ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття рівня вищої освіти «бакалавр»
(рівень вищої освіти)

Апаратно-програмний комплекс розпізнавання графічних образів на основі нейронної мережі
(тема кваліфікаційної роботи українською мовою)

Hardware and software complex for recognizing graphic images based on a neural network
(тема кваліфікаційної роботи англійською мовою)

Виконав(ла): здобувач(ка) ВО денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Астєненко Станіслав Володимирович
(прізвище, ім'я, по-батькові)

Керівник к.т.н., доцент Нєнов О. Л.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.т.н., доцент Пенко В. Г.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2026 р.

Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № __5__
протокол № __ від «___» ____ 2026 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ _Світлана АНТОЩУК_
(підпис) (ім'я, прізвище)

Одеса - 2026

АНОТАЦІЯ

У даній кваліфікаційній роботі розглянуто розробку локальної програмно-апаратної системи комп’ютерного зору для аналізу відеопотоку і розпізнавання образів в режимі реального часу з використанням технологій штучного інтелекту. Основною метою роботи є створення програмного забезпечення, здатного виявляти потенційно небезпечні ситуації без використання віддалених серверів.

У роботі реалізовано веб-додаток, який функціонує безпосередньо в браузері та використовує бібліотеку TensorFlow.js разом із попередньо навченими моделями, зокрема COCO-SSD для детекції об’єктів, MobileNet для класифікації сцени, FaceMesh для аналізу обличчя та PoseNet для визначення пози людини. У межах дослідження також розроблено архітектуру програмно-апаратної системи та обґрунтовано основні проєктні рішення, що забезпечують ефективну обробку відеопотоку в режимі реального часу. Окрім цього, реалізовано ключові функціональні модулі системи та описано основні аспекти її програмної реалізації.

Система забезпечує обробку відео з камери в режимі реального часу, визначення контексту сцени, виявлення потенційно небезпечних об’єктів (наприклад, гострих предметів або вогню), а також аналіз поведінки (падіння людини, різкі рухи, підняті руки). Реалізовано механізм оцінки рівня загрози, звукового сповіщення та автоматичного збереження знімків у випадку небезпечних подій.

Основною перевагою розробленого рішення є повністю локальна обробка даних, що забезпечує високу швидкодію, відсутність затримок, незалежність від інтернет-з’єднання та підвищений рівень конфіденційності. Запропонований підхід може бути використаний у системах відеоспостереження, безпеки приміщень та інших прикладних задачах.

Ключові слова: комп’ютерний зір, аналіз відеопотоку, машинне навчання, локальна обробка даних, детекція об’єктів, аналіз поведінки, розпізнавання обличчя, система безпеки, системи реального часу.

ABSTRACT

This thesis examines the development of a local computer vision hardware-software system for real-time video stream analysis and image recognition using artificial intelligence technologies. The main objective of this work is to create software capable of detecting potentially dangerous situations without the use of remote servers.

The thesis implements a web application that runs directly in a browser and uses the TensorFlow.js library along with pre-trained models, specifically COCO-SSD for object detection, MobileNet for scene classification, FaceMesh for facial analysis, and PoseNet for determining a person's pose. The research also developed the architecture of a hardware-software system and justified the main design decisions that ensure efficient real-time video stream processing. In addition, key functional modules of the system were implemented, and the main aspects of its software implementation were described.

The system provides real-time processing of camera video, scene context recognition, detection of potentially hazardous objects (such as sharp objects or fire), and behavioral analysis (a person falling, sudden movements, raised hands). A mechanism for threat level assessment, audio alerts, and automatic image saving in the event of dangerous incidents has been implemented.

The main advantage of the developed solution is fully local data processing, which ensures high performance, no delays, independence from an internet connection, and an increased level of privacy. The proposed approach can be used in video surveillance systems, building security systems, and other practical applications.

Keywords: computer vision, video stream analysis, machine learning, local data processing, object detection, behavior analysis, face recognition, security system, real-time systems.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 7

ВСТУП 8

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ, ПОСТАНОВКА ЗАДАЧІ 10

1.1 Основи комп’ютерного зору 10

1.2 Методи розпізнавання об’єктів 10

1.3 Аналіз поведінки людини 11

1.4 Огляд сучасних систем відеоспостереження 12

1.5 Обґрунтування вибору підходу 13

1.6 Постановка задачі 13

Висновки до розділу 1 15

2 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ АНАЛІЗУ ВІДЕОПОТОКУ 16

2.1 Обґрунтування вибору засобів розробки 16

2.2 Загальна архітектура програмно-апаратної системи 20

2.3 Обґрунтування вибору апаратних засобів 21

2.4 Опис функціональних модулів системи 25

2.4.1 Модуль отримання відеопотоку 25

2.4.2 Модуль обробки зображень 25

2.4.3 Модуль штучного інтелекту 26

2.4.4 Модуль логічного аналізу 26

2.4.5 Модуль взаємодії з користувачем 26

2.5 Алгоритм роботи програмно-апаратної системи 27

2.6 Логіка визначення потенційно небезпечних ситуацій 28

Висновки до розділу 2 30

3 РЕАЛІЗАЦІЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ 31

3.1 Загальна характеристика програмної реалізації 31

3.2 Реалізація модулів обробки відеопотоку 31

3.2.1 Модуль отримання відеопотоку 32

3.2.2 Модуль обробки кадрів 32

3.2.3 Модуль керування циклом обробки 33

3.2.4 Модуль аналізу відеоданих 34

3.2.5 Модуль візуалізації результатів 34

3.2.6 Модуль оптимізації продуктивності 35

3.2.7 Модуль взаємодії з користувачем 35

3.3 Інтеграція моделей машинного навчання 36

3.3.1 Інтеграція моделі детекції об’єктів 36

3.3.2 Інтеграція моделі класифікації сцени 37

3.3.3 Інтеграція моделі аналізу обличчя 37

3.3.4 Інтеграція моделі визначення пози людини 38

3.3.5 Спільна робота моделей 39

3.3.6 Оптимізація роботи моделей 41

3.4 Реалізація механізму визначення потенційно небезпечних ситуацій 41

3.5 Тестування та оцінка ефективності системи 44

3.5.1 Тестування інтерфейсу користувача програмно-апаратної системи 45

3.5.2 Тестування детекції об’єктів 46

3.5.3 Тестування аналізу поведінки людини 46

3.5.4 Тестування механізму визначення небезпечних ситуацій..... 48
3.5.5 Аналіз продуктивності програмно-апаратної системи 50
3.6 Обґрунтування ефективності обраної архітектури 51
Висновки до розділу 3 54
ВИСНОВКИ..... 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 59
ДОДАТОК А Програмний код 61

ПЕРЕЛІК СКОРОЧЕНЬ

AI — Artificial Intelligence (штучний інтелект)

API — Application Programming Interface (інтерфейс прикладного програмування)

CNN — Convolutional Neural Network (згорткова нейронна мережа)

COCO-SSD — модель детекції об’єктів

FPS — Frames Per Second (кількість кадрів на секунду)

JS — JavaScript

ML — Machine Learning (машинне навчання)

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та систем безпеки особливої актуальності набувають засоби автоматизованого аналізу відеопотоків. Зростання кількості об’єктів відеоспостереження, необхідність оперативного реагування на потенційно небезпечні ситуації, а також обмеженість людських ресурсів обумовлюють потребу у впровадженні інтелектуальних систем на основі технологій штучного інтелекту [1].

На сьогодні існує значна кількість рішень у сфері комп’ютерного зору, які базуються на використанні хмарних сервісів та централізованих обчислювальних ресурсів. Такі підходи забезпечують високу точність аналізу, однак мають ряд суттєвих недоліків, зокрема затримки при передачі даних, залежність від стабільності інтернет-з’єднання та ризики, пов’язані з конфіденційністю інформації. У світовій практиці спостерігається тенденція переходу до локальних (edge) обчислень, які дозволяють виконувати обробку даних безпосередньо на пристрої користувача.

Актуальність даної роботи полягає у необхідності створення ефективного програмно-апаратного комплексу комп’ютерного зору, здатної в режимі реального часу аналізувати відеопотік, виявляти об’єкти, інтерпретувати поведінку людини та визначати потенційно небезпечні ситуації без використання віддалених серверів. Такий підхід дозволяє підвищити швидкодію системи, забезпечити автономність її роботи та мінімізувати ризики витоку даних.

Метою кваліфікаційної роботи є розробка програмно-апаратної системи локального аналізу відеопотоку в режимі реального часу з використанням методів комп’ютерного зору та машинного навчання, здатного виявляти об’єкти, аналізувати поведінкові характеристики людини та визначати потенційні загрози.

Об’єктом дослідження є процес аналізу відеопотоку в системах комп’ютерного зору.

Предметом дослідження є методи та програмні засоби локальної обробки відеоданих з використанням алгоритмів машинного навчання для виявлення об’єктів та поведінкових патернів.

У роботі використано методи машинного навчання, аналізу відеоданих, а також сучасні програмні засоби комп’ютерного зору та машинного навчання, зокрема бібліотеку TensorFlow.js [2] та попередньо навчені моделі, такі як COCO-SSD [3], MobileNet [4], FaceMesh [5] та PoseNet [6].

Практичне значення отриманих результатів полягає у можливості використання розробленої системи в задачах відеоспостереження, забезпечення безпеки приміщень, а також у навчальному процесі для демонстрації можливостей сучасних технологій штучного інтелекту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ, ПОСТАНОВКА ЗАДАЧІ

1.1 Основи комп’ютерного зору

Комп’ютерний зір є одним із ключових напрямів розвитку сучасних інформаційних технологій, що забезпечує можливість автоматизованого аналізу та інтерпретації візуальних даних [7]. Основною метою комп’ютерного зору є моделювання процесів сприйняття зображень людиною з використанням програмних та апаратних засобів [8].

У сучасних системах комп’ютерного зору широко застосовуються методи машинного навчання та глибинного навчання, зокрема згорткові нейронні мережі (Convolutional Neural Networks, CNN), які дозволяють ефективно вирішувати задачі класифікації, сегментації та детекції об’єктів [1, 9]. Використання таких підходів забезпечує високу точність розпізнавання навіть у складних умовах, зокрема при зміні освітлення, наявності шумів або частковому перекритті об’єктів.

Комп’ютерний зір знаходить широке застосування у різних сферах, таких як системи відеоспостереження, автономні транспортні засоби, медична діагностика, промислова автоматизація та системи безпеки [10, 11, 12]. Особливо актуальним є використання технологій комп’ютерного зору для автоматизованого моніторингу та виявлення потенційно небезпечних ситуацій у режимі реального часу.

1.2 Методи розпізнавання об’єктів

Однією з ключових задач комп’ютерного зору є розпізнавання об’єктів, яке включає визначення наявності об’єкта на зображенні та його класифікацію. Сучасні підходи до розв’язання цієї задачі базуються на використанні глибинних нейронних мереж [1].

Існують два основних підходи до аналізу зображень: класифікація та детекція. Класифікація передбачає визначення загального класу зображення, тоді як детекція дозволяє визначити місцезнаходження об’єктів та їхні межі у вигляді обмежувальних прямокутників [7].

Для задач детекції об’єктів широко використовуються моделі, такі як COCO-SSD, яка забезпечує баланс між швидкістю та точністю та підходить для роботи в режимі реального часу [3]. Для класифікації зображень застосовується модель MobileNet, оптимізована для роботи на пристроях з обмеженими обчислювальними ресурсами [4].

Застосування цих моделей дозволяє ефективно виявляти об’єкти у відеопотоці та визначати їхній тип, що є основою для подальшого аналізу ситуації.

1.3 Аналіз поведінки людини

Аналіз поведінки людини є важливим аспектом сучасних систем комп’ютерного зору, особливо у задачах забезпечення безпеки. На відміну від простого розпізнавання об’єктів, поведінковий аналіз передбачає інтерпретацію дій та положення людини у просторі.

Одним із ефективних підходів до вирішення цієї задачі є оцінка пози людини (pose estimation), яка дозволяє визначити положення ключових точок тіла, таких як голова, руки, ноги та інші суглоби [7]. На основі цих даних можна аналізувати рухи, визначати аномальні ситуації, такі як падіння або різкі рухи, а також розпізнавати жести.

Для реалізації такого підходу використовується модель PoseNet, яка дозволяє в режимі реального часу визначати координати ключових точок тіла людини [6]. Крім того, для аналізу обличчя та визначення його характеристик застосовується модель FaceMesh [5].

Використання цих моделей розширює можливості системи та дозволяє перейти від простого розпізнавання об’єктів до більш складного аналізу поведінки.

1.4 Огляд сучасних систем відеоспостереження

Сучасні системи відеоспостереження можна умовно поділити на три основні категорії: централізовані (хмарні), локальні та гібридні системи.

Централізовані системи передбачають передачу відеоданих на віддалені сервери для обробки. Основною перевагою таких систем є висока обчислювальна потужність та можливість використання складних алгоритмів. Однак вони мають суттєві недоліки, серед яких затримки передачі даних, залежність від інтернет-з’єднання та ризики витоку конфіденційної інформації.

Локальні системи виконують обробку відеоданих безпосередньо на пристрої користувача. Такий підхід забезпечує мінімальні затримки, автономність роботи та підвищений рівень безпеки даних. У сучасних умовах спостерігається тенденція до переходу від хмарних рішень до локальних систем, що обумовлено розвитком обчислювальних можливостей клієнтських пристроїв [2].

