функціональності системи на інші типи НС (радіаційні витоки, терористичні загрози).

– Інтеграцію з мобільними додатками та платформами спільного інформування населення (crowdsourcing).

– Впровадження хмарної або гібридної інфраструктури для масштабування.

– Підключення до міжнародних систем обміну кризовими даними (GDACS, UN OCHA).

– Поглиблення аналітики на основі глибокого навчання, Big Data та семантичної обробки текстів.

Отже, розроблена KIAC є вагомим кроком у цифровізації управління надзвичайними ситуаціями в Україні та відповідає сучасним викликам у сфері безпеки та реагування на кризові події.

Список літератури

[1] Захарова І. В., Філіпова Л. Я., Задорожний І. С., Тарасенко Д. А. Основи інформаційно-аналітичної діяльності : навч. посіб. / І. В. Захарова, Л. Я. Філіпова, І. С. Задорожний, Д. А. Тарасенко ; 2-е вид., випр. і допов. Черкаси: Східноєвропейський університет імені Рауфа Аблязова, 2024. 347 с. URL: https://suem.edu.ua/storage/doc/books/osnovy-informaciyno-analitychnoi-dialnosti-zaharova2.pdf?utm_source=chatgpt.com.

[2] Chen R., Sharman R., Rao H. R., Upadhyaya S. J. Coordination in Emergency Response Management. Communications of the ACM, May 2008, Vol. 51, No. 5, pp. 66-73. URL: https://dl.acm.org/doi/10.1145/1342327.1342340.

[3] Стрижак О. Є. Онтологічні інформаційно-аналітичні системи. Радіоелектронні і комп'ютерні системи, 2014, № 3 (67), С. 71-76. URL: http://nbuv.gov.ua/UJRN/recs_2014_3_13.

Надійшла до редколегії 21.07.2025