Гібридні системи поєднують переваги обох підходів, розподіляючи обчислювальні задачі між локальним пристроєм та віддаленими серверами. Як правило, первинна обробка даних (наприклад, детекція об’єктів або фільтрація подій) виконується локально, тоді як більш складний аналіз або збереження даних здійснюється у хмарному середовищі. Такий підхід дозволяє зменшити затримки, оптимізувати використання ресурсів та підвищити ефективність системи в цілому. Разом з тим, гібридні системи залишаються залежними від мережевої інфраструктури та потребують додаткових механізмів забезпечення безпеки даних.

1.5 Обґрунтування вибору підходу

У межах даної роботи обрано підхід локальної обробки відеопотоку з використанням веб-технологій. Основою реалізації є бібліотека TensorFlow.js, яка дозволяє виконувати моделі машинного навчання безпосередньо у браузері з використанням апаратного прискорення (WebGL) [2].

Вибір даного підходу обумовлений рядом переваг:

1) відсутність необхідності передачі даних на віддалені сервери;
2) зменшення затримок при обробці відео;
3) забезпечення конфіденційності інформації;
4) можливість роботи без підключення до мережі Інтернет;
5) кросплатформеність та доступність.

Таким чином, використання локальної системи комп'ютерного зору дозволяє створити ефективне рішення для аналізу відеопотоку в режимі реального часу з урахуванням сучасних вимог до швидкодії та безпеки.

1.6 Постановка задачі

У сучасних умовах стрімкого розвитку інформаційних технологій та зростання обсягів відеоданих особливої актуальності набувають системи автоматизованого аналізу відеопотоку. Використання традиційних методів відеоспостереження, що базуються на участі оператора або необхідності повного запису інформації з камер спостережень, що є недостатньо ефективним через значне навантаження на людину та обладнання або високу ймовірність пропуску важливих подій.

Сучасні інтелектуальні системи аналізу відео покликані автоматизувати процес спостереження, забезпечуючи своєчасне виявлення об'єктів, аналіз поведінки людини та визначення потенційно небезпечних ситуацій. Однак існуючі рішення часто мають обмеження, пов'язані із затримками обробки,

залежністю від зовнішніх ресурсів, а також недостатньою адаптивністю до умов реального часу.

У зв’язку з цим виникає необхідність розробки рішень, що забезпечує більш ефективний аналіз відеопотоку безпосередньо під час його отримання, із мінімальними затримками та можливістю оперативного реагування на зміну ситуації.

Об’єктом дослідження є процес аналізу відеопотоку в системах комп’ютерного зору.

Предметом дослідження є методи та програмні засоби локальної обробки відеоданих з використанням алгоритмів машинного навчання для виявлення об’єктів та поведінкових патернів.

Метою кваліфікаційної роботи є розробка програмно-апаратної системи локального аналізу відеопотоку в режимі реального часу з використанням методів комп’ютерного зору та машинного навчання, здатної виявляти об’єкти, аналізувати поведінкові характеристики людини та визначати потенційні загрози.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) провести аналіз сучасних методів комп’ютерного зору;
2) дослідити існуючі підходи до аналізу відеопотоку;
3) обґрунтувати вибір програмних та технічних засобів для реалізації системи;
4) розробити архітектуру програмної системи;
5) реалізувати модулі обробки відеоданих;
6) забезпечити інтеграцію моделей машинного навчання;
7) реалізувати механізм визначення потенційно небезпечних ситуацій;
8) провести тестування та оцінку ефективності системи.

Результатом виконання роботи має стати прототип програмно-апаратної системи, що дозволяє здійснювати автоматизований аналіз відеопотоку з можливістю своєчасного виявлення об’єктів та небезпечних ситуацій, що підвищує ефективність систем спостереження та рівень безпеки.

Висновки до розділу 1

Проведений аналіз теоретичних основ комп’ютерного зору та сучасних підходів до розпізнавання об’єктів і аналізу поведінки людини дозволяє зробити висновок, що використання методів машинного навчання є ефективним інструментом для автоматизованої обробки відеоданих [1].

Розглянуті моделі, зокрема COCO-SSD, MobileNet, PoseNet та FaceMesh, забезпечують можливість реалізації системи, здатної не лише визначати об’єкти, але й аналізувати поведінкові характеристики людини в режимі реального часу [3-6].

Аналіз існуючих систем відеоспостереження показав, що хмарні рішення мають ряд обмежень, пов’язаних із затримками та питаннями конфіденційності, що обґрунтовує доцільність використання локального підходу до обробки даних.

На основі отриманих результатів сформульовано задачу дослідження, яка полягає у розробці програмно-апаратної системи автоматизованого аналізу відеопотоку в режимі реального часу, здатної виявляти об’єкти, аналізувати поведінку людини та визначати потенційно небезпечні ситуації. Сформульована задача обумовлює необхідність вибору відповідних засобів реалізації та розробки архітектури системи, що розглядається у наступному розділі.

Таким чином, результати проведеного аналізу підтверджують доцільність розробки локальної системи комп’ютерного зору, що і визначає подальший напрямок виконання роботи, зокрема проєктування архітектури та реалізацію програмного забезпечення.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ АНАЛІЗУ ВІДЕОПОТОКУ

2.1 Обґрунтування вибору засобів розробки

У процесі розробки програмно-апаратної системи аналізу відеопотоку важливим етапом є вибір засобів реалізації, які повинні забезпечувати обробку даних у режимі реального часу, високу продуктивність та автономність роботи.

Серед можливих підходів до реалізації програмно-апаратних систем розпізнавання образів можна виділити використання нативних застосунків, серверних рішень та веб-технологій. Нативні застосунки забезпечують високу продуктивність, однак потребують встановлення та адаптації під різні платформи. Серверні рішення дозволяють використовувати значні обчислювальні ресурси, проте мають недоліки у вигляді затримок передачі даних та залежності від мережі.

Веб-технології дозволяють реалізувати систему без необхідності встановлення додаткового програмного забезпечення, забезпечують кросплатформеність та доступність. Завдяки розвитку сучасних засобів апаратного прискорення, такі рішення здатні виконувати обробку відеоданих у режимі реального часу без використання віддалених серверів [2].

Для реалізації алгоритмів машинного навчання було розглянуто ряд бібліотек, серед яких TensorFlow.js, OpenCV та PyTorch. Бібліотека TensorFlow.js версії 3.21.0 була обрана завдяки можливості виконання моделей безпосередньо у браузері, використанню апаратного прискорення та відсутності необхідності передачі даних на сервер [2] (див. таблицю 2.1).

Для вирішення задач аналізу відеопотоку було розглянуто різні моделі комп’ютерного зору. Для задачі детекції об’єктів можуть використовуватись такі моделі, як YOLO та EfficientDet, що забезпечують високу точність, однак потребують значних обчислювальних ресурсів [8].

Таблиця 2.1 - Порівняння засобів реалізації машинного навчання

Засіб	Переваги	Недоліки	Доцільність використання
TensorFlow.js	Робота у браузері; відсутність серверної частини; використання GPU (WebGL); простота інтеграції	Менша точність у порівнянні з серверними рішеннями; обмеження продуктивності	Висока — оптимально для локальних систем реального часу
OpenCV	Потужні інструменти комп'ютерного зору; велика кількість алгоритмів	Складність інтеграції у веб; потребує додаткових налаштувань	Середня — більше підходить для десктоп/серверних рішень
PyTorch	Висока точність моделей; гнучкість; популярність у наукових дослідженнях	Потребує серверного виконання; складність інтеграції у веб	Низька — не підходить для браузерної реалізації
ONNX Runtime	Універсальність; підтримка різних моделей	Складність налаштування; обмежена підтримка у браузері	Середня — перспективний, але складніший у реалізації

У даній роботі обрано модель COCO-SSD версії 2.2.2, яка характеризується меншою обчислювальною складністю та здатністю працювати у режимі реального часу на клієнтських пристроях [3] (див. таблицю 2.2)

Для задачі класифікації зображень можуть використовуватись моделі ResNet та EfficientNet, що забезпечують високу точність, проте є ресурсомісткими [1].

Таблиця 2.2 – Порівняння моделей детекції об’єктів

Засіб	Переваги	Недоліки	Доцільність використання
COCO-SSD	Висока швидкість, легка інтеграція, оптимізована для браузера	Нижча точність	Висока для реального часу
YOLO	Висока точність, швидка робота на GPU	Велике навантаження, складність інтеграції у web	Середня
EfficientDet	Хороший баланс між точністю та швидкістю	Високі вимоги до ресурсів	Середня

У роботі використано модель MobileNet версії 2.1.0, яка оптимізована для швидкої роботи та використання на пристроях з обмеженими ресурсами [4] (див. таблицю 2.3).

Таблиця 2.3 – Порівняння моделей класифікації зображень

Засіб	Переваги	Недоліки	Доцільність використання
MobileNet	Легка, швидка, оптимізована для мобільних пристроїв і web	Менша точність	Висока
ResNet	Висока точність	Високе навантаження	Середня
EfficientNet	Дуже хороша точність	Великі вимоги до ресурсів	Середня

Для аналізу положення людини розглянуто моделі OpenPose та MoveNet. У роботі використано модель PoseNet версії 2.2.1, яка забезпечує достатню точність при низьких обчислювальних витратах [6] (див. таблицю 2.4).

Таблиця 2.4 – Порівняння моделей визначення пози людини

Засіб	Переваги	Недоліки	Доцільність використання
PoseNet	Легка, працює у браузері	Менша точність	Висока
OpenPose	Висока точність	Дуже ресурсомістка	Низька
MoveNet	Швидша за PoseNet	Обмежена функціональність (менша кількість ключових точок, відсутність аналізу кистей та обличчя, обмеження при роботі з кількома людьми)	Висока

Для аналізу обличчя було розглянуто різні підходи, серед яких бібліотеки Dlib та MediaPipe. У роботі використано модель FaceMesh версії 0.0.4, яка дозволяє отримувати детальну інформацію про ключові точки обличчя в режимі реального часу [5] (див. таблицю 2.5).

Таким чином, вибір моделей обумовлений необхідністю забезпечення балансу між точністю та швидкодією, що є критичним для програмно-апаратних систем, які працюють у режимі реального часу.

Загалом, обрані засоби розробки та моделі машинного навчання забезпечують можливість створення ефективної системи аналізу відеопотоку в режимі реального часу з урахуванням вимог автономності та швидкодії.

Таблиця 2.5 – Порівняння моделей аналізу обличчя

Засіб	Переваги	Недоліки	Доцільність використання
FaceMesh	Детальні ключові точки	Високе навантаження	Висока
Dlib	Стабільність, точність	Не для браузера	Середня
MediaPipe	Висока оптимізація, сучасність	Складність інтеграції	Висока

2.2 Загальна архітектура програмно-апаратної системи

Розроблена програмно-апаратна система призначена для локального аналізу відеопотоку в режимі реального часу з використанням сучасних методів комп’ютерного зору. Архітектура побудована за модульним принципом, що забезпечує її гнучкість, масштабованість та можливість подальшого вдосконалення.

Функціонування програмно-апаратної системи базується на послідовній обробці даних, що надходять з відеокамери, із застосуванням моделей машинного навчання та логічного аналізу [1]. Важливою особливістю є те, що всі обчислення виконуються локально на стороні клієнта, без використання віддалених серверів, що дозволяє мінімізувати затримки та підвищити рівень конфіденційності.

Система включає наступні основні компоненти (рис. 2.1):

1) модуль отримання відеопотоку;
2) модуль попередньої обробки;
3) модуль штучного інтелекту;
4) модуль логічного аналізу;
5) модуль відображення результатів.

Процес обробки відеоданих здійснюється у циклічному режимі, що забезпечує безперервний аналіз сцени.

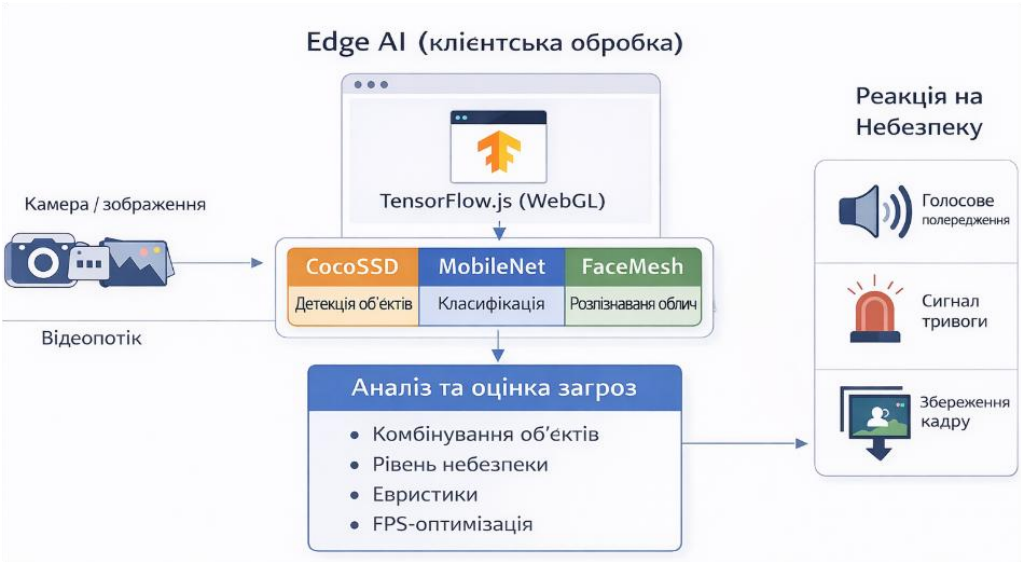


Рисунок 2.1 – Загальна архітектура програмно-апаратної системи

З наведеної схеми видно, що програмно-апаратна система реалізує послідовний підхід до обробки даних, у якому кожен модуль виконує чітко визначену функцію. Така організація забезпечує ефективну взаємодію між компонентами системи та спрощує її модифікацію [7].

2.3 Обґрунтування вибору апаратних засобів

Для забезпечення функціонування програмно-апаратної системи аналізу відеопотоку в режимі реального часу необхідним є використання відповідних апаратних засобів, здатних виконувати обробку відеоданих та запуск моделей машинного навчання (рис. 2.2).

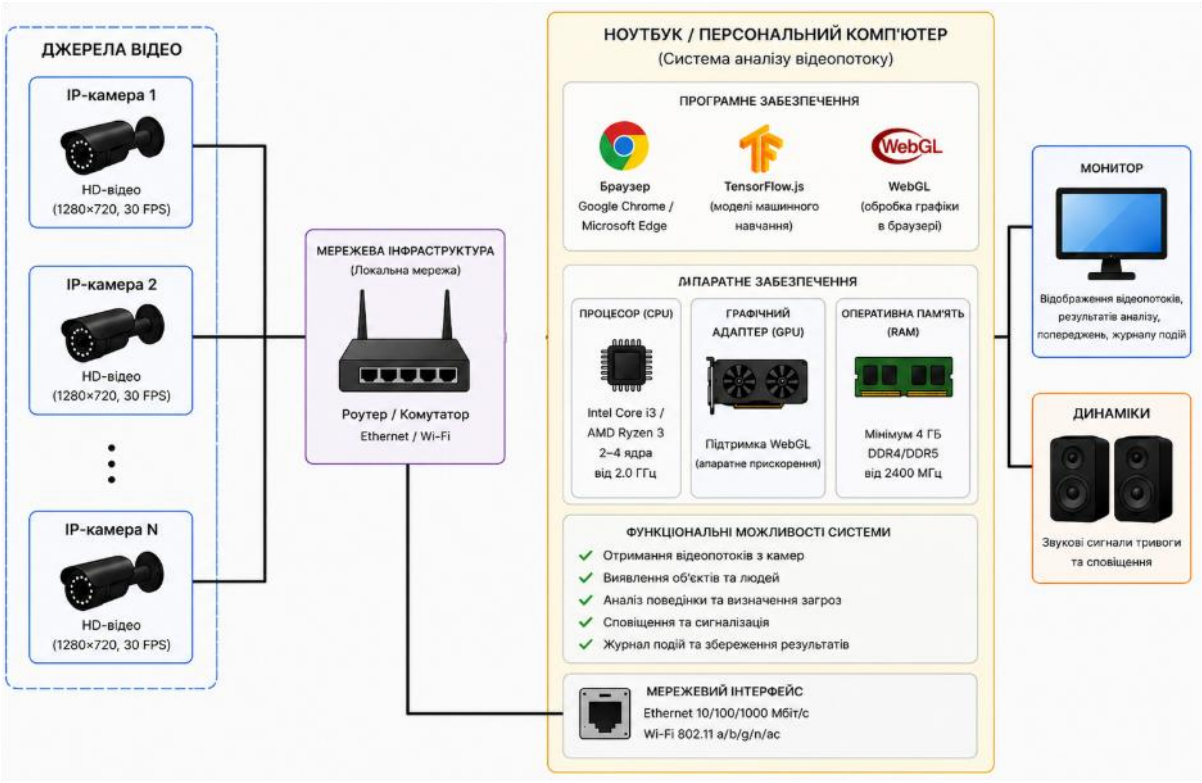


Рисунок 2.2 – Апаратна архітектура

Основними апаратними компонентами системи були обрані (таблиця 2.6):

1) персональний комп’ютер або ноутбук;
2) камера відеоспостереження;
3) монітор;
4) динаміки.

Таблиця 2.6 – Мінімальні апаратні вимоги системи

Компонент	Мінімальні вимоги
Процесор	Intel Core I3 / AMD Ryzen 3
Оперативна пам’ять	4 ГБ
Графічний адаптер	Підтримка WebGL
Камера	IP-камера або відеокамера з HD-відео
Мережевий інтерфейс	Ethernet / Wi-Fi
Браузер	Google Chrome / Microsoft Edge
Операційна система	Windows 10/11
Підключення до інтернету	Необхідне на етапі завантаження моделей

У сучасних системах комп’ютерного зору важливим фактором є можливість локальної обробки даних без використання серверної інфраструктури. Такий підхід дозволяє мінімізувати затримки передачі даних, підвищити швидкодію системи та забезпечити конфіденційність відеоінформації [2].

Для реалізації системи може використовуватись персональний комп’ютер або ноутбук із підтримкою сучасних веб-браузерів та технології апаратного прискорення WebGL. Використання графічного процесора дозволяє прискорити виконання моделей машинного навчання та забезпечити стабільну роботу системи в режимі реального часу [1].

У якості джерела відеопотоку можуть використовуватись як звичайні вебкамери, так і спеціалізовані IP-камери відеоспостереження. Для задач систем безпеки доцільним є використання IP-камер, оскільки вони забезпечують стабільну передачу відеоданих, високу якість зображення та можливість інтеграції у мережеву інфраструктуру.

У роботі розглянуто використання камери Hikvision DS-2CD2043G2-I (таблиця 2.7), яка підтримує передачу відеопотоку у високій роздільній здатності та може використовуватись у системах відеоспостереження реального часу. Дана камера забезпечує достатню якість відеоданих для задач детекції об’єктів, аналізу поведінки людини та визначення потенційно небезпечних ситуацій.

При виборі параметрів відеопотоку також враховується вплив роздільної здатності зображення на продуктивність програмно-апаратної системи. Зі збільшенням роздільної здатності відео зростає обсяг оброблюваних даних, що призводить до підвищення обчислювального навантаження на процесор та графічний адаптер. Це, у свою чергу, може зменшувати швидкість обробки кадрів та впливати на стабільність роботи системи в режимі реального часу. Практичні результати показують, що для забезпечення оптимального балансу між якістю зображення та швидкодією доцільно використовувати роздільну здатність рівня 720p, яка забезпечує достатню точність розпізнавання при збереженні прийнятного рівня продуктивності.

Таблиця 2.7 – Характеристики обраної камери відеоспостереження

Характеристика	Значення
Модель камери	Hikvision DS-2CD2043G2-I
Тип пристрою	IP-камера відеоспостереження
Максимальна роздільна здатність	2688 × 1520
Частота кадрів	До 30 FPS
Тип підключення	Ethernet
Підтримка мережі	TCP/IP
Наявність нічного режиму	Так
Тип встановлення	Внутрішнє / зовнішнє
Можливість роботи в реальному часі	Так
Сумісність із системами комп’ютерного зору	Так

Підключення камери до персонального комп’ютера або ноутбука може здійснюватись через локальну мережу Ethernet за допомогою маршрутизатора або безпосереднього мережевого підключення. Отримання відеопотоку виконується через мережевий інтерфейс із подальшою передачею кадрів до програмної системи аналізу.

Використання IP-камери дозволяє розміщувати пристрій спостереження на значній відстані від обчислювальної системи, що є важливим для задач охоронного відеоспостереження та моніторингу приміщень.

Практична реалізація та тестування програмно-апаратної системи виконувались на персональному ноутбуці з наступними характеристиками:

1) процесор AMD Ryzen 5 7200;
2) відеокарта NVIDIA GeForce RTX 2050;
3) оперативна пам’ять 16 ГБ.

Застосування даної апаратної конфігурації дозволило забезпечити стабільну обробку відеопотоку в режимі реального часу із використанням декількох моделей машинного навчання одночасно. Наявність дискретного графічного адаптера значно підвищує продуктивність обчислень, що особливо важливо при виконанні задач комп’ютерного зору та аналізу поведінки людини.

Отримані результати тестування підтверджують, що навіть на відносно доступному апаратному забезпеченні можлива ефективна реалізація локальних систем комп’ютерного зору без використання серверної інфраструктури.

При виборі апаратної платформи також враховувались вимоги до автономності роботи, енергоефективності та можливості функціонування системи без постійного підключення до мережі Інтернет.

Таким чином, обрані апаратні засоби забезпечують можливість стабільної роботи системи аналізу відеопотоку в режимі реального часу та відповідають вимогам локальної обробки даних.

У розробленій програмно-апаратній системі реалізовано прототип, який працює з одним джерелом відеопотоку. Водночас запропонована архітектура допускає масштабування на декілька IP-камер. У такому випадку кожен відеопотік обробляється окремим програмним модулем, а результати аналізу передаються до спільного модуля логічного аналізу та журналювання подій. Це дозволяє використовувати систему для моніторингу декількох приміщень або зон спостереження одночасно без суттєвих змін програмної архітектури.

2.4 Опис функціональних модулів системи

2.4.1 Модуль отримання відеопотоку

Модуль відповідає за доступ до відеокамери пристрою та передачу зображення для подальшої обробки. Для реалізації використовується стандартний веб-інтерфейс getUserMedia, що дозволяє отримувати відеопотік у режимі реального часу.

2.4.2 Модуль обробки зображень

Отриманий відеопотік передається у графічний елемент canvas, який використовується як буфер для подальшої обробки. На цьому етапі виконується

масштабування зображення та підготовка даних для подачі на вхід моделей машинного навчання.

2.4.3 Модуль штучного інтелекту

Модуль штучного інтелекту реалізує багаторівневий аналіз зображення. Його особливістю є використання декількох моделей, кожна з яких виконує окрему задачу.

Поєднання моделей дозволяє:

1) отримувати інформацію про об’єкти у кадрі;
2) визначати контекст сцени;
3) аналізувати поведінку людини;
4) підвищувати точність прийняття рішень за рахунок комбінування результатів.

Такий підхід є більш ефективним порівняно з використанням однієї моделі, оскільки дозволяє враховувати різні аспекти аналізу зображення.

2.4.4 Модуль логічного аналізу

Модуль логіки відповідає за інтерпретацію результатів, отриманих від моделей штучного інтелекту. На основі виявлених об’єктів та поведінкових характеристик формуються висновки про стан сцени.

У системі реалізовано механізм визначення рівня загрози, який базується на аналізі комбінацій об’єктів та поведінкових патернів. Наприклад, одночасна присутність людини та небезпечного предмета може свідчити про потенційну загрозу.

2.4.5 Модуль взаємодії з користувачем

Даний модуль забезпечує відображення результатів роботи системи, включаючи:

1) візуалізацію виявлених об’єктів;
2) текстовий опис сцени;
3) індикацію рівня загрози;
4) звукові сповіщення.

Інтерфейс користувача реалізований у вигляді веб-сторінки, що забезпечує простоту використання та доступність системи.

2.5 Алгоритм роботи програмно-апаратної системи

Алгоритм функціонування програмно-апаратної системи реалізовано у вигляді безперервного циклу обробки відеокадрів (рис. 2.3). Кожен кадр проходить послідовні етапи аналізу, що дозволяє системі реагувати на зміни сцени у реальному часі [2].

Основні етапи роботи системи:

1) Ініціалізація моделей машинного навчання та підготовка середовища виконання [1].
2) Отримання відеопотоку з камери користувача.
3) Захоплення поточного кадру та його передача у буфер обробки.
4) Попередня обробка зображення (масштабування, нормалізація) [7].
5) Передача даних до моделей штучного інтелекту.
6) Отримання результатів (об’єкти, координати, класи).
7) Аналіз отриманої інформації логічним модулем.
8) Визначення контексту сцени та рівня загрози.
9) Виведення результатів користувачу.
10) Перехід до обробки наступного кадру.

Такий підхід дозволяє забезпечити безперервність роботи системи та її адаптацію до змін у відеопотоці.

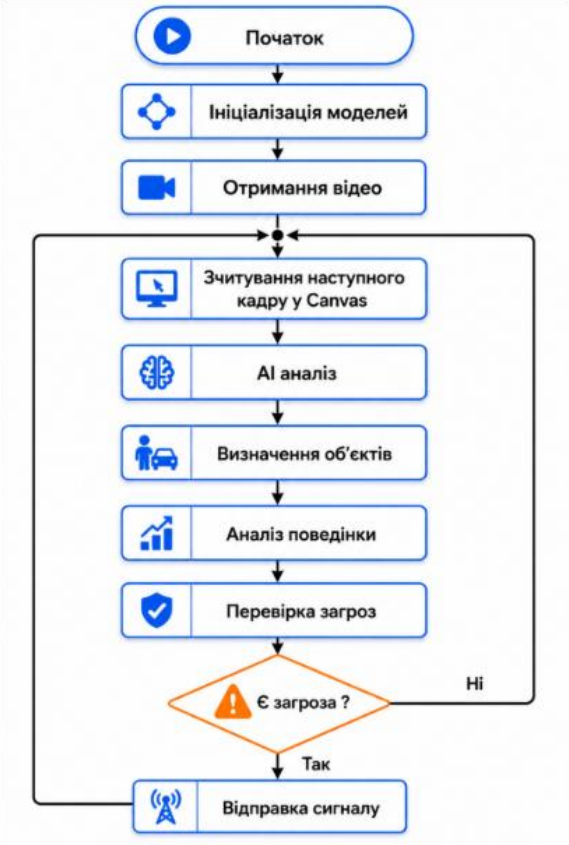


Рисунок 2.3 – Алгоритм роботи програмно-апаратної системи

Представлений алгоритм відображає логіку послідовної обробки даних та демонструє циклічний характер роботи, що є необхідною умовою для функціонування в режимі реального часу [1].

2.6 Логіка визначення потенційно небезпечних ситуацій

Однією з ключових задач програмно-апаратної системи є не лише виявлення об’єктів, але й інтерпретація їх взаємодії та поведінки [1, 6] (рис. 2.4). Для цього використовується механізм логічного аналізу, що базується на комбінуванні результатів роботи моделей штучного інтелекту [4, 7].

Визначення загроз здійснюється на основі наступних принципів:

1) аналіз окремих об’єктів;
2) аналіз комбінацій об’єктів;
3) аналіз поведінкових характеристик;
4) врахування часових факторів.

До основних сценаріїв загроз належать:

1) наявність потенційно небезпечних предметів;
2) взаємодія людини з небезпечними об’єктами;
3) аномальна поведінка (падіння, різкі рухи);
4) поява активності у нетиповий час;
5) перевищення кількості людей у кадрі.

Для кожного сценарію формується рівень загрози, що дозволяє системі диференціювати події за ступенем небезпеки та відповідним чином реагувати.

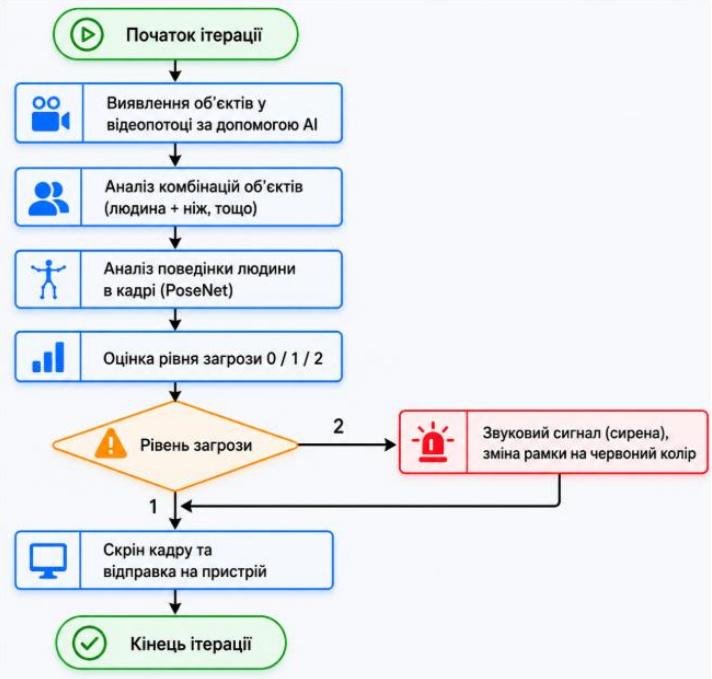


Рисунок 2.4 – Логіка визначення потенційно небезпечних ситуацій

Наведена схема демонструє процес переходу від первинних даних, отриманих від моделей, до формування кінцевого рішення щодо рівня загрози. Такий підхід дозволяє підвищити точність аналізу та зменшити кількість хибних спрацювань [8].

Висновки до розділу 2

У результаті виконання даного розділу було розроблено архітектуру програмно-апаратної системи локального аналізу відеопотоку та визначено її основні функціональні компоненти. Сформовано алгоритм роботи системи та описано механізм визначення потенційно небезпечних ситуацій.

Проведений аналіз показав, що використання модульного підходу та поєднання методів комп’ютерного зору з логічним аналізом дозволяє створити ефективну систему, здатну працювати в режимі реального часу.

Крім того, запропонована архітектура має достатній рівень масштабованості та може бути розширена шляхом інтеграції додаткових моделей штучного інтелекту, нових сценаріїв аналізу або підключення зовнішніх систем сповіщення та моніторингу. Це дозволяє адаптувати систему до різних умов використання та типів задач.

Отримані результати є основою для подальшої реалізації програмного забезпечення, що буде детально розглянуто у наступному розділі.

3 РЕАЛІЗАЦІЯ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ

3.1 Загальна характеристика програмної реалізації

У рамках кваліфікаційної роботи було реалізовано програмно-апаратну систему аналізу відеопотоку, яка функціонує у середовищі веб-браузера та забезпечує обробку відеоданих у режимі реального часу.

Програмна реалізація виконана у вигляді веб-додатку, що не потребує встановлення додаткового програмного забезпечення та може працювати на різних платформах. Основою системи є використання мови програмування JavaScript, що забезпечує інтеграцію з браузерними API для роботи з відео та графікою.

Програмно-апаратна система використовує можливості сучасних веб-технологій для доступу до камери пристрою, обробки відеопотоку та відображення результатів аналізу. Отримання відеоданих здійснюється за допомогою відповідних засобів браузера, після чого кожен кадр передається на обробку.

Обробка відеопотоку виконується у декілька етапів: захоплення кадру, попередня обробка, аналіз за допомогою моделей машинного навчання та формування результатів. Для візуалізації результатів використовується графічний інтерфейс, який відображає знайдені об’єкти, додаткову інформацію та попередження про потенційно небезпечні ситуації.

Реалізована програмно-апаратна система забезпечує автономну роботу без використання віддалених серверів, що дозволяє досягти мінімальних затримок та підвищити рівень конфіденційності даних.

3.2 Реалізація модулів обробки відеопотоку

Програмно-апаратна система аналізу відеопотоку реалізована у вигляді набору взаємопов’язаних модулів, кожен з яких виконує окрему функцію в

процесі обробки даних. Такий підхід дозволяє забезпечити гнучкість системи, спростити її розширення та підвищити ефективність роботи (лістинг 3.1).

3.2.1 Модуль отримання відеопотоку

Отримання відеоданих здійснюється за допомогою використання можливостей веб-браузера для доступу до камери пристрою. Відеопотік передається у відповідний елемент інтерфейсу, що дозволяє відображати зображення в режимі реального часу (лістинг 3.1).

```
navigator.mediaDevices
.getUserMedia({ video: true })
.then((stream) => (camera.srcObject = stream));
```

Лістинг 3.1 – Ініціалізація доступу до камери

Наведений фрагмент коду забезпечує отримання відеопотоку з камери та його відображення у веб-додатку.

3.2.2 Модуль обробки кадрів

Для обробки відеоданих використовується графічний елемент canvas, який дозволяє працювати з окремими кадрами відео. Кожен кадр копіюється з відеопотоку та передається на подальший аналіз (лістинг 3.2).

Цей фрагмент коду дозволяє отримати поточний кадр відео для подальшої обробки.

Обробка виконується циклічно, що забезпечує наближений до реального часу режим роботи системи. Для оптимізації продуктивності застосовується часткове пропускання кадрів.

```
 async function loop() {
          if (!realtime) return;

          setCanvasSize();
          ctx.drawImage(camera, 0, 0);

          const start = performance.now();

          await analyzeFrame();

          fpsEl.innerText =
            'FPS: ' + Math.round(1000 / (performance.now() -
start));

          setTimeout(() => {
            rafId = requestAnimationFrame(loop);
          }, 50);

          if (threatLevel >= 2) {
            startSiren();
          } else {
            stopSiren();
          }
          updateUIWarning();
        }
```

Лістинг 3.2 – Отримання кадру з відеопотоку

3.2.3 Модуль керування циклом обробки

Цикл обробки реалізовано з використанням механізму, що дозволяє синхронізувати обробку кадрів із оновленням зображення (лістинг 3.3).

```
async function loop() {
if (!realtime) return;

ctx.drawImage(camera, 0, 0);

await analyzeFrame();

requestAnimationFrame(loop);
}
```

Лістинг 3.3 – Основний цикл обробки відеопотоку

Такий підхід дозволяє забезпечити безперервну обробку відеопотоку та підтримувати стабільну частоту кадрів.

3.2.4 Модуль аналізу відеоданих

Аналіз зображення здійснюється із використанням моделей машинного навчання, які виконують детекцію об’єктів, класифікацію та інші задачі (лістинг 3.4).

```
async function analyzeFrame() {
if (!modelsLoaded) return;

try {
// Детекція об'єктів
const predictions = await cocoModel.detect(canvas);
// Отримання класів об'єктів
const classes = predictions.map(p => p.class);
// Відображення результатів
predictions.forEach(p => {
ctx.strokeStyle = 'lime';
ctx.strokeRect(...p.bbox);
ctx.fillText(p.class, p.bbox[0], p.bbox[1] - 5);
});
console.log('Detected objects:', classes);
} catch (e) {
console.error(e);
}
}
```

Лістинг 3.4 – Виклик моделі детекції об’єктів

Даний фрагмент коду дозволяє визначити об’єкти, що присутні на зображенні, та отримати їх характеристики.

3.2.5 Модуль візуалізації результатів

Результати аналізу відображаються безпосередньо на зображенні у вигляді графічних елементів (лістинг 3.5).

```
ctx.strokeRect(...p.bbox);
ctx.fillText(p.class, p.bbox[0], p.bbox[1] - 5);
```

Лістинг 3.5 – Відображення результатів детекції

Цей код дозволяє відобразити межі знайдених об’єктів та підписати їх.

Додатково результати виводяться у текстовому вигляді у відповідному блоці інтерфейсу.

3.2.6 Модуль оптимізації продуктивності

Для забезпечення стабільної роботи програмно-апаратної системи у режимі реального часу реалізовано механізми оптимізації (лістинг 3.6).

```
if (frameSkip % 5 === 0) {
lastRes.mobilePreds = await mobilenetModel.classify(canvas);
}
```

Лістинг 3.6 – Пропуск кадрів для оптимізації

Такий підхід дозволяє зменшити навантаження на програмно-апаратну систему шляхом зниження частоти виконання окремих операцій.

3.2.7 Модуль взаємодії з користувачем

Програмно-апаратна система передбачає можливість взаємодії з користувачем через графічний інтерфейс (лістинг 3.7).

```
realtimeToggle.onclick = () => {
realtime = !realtime;
}
```

Лістинг 3.7 – Керування режимом реального часу

Цей фрагмент коду дозволяє користувачу керувати режимом роботи програмно-апаратної системи.

Реалізація модулів обробки відеопотоку дозволила створити ефективну програмно-апаратну систему, здатну виконувати аналіз відеоданих у режимі реального часу. Модульна структура забезпечує можливість подальшого розширення функціональності системи та інтеграції нових алгоритмів.

3.3 Інтеграція моделей машинного навчання

Для забезпечення функціонування програмно-апаратної системи аналізу відеопотоку було реалізовано інтеграцію декількох моделей машинного навчання, кожна з яких виконує окрему задачу обробки зображення. Використання декількох моделей дозволяє забезпечити комплексний аналіз сцени та підвищити інформативність результатів.

У програмно-апаратній системі використано моделі COCO-SSD, MobileNet, FaceMesh та PoseNet [3-6]. Їх інтеграція реалізована за допомогою бібліотеки TensorFlow.js [2], що дозволяє виконувати обробку безпосередньо у браузері.

3.3.1 Інтеграція моделі детекції об’єктів

Для визначення об’єктів на зображенні використовується модель COCO-SSD [3]. Модель дозволяє визначати тип об’єкта та його координати на зображенні.

Перед початком роботи система виконує завантаження моделі у пам’ять браузера (лістинг 3.8).

```
cocoModel = await cocoSsd.load();
...
lastRes.cocoPreds = await cocoModel.classify(canvas);
```

Лістинг 3.8 – Завантаження моделі детекції об’єктів та аналіз кадру

Після завантаження модель використовується для аналізу кожного кадру відеопотоку.

Результатом роботи моделі є масив знайдених об’єктів із відповідними координатами та рівнем достовірності.

3.3.2 Інтеграція моделі класифікації сцени

Для загального аналізу сцени використовується модель MobileNet [4]. Модель дозволяє визначити загальний вміст кадру та сформувати додатковий опис сцени (лістинг 3.9).

```
mobilenetModel = await mobilenet.load();
...
lastRes.mobilePreds = await mobilenetModel.classify(canvas);
```

Лістинг 3.9 – Завантаження моделі MobileNet та класифікація зображень

Після завантаження модель використовується для класифікації поточного кадру.

Для оптимізації продуктивності класифікація виконується не для кожного кадру, а через певний інтервал.

3.3.3 Інтеграція моделі аналізу обличчя

Для аналізу обличчя та визначення ключових точок використовується модель FaceMesh [5] (лістинг 3.10).

```
faceModel = await facemesh.load();
...
lastRes.faces = await faceModel.estimateFaces(canvas);
```

Лістинг 3.10 – Завантаження моделі FaceMesh та аналіз обличчя

Модель дозволяє отримувати координати ключових точок обличчя, що використовуються для визначення напрямку погляду та положення голови.

Отримані результати використовуються для подальшого аналізу поведінки людини. Зокрема, модель FaceMesh повертає масив координат ключових точок обличчя (landmarks), які описують положення очей, носа, рота та інших частин.

На основі цих координат у системі реалізовано визначення напрямку погляду та орієнтації голови. Наприклад, шляхом обчислення відстані між характерними точками обличчя (лівим та правим оком) можна оцінити, чи дивиться людина прямо перед собою, чи відвернула голову вбік (лістинг 3.11).

```
function analyzeFace(face) {
if (!face) return null;

const dx = face.scaledMesh[263][0] - face.scaledMesh[33][0];

if (dx > 60) return 'Дивиться вперед';
if (dx < 40) return 'Відвернув голову';

return 'Нейтральне положення';
}
```

Лістинг 3.11 – Визначення напрямку погляду на основі ключових точок

Отримана інформація інтегрується з результатами інших моделей (детекції об’єктів та оцінки пози людини) та використовується модулем логічного аналізу для формування узагальненого опису сцени. Це дозволяє підвищити точність інтерпретації поведінки користувача та більш ефективно визначати потенційно небезпечні ситуації.

3.3.4 Інтеграція моделі визначення пози людини

Для визначення положення людини у кадрі використовується модель PoseNet [6]. Модель дозволяє визначати координати основних частин тіла та аналізувати позу людини (лістинг 3.12).

```
poseModel = await posenet.load();
...
const pose = await poseModel.estimateSinglePose(canvas);
```

Лістинг 3.12 – Завантаження моделі PoseNet та визначення пози людини

Після завантаження модель використовується для визначення ключових точок тіла.

Результати роботи моделі використовуються для аналізу поведінки людини та виявлення потенційно небезпечних ситуацій. Модель PoseNet повертає набір ключових точок тіла (keypoints), таких як голова, руки, ноги та інші суглоби, кожна з яких має координати у просторі зображення.

На основі цих координат у системі реалізовано ряд евристичних алгоритмів для інтерпретації поведінки людини. Зокрема, визначається можливе падіння, різкі рухи або підняття рук.

3.3.5 Спільна робота моделей

Інтеграція декількох моделей дозволяє забезпечити комплексний аналіз відеопотоку. Кожна модель виконує окрему задачу, а результати їх роботи об’єднуються для формування загального опису сцени.

Наприклад, система може одночасно:

1) визначати наявність людини у кадрі;
2) аналізувати положення тіла;
3) визначати напрямок погляду;
4) класифікувати об’єкти навколо;
5) формувати повідомлення про потенційні загрози.

Однак ключовою особливістю системи є не лише отримання окремих результатів, а їх комбіноване використання для визначення небезпечних ситуацій.

Зокрема, у системі реалізовано логічний модуль, який аналізує поєднання результатів різних моделей та формує висновки про стан сцени (лістинг 3.13). Наприклад:

1) якщо модель детекції об’єктів (COCO-SSD) визначає наявність людини та потенційно небезпечного предмета (наприклад, ножа), система формує повідомлення про загрозу;

2) якщо модель PoseNet фіксує аномальне положення тіла (наприклад, падіння), генерується попередження про можливу небезпечну ситуацію;

3) якщо аналіз руху показує різке переміщення, це може інтерпретуватись як підозріла активність;

4) якщо модель FaceMesh визначає, що людина не контролює напрямок погляду або поводиться нетипово, це використовується як додатковий фактор оцінки ситуації.

```
function detectDanger(classes, pose, face) {
const set = new Set(classes);
// Людина + небезпечний предмет
if (set.has('person') && set.has('knife')) {
return 'Небезпечна ситуація: людина з ножем';
}
// Падіння
const fall = detectFall(pose);
if (fall) {
return fall;
}
// Різкий рух
const movement = detectMovement(pose);
if (movement) {
return movement;
}
return null;
}
```

Лістинг 3.13 – Функція визначення небезпечних ситуацій

Таким чином, визначення небезпечних ситуацій здійснюється не на основі окремого параметра, а шляхом аналізу комбінації ознак, отриманих з різних моделей. Це дозволяє значно підвищити точність роботи системи та зменшити кількість помилкових спрацювань.

Запропонований підхід забезпечує більш глибоке розуміння сцени та дозволяє ефективно використовувати можливості сучасних методів комп’ютерного зору у задачах безпеки

3.3.6 Оптимізація роботи моделей

Оскільки одночасне використання декількох моделей створює значне навантаження на програмно-апаратну систему, було реалізовано механізми оптимізації. Для різних моделей використовується різна частота оновлення результатів (лістинг 3.14).

```
if (frameSkip % 5 === 0) {
lastRes.mobilePreds = await mobilenetModel.classify(canvas);
}

if (frameSkip % 7 === 0) {
lastRes.faces = await faceModel.estimateFaces(canvas);
}
```

Лістинг 3.14 – Оптимізація частоти виклику моделей

Такий підхід дозволяє зменшити навантаження на систему та забезпечити стабільну роботу у режимі реального часу.

Інтеграція моделей машинного навчання дозволила реалізувати комплексну програмно-апаратну систему аналізу відеопотоку, здатну виконувати детекцію об’єктів, класифікацію сцени, аналіз обличчя та визначення пози людини. Використання декількох моделей забезпечує підвищення інформативності програмно-апаратної системи та розширення її функціональних можливостей.

3.4 Реалізація механізму визначення потенційно небезпечних ситуацій

Однією з ключових функцій розробленої програмно-апаратної системи є автоматичне виявлення потенційно небезпечних ситуацій у відеопотоці. Даний механізм базується на поєднанні результатів роботи моделей комп’ютерного зору та евристичних правил аналізу сцени [1, 6].

Загальна логіка роботи полягає у послідовному аналізі об’єктів, виявлених на зображенні, та визначенні їх комбінацій, які можуть свідчити про ризикову або підозрілу ситуацію.

На першому етапі використовується модель COCO-SSD [3], яка визначає об’єкти на кожному кадрі відеопотоку. Результатом її роботи є набір об’єктів із координатами та класами, наприклад: person, knife, cell phone, cup тощо.

Далі отриманий список класів використовується для подальшого логічного аналізу.

У системі реалізовано набір правил, які дозволяють визначати потенційно небезпечні ситуації на основі комбінацій об’єктів (лістинг 3.15).

Наприклад:

1) одночасна наявність об’єктів типу person і knife інтерпретується як потенційна загроза;

2) виявлення вогню (fire) класифікується як критична подія;

3) повторювана поява підозрілих об’єктів підвищує рівень загрози.

```
function detectSuspicious(classes) {
const set = new Set(classes);
if (set.has('knife') && set.has('person')) {
return '🚨 УВАГА: людина з ножем!';
}
if (set.has('fire')) {
return '🔥 УВАГА: вогонь!';
}
return null;
}
```

Лістинг 3.15 – Визначення потенційної загрози

Даний підхід дозволяє швидко ідентифікувати базові сценарії небезпечної поведінки без використання складних аналітичних моделей.

Для більш точного реагування у програмно-апаратній системі реалізовано поняття рівня загрози. Він змінюється залежно від частоти повторення небезпечних подій.

Якщо певна подія повторюється декілька разів, рівень загрози підвищується до критичного (лістинг 3.16).

```
function updateThreatLevel(reason) {
  dangerCounter[reason] = (dangerCounter[reason] || 0) + 1;

  if (dangerCounter[reason] >= 2) {
    threatLevel = 2;
  } else {
    threatLevel = 1;
  }
}
```

Лістинг 3.16 – Підвищення рівня загрози

Таким чином, програмно-апаратна система враховує не лише факт появи об’єкта, але й його повторюваність у часі.

Для підвищення точності оцінки ситуації використовується модель PoseNet. Вона дозволяє визначати положення тіла людини та аналізувати її поведінкові характеристики.

Отримані ключові точки можуть використовуватись для визначення напрямку руху, положення тіла або нетипових поз, що можуть свідчити про небезпечну ситуацію (лістинг 3.17).

У разі виявлення критичної ситуації система може:

1) змінювати візуальне оформлення інтерфейсу;
2) виділяти об’єкти червоним контуром;
3) генерувати звуковий сигнал тривоги;
4) створювати автоматичний знімок кадру.

```
if (threatLevel >= 2) {
canvas.classList.add('danger-frame');
startSiren();
}
```

Лістинг 3.17 – Активація сигналу тривоги

Цей механізм дозволяє миттєво привернути увагу користувача до потенційно небезпечної ситуації.

Реалізований механізм визначення потенційно небезпечних ситуацій базується на поєднанні моделей машинного навчання та правил логічного аналізу. Такий підхід дозволяє досягти балансу між швидкодією та точністю визначення загроз.

Програмно-апаратна система здатна працювати в режимі реального часу та оперативно реагувати на зміну сцен у відеопотоці, що робить її ефективною для використання у задачах відеоспостереження та безпеки.

3.5 Тестування та оцінка ефективності системи

Для перевірки працездатності розробленої програмно-апаратної системи було проведено тестування основних функціональних можливостей програмно-апаратного комплексу аналізу відеопотоку. Основною метою тестування була перевірка коректності роботи моделей комп’ютерного зору, швидкодії системи та здатності визначати потенційно небезпечні ситуації у режимі реального часу.

Тестування проводилось на персональному комп’ютері із використанням вебкамери та локального виконання моделей машинного навчання у браузері. У процесі дослідження оцінювались:

1) швидкість обробки відеопотоку – оцінювалась за кількістю кадрів, що обробляються за секунду (FPS). У процесі тестування середнє значення становило 50–55 FPS залежно від навантаження та кількості активних моделей.;

2) коректність детекції об’єктів – визначалась шляхом порівняння результатів роботи моделі з фактичним вмістом сцени. Оцінювалась як відсоток правильно розпізнаних об’єктів від загальної кількості (точність детекції).;

3) стабільність роботи системи – оцінювалась за тривалістю безперервної роботи без збоїв, помилок або зависань. Додатково враховувалась відсутність витоків пам’яті та коректність повторного запуску моделей.;

4) швидкість реагування на небезпечні ситуації – визначалась як час між появою небезпечного об’єкта або події у кадрі та формуванням відповідного повідомлення системою. У середньому цей показник становить менше 1 секунди.;

5) ефективність механізму журналювання подій – оцінювалась за здатністю системи коректно фіксувати події, зберігати історію змін та забезпечувати доступ до попередніх результатів аналізу. Враховувалась повнота записів та їх актуальність.

3.5.1 Тестування інтерфейсу користувача програмно-апаратної системи

У процесі роботи програмно-апаратна система забезпечує відображення відеопотоку, результатів аналізу сцени, журналу подій та технічної інформації про швидкодію системи (рис. 3.1).

У правій частині інтерфейсу відображаються:

1) результати аналізу сцени;
2) визначені об’єкти;
3) інформація про положення людини;
4) журнал подій;
5) поточне значення FPS.

Під час тестування середнє значення FPS становило 50–55 кадрів за секунду, що підтверджує можливість роботи системи у режимі реального часу.

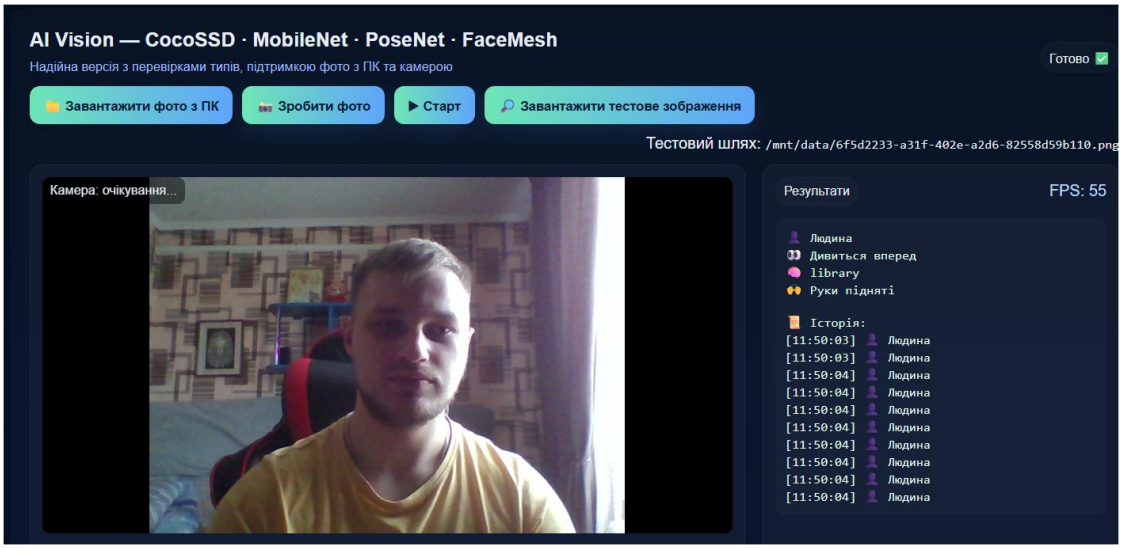


Рисунок 3.1 – Загальний вигляд інтерфейсу програмно-апаратної системи

Обчислення даного показника здійснювалось на етапі обробки відеокадру після його відображення на canvas та виконання основних алгоритмів аналізу, включаючи детекцію об’єктів та часткову обробку результатів моделей.

3.5.2 Тестування детекції об’єктів

Для перевірки роботи моделі COCO-SSD було проведено тестування із різними типами об’єктів. Програмно-апаратна система успішно виконувала детекцію людини, мобільного телефону та інших об’єктів у кадрі (рис. 3.2).

У процесі тестування програмно-апаратна система формувала обмежувальні рамки навколо об’єктів та виводила назву визначеного класу. Отримані результати підтверджують коректність роботи алгоритму детекції.

3.5.3 Тестування аналізу поведінки людини

Для аналізу положення людини використовувались моделі FaceMesh та PoseNet. Програмно-апаратна система визначала напрямок погляду користувача, положення рук та загальну позу людини (рис. 3.3).

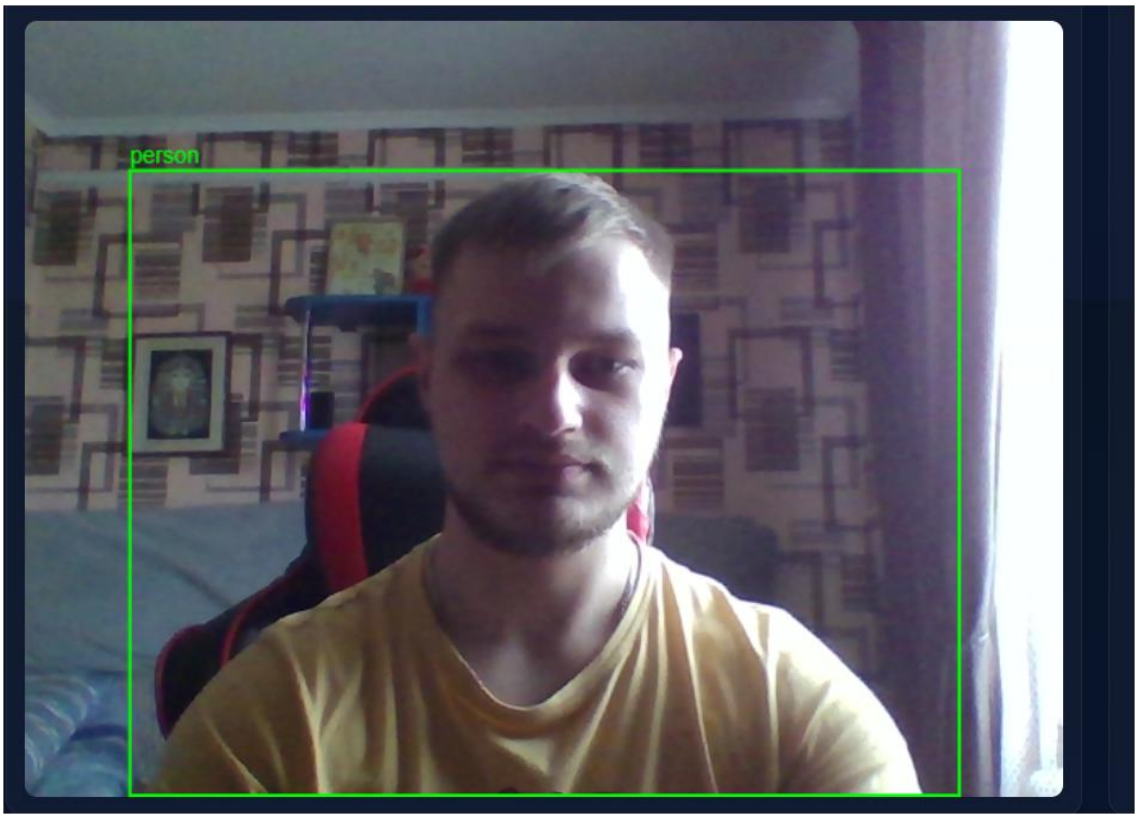


Рисунок 3.2 – Детекція людини у кадрі

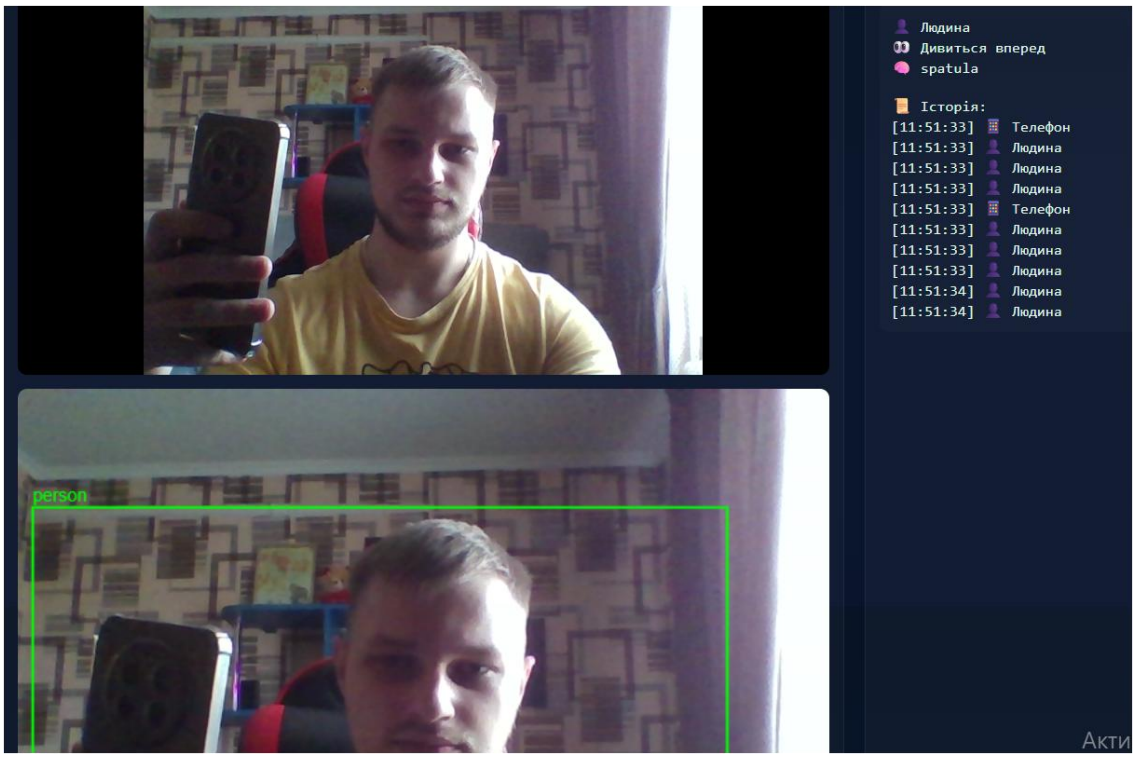


Рисунок 3.3 – Виявлення мобільного телефону та аналіз положення людини

У процесі тестування програмно-апаратна система коректно визначила:

1) наявність людини;
2) напрямок погляду;
3) мобільний телефон у кадрі.

Отримані результати демонструють можливість комплексного аналізу сцени із використанням декількох моделей машинного навчання одночасно.

3.5.4 Тестування механізму визначення небезпечних ситуацій

Окремий етап тестування був присвячений перевірці роботи механізму визначення потенційно небезпечних ситуацій.

Для цього у кадрі імітувалась поява предмета, схожого на холодну зброю. У разі виявлення небезпечної ситуації система (рис. 3.4):

1) активувала режим тривоги;
2) змінювала візуальне оформлення інтерфейсу;
3) створювала автоматичні скріншоти події;
4) продовжувала журналювання інцидентів.

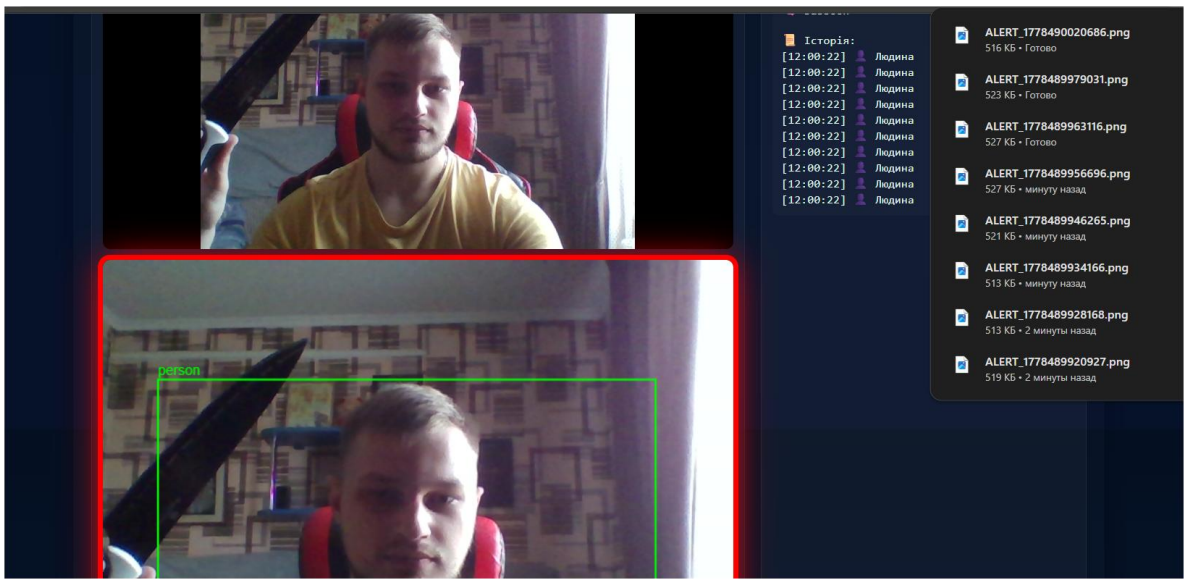


Рисунок 3.4 – Спрацювання механізму визначення потенційно небезпечної ситуації

Під час тестування було встановлено, що для стабільного виявлення об’єкта необхідна його присутність у кадрі протягом приблизно 0,3–0,8 секунди, що відповідає 15–40 кадрам при середній швидкості обробки 50–55 FPS.

У процесі тестування було встановлено, що використання попередньо навчених безкоштовних моделей машинного навчання може призводити до нестабільного визначення окремих об’єктів у складних умовах освітлення або при частковому перекритті об’єкта (таблиця 3.1). Зокрема, система не завжди коректно визначає потенційно небезпечні предмети з першого кадру, що пояснюється обмеженнями навчальних вибірок та спрощеними архітектурами моделей.

Таблиця 3.1 – Результати тестування

Функція	Кількість успішних спрацювань з 10	Відсоток успішності, %
Виявлення людини	10	100
Виявлення ноутбука	9	90
Виявлення мобільного телефона	7	70
Виявлення чашки	6	60
Виявлення напрямку погляду	9	90
Виявлення піднятих рук	8	80
Виявлення різкого руху	8	80
Виявлення падіння	7	70
Виявлення ножа	4	40
Формування попередження про небезпеку	4	40

Проте навіть за таких умов система здатна повторно аналізувати відеопотік та виявляти небезпечні ситуації протягом наступних кадрів, що забезпечує достатній рівень ефективності для задач локального відеоспостереження.

3.5.5 Аналіз продуктивності програмно-апаратної системи

Під час тестування було встановлено, що локальна обробка відеопотоку забезпечує значно менші затримки у порівнянні з хмарними системами.

Середня продуктивність системи наведена у таблиці 3.2.

Таблиця 3.2 – Результати тестування продуктивності програмно-апаратної системи

Параметр	HD	Full HD
Середній FPS	50-55 FPS	25-30 FPS
Середня затримка обробки	30-80 мс	60-80 мс
Режим роботи	Реальний час	
Тип обробки	Локальна	
Кількість одночасно активних моделей	4	
Наявність мережевих затримок	Відсутня	
Автоматичне створення скріншотів	Так	
Підтримка журналу подій	Так	

У результаті проведеного тестування було підтверджено працездатність розробленої програмно-апаратної системи аналізу відеопотоку у режимі реального часу. Програмно-апаратна система успішно виконує детекцію об’єктів, аналіз поведінки людини та визначення потенційно небезпечних ситуацій.

Отримані результати демонструють ефективність використання локальної архітектури та можливість застосування програмно-апаратної системи у задачах відеоспостереження, моніторингу та безпеки.

3.6 Обґрунтування ефективності обраної архітектури

Ефективність розробленої програмно-апаратної системи значною мірою визначається особливостями обраної архітектури, яка базується на локальній обробці відеопотоку, модульному принципі побудови та використанні веб-технологій.

На відміну від традиційних хмарних систем відеоспостереження, запропоноване рішення виконує всі обчислення безпосередньо на пристрої користувача. Такий підхід дозволяє мінімізувати затримки, що є критично важливим для систем реального часу. Відсутність необхідності передавання відеопотоку на віддалені сервери забезпечує швидке реагування програмно-апаратної системи на зміни сцени та дозволяє миттєво формувати повідомлення про потенційно небезпечні ситуації.

Важливою перевагою локальної архітектури є також підвищений рівень конфіденційності. Усі відеодані залишаються на пристрої користувача та не передаються стороннім сервісам, що знижує ризик витоку інформації та підвищує рівень безпеки.

Використання веб-технологій забезпечує кросплатформеність. Робота через браузер дозволяє запускати програмно-апаратну систему на різних операційних системах без необхідності встановлення спеціалізованого програмного забезпечення. Це спрощує розгортання системи та забезпечує її доступність для широкого кола користувачів.

Однією з головних переваг локальної архітектури є зменшення затримок під час аналізу відеопотоку (таблиця 3.3). У традиційних хмарних системах значна частина часу витрачається на передачу даних через мережу Інтернет та обробку інформації на віддалених серверах. У середньому затримка таких систем може становити від 300 до 1000 мс залежно від швидкості мережі та навантаження серверної інфраструктури.

У розробленій програмно-апаратній системі обробка виконується локально на пристрої користувача, що дозволяє зменшити середню затримку до

30–80 мс. Таким чином, швидкість реакції є приблизно у 5–10 разів вищою порівняно з хмарними рішеннями.

Крім того, локальна архітектура дозволяє значно знизити навантаження на мережу. Передача відеопотоку у Full HD якості може потребувати від 3 до 8 Мбіт/с мережевого трафіку, тоді як у запропонованій програмно-апаратній системі постійна передача відеопотоку на сервер відсутня. Це дозволяє зменшити використання мережевих ресурсів більш ніж на 90 %.

Ще однією перевагою є використання апаратного прискорення WebGL, яке дозволяє частково переносити обчислення на графічний процесор. Завдяки цьому підвищується швидкість виконання моделей машинного навчання та забезпечується можливість одночасної роботи декількох алгоритмів комп'ютерного зору.

Завдяки використанню апаратного прискорення WebGL та локального виконання моделей машинного навчання програмно-апаратна система забезпечує продуктивність на рівні 50–55 FPS, що перевищує середні показники багатьох веб-орієнтованих систем відеоаналізу, де продуктивність зазвичай становить 15–30 FPS.

Таблиця 3.3 – Порівняння локальної та хмарної архітектури

Характеристика	Локальна система	Хмарна система
Середня затримка	30-80 мс	300-1000 мс
Швидкість реакції	Висока	Середня
Мережевий трафік	Мінімальний	Високий
Використання Інтернету	Лише для завантаження моделей	Постійне
Конфіденційність	Висока	Середня
Продуктивність	50-55 FPS	15-30 FPS
Автономність	Висока	Низька

Для більш обґрунтованої оцінки ефективності запропонованої архітектури доцільно використовувати кількісні показники, отримані під час тестування системи.

Основними показниками ефективності є:

1) швидкість обробки відеопотоку (FPS);
2) затримка обробки кадру;
3) швидкість реагування на зміну сцени.

Під час експериментальних досліджень було встановлено, що розроблена система забезпечує продуктивність на рівні 50–55 кадрів за секунду при середній затримці обробки 30–80 мс. У той же час, відповідно до аналізу відкритих джерел та технічної документації хмарних сервісів комп'ютерного зору, середні значення для подібних систем становлять 15–30 FPS при затримках 300–1000 мс.

З метою узагальнення отриманих результатів введено коефіцієнт ефективності:

$$E = \frac{FPS}{T}$$

де FPS – частота обробки кадрів,

T – середній час затримки обробки кадру.

Для розробленої програмно-апаратної системи:

$$E \approx \frac{50}{0.05} = 1000$$

Для хмарних систем:

$$E \approx \frac{20}{0.5} = 40$$

Отримані результати свідчать про те, що запропонована локальна система є приблизно у 20–25 разів ефективнішою за традиційні хмарні рішення за критерієм швидкодії та затримки.

Кількісна оцінка підтверджує доцільність використання локальної архітектури для задач аналізу відеопотоку в режимі реального часу.

Модульна структура програмно-апаратної системи також позитивно впливає на ефективність архітектури. Кожна модель виконує окрему функцію:

1) COCO-SSD відповідає за детекцію об’єктів;
2) MobileNet використовується для класифікації сцени;
3) FaceMesh аналізує положення обличчя;
4) PoseNet визначає положення тіла людини.

Такий підхід дозволяє незалежно модернізувати окремі компоненти програмно-апаратної системи без зміни загальної структури програмного забезпечення.

Крім того, модульність забезпечує можливість подальшого масштабування. У майбутньому до архітектури можуть бути інтегровані нові моделі машинного навчання, додаткові механізми аналізу поведінки або засоби автоматичного реагування на інциденти.

Особливістю розробленої архітектури є також поєднання декількох моделей комп’ютерного зору в межах єдиного програмного середовища. Це дозволяє здійснювати комплексний аналіз сцени та підвищує інформативність програмно-апаратної системи у порівнянні з рішеннями, які використовують лише одну модель детекції.

Таким чином, обрана архітектура забезпечує ефективну роботу програмно-апаратної системи аналізу відеопотоку в режимі реального часу, поєднуючи швидкодію, автономність, масштабованість та підвищений рівень безпеки даних.

Висновки до розділу 3

У третьому розділі було розглянуто процес реалізації програмної частини програмно-апаратної системи аналізу відеопотоку, а також досліджено її функціональні можливості та ефективність роботи в реальних умовах.

У ході виконання роботи було реалізовано програмно-апаратну систему, яка забезпечує обробку відеопотоку у режимі реального часу з використанням сучасних методів комп’ютерного зору та машинного навчання. Програмно-апаратна система інтегрує декілька моделей, що дозволяє виконувати комплексний аналіз сцени, включаючи детекцію об’єктів, класифікацію, аналіз обличчя та визначення пози людини.

Особливу увагу було приділено розробці механізму визначення потенційно небезпечних ситуацій, який базується на аналізі комбінацій об’єктів та поведінкових характеристик людини. Реалізований підхід дозволяє своєчасно виявляти підозрілі події та забезпечувати відповідну реакцію, зокрема візуальне оповіщення, звукову сигналізацію та автоматичне збереження знімків.

Проведене тестування підтвердило здатність програмно-апаратної системи працювати у режимі реального часу із високою швидкодією. Отримані результати показали, що програмно-апаратна система забезпечує стабільну обробку відеопотоку із частотою близько 50–55 кадрів за секунду, що відповідає вимогам до сучасних систем відеоспостереження.

Аналіз ефективності обраної архітектури показав, що використання локальної обробки даних дозволяє суттєво зменшити затримки та підвищити швидкість реакції програмно-апаратної системи у порівнянні з хмарними рішеннями. Крім того, такий підхід забезпечує підвищений рівень конфіденційності та автономності роботи.

Разом з тим встановлено, що використання попередньо навчених моделей загального призначення може впливати на точність розпізнавання окремих об’єктів, особливо у складних умовах. Незважаючи на це, система демонструє достатній рівень ефективності для практичного застосування у задачах відеоспостереження.

Важливою особливістю розробленої програмно-апаратної системи є її гнучкість та можливість адаптації до різних умов використання. Завдяки модульній структурі програмного забезпечення існує можливість розширення функціональності шляхом підключення нових моделей машинного навчання,

удосконалення алгоритмів аналізу або інтеграції додаткових джерел даних. Це дозволяє розглядати програмно-апаратну систему як основу для подальших досліджень у сфері інтелектуального відеоспостереження та автоматизованого аналізу поведінки.

Крім того, запропонований підхід відкриває перспективи практичного застосування у різних галузях, зокрема у системах безпеки, контролю доступу, моніторингу приміщень та навчальних середовищах. Використання доступних інструментів та веб-технологій робить програмно-апаратну систему економічно доцільною та простою у впровадженні, що є важливим фактором при створенні сучасних програмно-апаратних комплексів.

Таким чином, у розділі підтверджено доцільність обраних підходів до реалізації програмно-апаратної системи системи, продемонстровано її працездатність та ефективність, що створює основу для подальшого вдосконалення та розширення функціональних можливостей.

ВИСНОВКИ

У кваліфікаційній роботі було вирішено актуальну задачу розробки програмно-апаратної системи локального аналізу відеопотоку в режимі реального часу з використанням методів комп’ютерного зору та машинного навчання. У ході виконання роботи проведено аналіз сучасних підходів до обробки відеоданих, досліджено існуючі рішення у сфері відеоспостереження та обґрунтовано доцільність використання локальної архітектури.

У результаті виконання роботи було спроєктовано архітектуру програмно-апаратної системи, яка базується на використанні веб-технологій та забезпечує обробку відеоданих безпосередньо на пристрої користувача. Реалізована система інтегрує декілька моделей машинного навчання, що дозволяє виконувати комплексний аналіз сцени, включаючи детекцію об’єктів, класифікацію, аналіз обличчя та визначення пози людини.

Особливу увагу було приділено розробці механізму визначення потенційно небезпечних ситуацій, який дозволяє виявляти підозрілі події на основі аналізу об’єктів та поведінкових характеристик. Реалізований підхід забезпечує своєчасне реагування програмно-апаратної системи шляхом візуального та звукового сповіщення, а також автоматичного збереження інформації про інциденти.

Проведене тестування підтвердило ефективність розробленого рішення. Програмно-апаратна система демонструє стабільну роботу у режимі реального часу із високою швидкодією (близько 50–55 кадрів за секунду) та низькими затримками. Порівняльний аналіз показав, що запропонована локальна архітектура дозволяє значно перевищити хмарні рішення за швидкістю реагування (у 5–10 разів) та зменшити мережеве навантаження більш ніж на 90 %.

Разом з тим встановлено, що використання попередньо навчених моделей загального призначення може впливати на точність розпізнавання окремих об’єктів у складних умовах. Це свідчить про доцільність подальшого

вдосконалення програмно-апаратної системи шляхом використання спеціалізованих або донавчених моделей.

Важливим результатом роботи є також підтвердження можливості ефективного використання веб-технологій для реалізації задач комп’ютерного зору, які традиційно виконувались у середовищах із високими обчислювальними ресурсами. Практична реалізація програмно-апаратної системи демонструє, що сучасні браузерні технології у поєднанні з апаратним прискоренням дозволяють створювати продуктивні та функціонально насичені рішення без необхідності використання складної інфраструктури. Це відкриває нові можливості для розробки доступних інтелектуальних систем, орієнтованих на широкий спектр користувачів.

У цілому, поставлена мета роботи досягнута, а всі визначені завдання виконані. Розроблена програмно-апаратна система може бути використана у практичних задачах відеоспостереження, забезпечення безпеки та автоматизованого аналізу поведінки людини. Отримані результати створюють основу для подальших досліджень та розвитку інтелектуальних систем комп’ютерного зору.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 775 p.

2) Smilkov D. та ін. TensorFlow.js: Machine Learning for the Web and Beyond // arXiv. 2019. URL: https://arxiv.org/abs/1901.05350 (дата звернення: 15.10.2025).

3) Lin T.-Y. та ін. Microsoft COCO: Common Objects in Context // Proceedings of the European Conference on Computer Vision (ECCV). 2014. URL: https://arxiv.org/abs/1405.0312 (дата звернення: 10.10.2025).

4) Howard A. та ін. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications // arXiv. 2017. URL: https://arxiv.org/abs/1704.04861 (дата звернення: 01.10.2025).

5) Bazarevsky V. та ін. MediaPipe FaceMesh: Real-time Face Landmark Detection // arXiv. 2019. URL: https://arxiv.org/abs/1906.08172 (дата звернення: 18.10.2025).

6) Google. PoseNet // TensorFlow Models. URL: https://github.com/tensorflow/tfjs-models/tree/master/posenet (дата звернення: 21.10.2025).

7) Szeliski R. Computer Vision: Algorithms and Applications. London : Springer, 2010. 812 p.

8) Redmon J. та ін. You Only Look Once: Unified, Real-Time Object Detection // arXiv. 2016. URL: https://arxiv.org/abs/1506.02640 (дата звернення: 01.11.2025).

9) OpenCV. Open Source Computer Vision Library. URL: https://opencv.org (дата звернення: 14.01.2026).

10) Клочко Т. Р. // Методи оптимізації процесів і систем. Київ : КПІ ім. Ігоря Сікорського, 2024. 194 с. URL: https://ela.kpi.ua/items/27f90505-4601-4960-bb26-ddcd60b15666 (дата звернення: 14.02.2026).

11) Кобилін О. А., Творошенко І. С. Методи цифрової обробки зображень: навчальний посібник. Харків : ХНУРЕ, 2021. URL: https://openarchive.nure.ua/server/api/core/bitstreams/c739b2e6-aa8c-4fa0-92b1-dfb0d76e88d2/content (дата звернення: 10.03.2026).

12) Лавер В. О. Моделювання та обробка відеозображень: навчально-методичний посібник. Ужгород: Ужгородський національний університет, 2021 URL: https://dspace.uzhnu.edu.ua/server/api/core/bitstreams/9024f4b9-54c2-4715-8a4a-dcb9b1c0f0b3/content (дата звернення: 12.04.2026).

ДОДАТОК А

Програмний код

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-
scale=1" />
    <title>AI Vision — повний робочий файл</title>

    <!-- TensorFlow.js (обов'язково ПЕРЕД моделями) -->
    <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@3.21.0/dist/tf.
min.js"></script>

    <!-- Моделі -->
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/mobilenet@2.1.0"></script>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/coco-ssd@2.2.2"></script>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/posenet@2.2.1"></script>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow-
models/facemesh@0.0.4"></script>

    <style>
      :root {
        --bg: linear-gradient(180deg, #041021, #06122a 60%);
        --card: rgba(255, 255, 255, 0.03);
        --accent: #60a5fa;
        --accent2: #6ee7b7;
        --glass-border: rgba(255, 255, 255, 0.06);
      }
      html,
      body {
        height: 100%;
        margin: 0;
        font-family: Inter, Arial, sans-serif;
        background: var(--bg);
        color: #e6eef8;
      }
      .wrap {
        max-width: 1100px;
        margin: 24px auto;
        padding: 20px;
        border-radius: 14px;
        background: linear-gradient(
          180deg,
          rgba(255, 255, 255, 0.02),
          rgba(255, 255, 255, 0.01)
```

```
        );
        box-shadow: 0 8px 30px rgba(2, 6, 23, 0.6);
      }
      h1 {
        margin: 0 0 8px;
        font-size: 20px;
      }
      .sub {
        color: #9fb7ff;
        font-size: 13px;
        margin-bottom: 12px;
      }
      .controls {
        display: flex;
        flex-wrap: wrap;
        gap: 10px;
        align-items: center;
        margin-bottom: 12px;
      }
      .btn {
        background: linear-gradient(90deg, var(--accent2), var(--
accent));
        border: none;
        padding: 10px 14px;
        border-radius: 10px;
        color: #04203a;
        font-weight: 600;
        cursor: pointer;
        box-shadow: 0 8px 20px rgba(96, 165, 250, 0.12);
      }
      .btn:active {
        transform: translateY(1px);
      }
      .file {
        position: relative;
        overflow: hidden;
        border-radius: 10px;
      }
      .file input[type='file'] {
        position: absolute;
        left: 0;
        top: 0;
        opacity: 0;
        height: 100%;
        width: 100%;
        cursor: pointer;
      }
      .layout {
        display: grid;
        grid-template-columns: 1fr 360px;
        gap: 18px;
      }
```

```
.card {
  background: var(--card);
  border: 1px solid var(--glass-border);
  padding: 12px;
  border-radius: 12px;
}
video,
canvas,
img {
  width: 100%;
  border-radius: 8px;
  display: block;
  background: #000;
}
#canvas {
  background: #000;
}
.overlay {
  position: absolute;
  left: 12px;
  top: 12px;
  background: rgba(0, 0, 0, 0.45);
  padding: 6px 8px;
  border-radius: 8px;
  font-size: 13px;
}
.status {
  display: flex;
  gap: 8px;
  align-items: center;
  margin-left: auto;
}
.badge {
  padding: 6px 8px;
  border-radius: 999px;
  background: rgba(255, 255, 255, 0.02);
  border: 1px solid rgba(255, 255, 255, 0.03);
  font-size: 13px;
}
pre {
  white-space: pre-wrap;
  background: rgba(255, 255, 255, 0.02);
  padding: 10px;
  border-radius: 8px;
  color: #dff7e6;
}
footer {
  margin-top: 14px;
  color: rgba(230, 238, 248, 0.6);
  font-size: 13px;
  text-align: center;
}
```

```
      .danger-frame {
        outline: 6px solid red;
        box-shadow: 0 0 40px red;
      }
      @media (max-width: 920px) {
        .layout {
          grid-template-columns: 1fr;
        }
      }
    </style>
  </head>
  <body>
    <div class="wrap">
      <header style="display: flex; align-items: center; gap:
12px">
        <div>
          <h1>AI Vision — CocoSSD · MobileNet · PoseNet ·
FaceMesh</h1>
          <div class="sub">
            Надійна версія з перевірками типів, підтримкою фото з
ПК та камерою
          </div>
        </div>
        <div class="status" style="margin-left: auto">
          <div class="badge" id="loader">Завантаження
моделей...</div>
        </div>
      </header>

      <div class="controls">
        <div class="file">
          <button class="btn">🗀 Завантажити фото з ПК</button>
          <input type="file" id="upload" accept="image/*" />
        </div>

        <button class="btn" id="captureBtn">📸 Зробити
фото</button>
        <button class="btn" id="realtimeToggle">▶ Авто (реальний
час)</button>
        <button class="btn" id="loadTestBtn">
          🔎 Завантажити тестове зображення
        </button>

        <div
          style="margin-left: auto"
          class="note"
          id="note"
          style="color: #9fb7ff"
        >
          Тестовий шлях:
```

```
          <code>/mnt/data/6f5d2233-a31f-402e-a2d6-
82558d59b110.png</code>
        </div>
      </div>

      <div class="layout">
        <div class="card" style="position: relative">
          <div class="overlay" id="overlayInfo">Камера:
очікування...</div>
          <video
            id="camera"
            autoplay
            playsinline
            muted
            style="max-height: 360px"
          ></video>
          <canvas
            id="canvas"
            width="640"
            height="480"
            style="margin-top: 12px"
          ></canvas>
        </div>

        <div class="card">
          <div
            style="
              display: flex;
              align-items: center;
              gap: 8px;
              margin-bottom: 8px;
            "
          >
            <div class="badge">Результати</div>
            <div id="fps" style="margin-left: auto; color:
#bcd6ff">FPS: —</div>
          </div>
          <pre id="result">Статус: очікування...</pre>
          <div style="height: 10px"></div>
          <div
            id="thumbs"
            style="display: flex; gap: 8px; flex-wrap: wrap"
          ></div>
        </div>
      </div>

      <footer>
        Порада: якщо браузер пригальмовує — вимкни «Авто» або
окремі моделі.
      </footer>
    </div>
```

```
    <script>
      document.addEventListener('DOMContentLoaded', async () => {
        await tf.setBackend('webgl');
        await tf.ready();

        const camera = document.getElementById('camera');
        const canvas = document.getElementById('canvas');
        const ctx = canvas.getContext('2d');
        const result = document.getElementById('result');
        const fpsEl = document.getElementById('fps');
        const loader = document.getElementById('loader');

        let cocoModel, mobilenetModel, faceModel, poseModel;
        let modelsLoaded = false;

        let realtime = false;
        let rafId = null;
        let frameSkip = 0;

        let lastClasses = [];
        let eventLog = [];

        let lastDangerShotTime = 0;
        let screenshots = [];
        let dangerCounter = {};
        let threatLevel = 0; // 0 = норм, 1 = підозра, 2 =
небезпека
        let sirenPlaying = false;
        let audioCtx;

        let lastRes = {
          cocoPreds: [],
          mobilePreds: [],
          faces: [],
          pose: [],
        };

        let lastSpokenText = '';
        let lastSpeakTime = 0;

        let lastPose = null;

        // ===== SPEECH =====
        function speakSmart(text) {
          const now = Date.now();
          if (text === lastSpokenText) return;
          if (now - lastSpeakTime < 3000) return;

          const u = new SpeechSynthesisUtterance(text);
          u.lang = 'uk-UA';

          speechSynthesis.cancel();
```

```
    speechSynthesis.speak(u);

    lastSpokenText = text;
    lastSpeakTime = now;
  }

  // ===== CAMERA =====
  navigator.mediaDevices
    .getUserMedia({ video: true })
    .then((stream) => (camera.srcObject = stream));

  function setCanvasSize() {
    canvas.width = camera.videoWidth || 640;
    canvas.height = camera.videoHeight || 480;
  }

  // ===== LOAD MODELS =====
  async function loadModels() {
    loader.innerText = 'Loading...';
    cocoModel = await cocoSsd.load();
    mobilenetModel = await mobilenet.load();
    faceModel = await facemesh.load();
    poseModel = await posenet.load();
    loader.innerText = 'Готово ☑';
    modelsLoaded = true;
  }
  loadModels();

  // ===== SMART SCENE =====
  function smartScene(classes) {
    const set = new Set(classes);

    if (!classes.length) return '😶 Нічого не видно';
    if (set.has('person') && set.has('laptop')) return '💻
Людина працює';
    if (set.has('person') && set.has('cell phone')) return
'📱 Телефон';
    if (set.has('person') && set.has('cup')) return '☕
Людина п’є';
    if (set.has('cat')) return '🐱 Кіт';
    if (set.has('dog')) return '🐶 Собака';
    if (set.has('knife') && set.has('person'))
      return '🚨 Небезпека! Ніж!';
    if (set.has('fire')) return '🔥 Вогонь!';

    if (set.has('person')) return '🧍 Людина';

    return '🤖 ' + classes.join(', ');
  }
```

```
        // ===== FACE =====
        function analyzeFace(face) {
          if (!face) return null;

          const dx = face.scaledMesh[263][0] -
face.scaledMesh[33][0];

          if (dx > 60) return '👀 Дивиться вперед';
          if (dx < 40) return '↔ Вбік';

          return '😳 Обличчя';
        }

        function takeScreenshot(reason) {
          const now = Date.now();

          // ❄ антиспам (5 сек)
          if (now - lastDangerShotTime < 5000) return;

          lastDangerShotTime = now;

          const img = canvas.toDataURL('image/png');

          const shot = {
            time: new Date().toLocaleTimeString(),
            reason,
            image: img,
          };

          screenshots.push(shot);

          // обмеження пам’яті
          if (screenshots.length > 10) screenshots.shift();

          // скачування (опціонально)
          const a = document.createElement('a');
          a.href = img;
          a.download = `ALERT_${Date.now()}.png`;
          a.click();
        }

        function detectWeaponHeuristic(preds) {
          for (const p of preds) {
            const c = p.class;

            if (
              c === 'knife' ||
              c === 'scissors' ||
              c === 'fork' ||
              c === 'spoon'
            ) {
```

```
        return '⚠ Можлива зброя або металевий предмет';
      }
    }

    return null;
  }

  function updateThreatLevel(reason) {
    if (!reason) return;

    dangerCounter[reason] = (dangerCounter[reason] || 0) +
1;

    // 🔥 якщо повторюється
    if (dangerCounter[reason] >= 2) {
      threatLevel = 2;
    } else {
      threatLevel = 1;
    }
  }

  function resetThreatDecay() {
    // поступове затухання загрози
    setInterval(() => {
      if (threatLevel > 0) threatLevel--;
    }, 5000);
  }
  resetThreatDecay();

  // ===== SUSPICIOUS =====
  function detectSuspicious(classes) {
    const set = new Set(classes);

    if (set.has('knife') && set.has('person')) {
      return '🚨 УВАГА: людина з ножем!';
    }

    if (set.has('fire')) {
      return '🔥 УВАГА: вогонь!';
    }

    return null;
  }

  // ===== EVENT LOG =====
  function addEvent(text) {
    const t = new Date().toLocaleTimeString();
    eventLog.push(`[${t}] ${text}`);
    if (eventLog.length > 10) eventLog.shift();
  }
```

```
        function startSiren() {
          if (sirenPlaying) return;

          sirenPlaying = true;

          audioCtx = new (window.AudioContext ||
window.webkitAudioContext)();

          const osc = audioCtx.createOscillator();
          const gain = audioCtx.createGain();

          osc.type = 'sawtooth';
          osc.frequency.value = 600;

          gain.gain.value = 0.05;

          osc.connect(gain);
          gain.connect(audioCtx.destination);

          osc.start();

          // 🚨 хвилі сирени
          let up = true;
          setInterval(() => {
            osc.frequency.value = up ? 900 : 500;
            up = !up;
          }, 300);
        }

        function stopSiren() {
          if (!audioCtx) return;
          audioCtx.close();
          sirenPlaying = false;
        }

        // ===== DRAW =====
        function drawAR(preds) {
          ctx.lineWidth = 2;
          ctx.font = '14px Arial';

          preds.forEach((p) => {
            ctx.strokeStyle = 'lime';
            ctx.strokeRect(...p.bbox);
            ctx.fillStyle = 'lime';
            ctx.fillText(p.class, p.bbox[0], p.bbox[1] - 5);
          });
        }

        function updateUIWarning() {
          if (threatLevel === 2) {
            canvas.classList.add('danger-frame');
          } else {
```

```
            canvas.classList.remove('danger-frame');
          }
        }

        function detectFall(pose) {
          if (!pose) return null;

          const nose = pose.keypoints.find((k) => k.part ===
'nose');
          const ankles = pose.keypoints.filter((k) =>
k.part.includes('ankle'));

          if (!nose || ankles.length < 2) return null;

          if (nose.position.y > ankles[0].position.y) {
            return '🆘 Можливе падіння людини!';
          }

          return null;
        }

        function detectMovement(pose) {
          if (!pose || !lastPose) {
            lastPose = pose;
            return null;
          }

          const dx = Math.abs(
            pose.keypoints[0].position.x -
lastPose.keypoints[0].position.x
          );

          lastPose = pose;

          if (dx > 50) {
            return '🏃 Різкий рух';
          }

          return null;
        }

        function detectHandsUp(pose) {
          if (!pose) return null;

          const leftWrist = pose.keypoints.find((k) => k.part ===
'leftWrist');
          const rightWrist = pose.keypoints.find(
            (k) => k.part === 'rightWrist'
          );
          const nose = pose.keypoints.find((k) => k.part ===
'nose');
```

```
        if (!leftWrist || !rightWrist || !nose) return null;

        if (
          leftWrist.position.y < nose.position.y &&
          rightWrist.position.y < nose.position.y
        ) {
          return '🙌 Руки підняті';
        }

        return null;
      }

      // ===== ANALYZE =====
      async function analyzeFrame() {
        if (!modelsLoaded) return;

        try {
          lastRes.cocoPreds = await cocoModel.detect(canvas);

          if (frameSkip % 5 === 0) {
            lastRes.mobilePreds = await
mobilenetModel.classify(canvas);
          }

          if (frameSkip % 6 === 0) {
            lastRes.pose = await
poseModel.estimateSinglePose(canvas);
          }

          if (frameSkip % 7 === 0) {
            lastRes.faces = await
faceModel.estimateFaces(canvas);
          }

          frameSkip++;

          const classes = lastRes.cocoPreds.map((p) => p.class);
          let output = [];

          const scene = smartScene(classes);
          output.push(scene);

          const weapon =
detectWeaponHeuristic(lastRes.cocoPreds);
          if (weapon) {
            output.push(weapon);
            speakSmart(weapon);

            // 🔥 теж скрін
            takeScreenshot(weapon);
          }
```

```
    const danger = detectSuspicious(classes);
    if (danger) {
      output.push(danger);
      speakSmart(danger);

      updateThreatLevel(danger);
      takeScreenshot(danger);
    }

    addEvent(scene);

    if (lastRes.faces.length) {
      const f = analyzeFace(lastRes.faces[0]);
      if (f) output.push(f);
    }

    if (lastRes.mobilePreds.length) {
      const p = lastRes.mobilePreds[0];
      output.push(`🧠 ${p.className}`);
    }

    if (lastRes.pose) {
      const fall = detectFall(lastRes.pose);
      const move = detectMovement(lastRes.pose);
      const hands = detectHandsUp(lastRes.pose);

      [fall, move, hands].forEach((msg) => {
        if (msg) {
          output.push(msg);
          speakSmart(msg);
        }
      });
    }

    output.push('\n📜 Історія:');
    output.push(eventLog.join('\n'));

    result.innerText = output.join('\n');

    speakSmart(scene);

    drawAR(lastRes.cocoPreds);
  } catch (e) {
    console.error(e);
  }
}

// ===== LOOP =====
async function loop() {
  if (!realtime) return;

  setCanvasSize();
```

```
          ctx.drawImage(camera, 0, 0);

          const start = performance.now();

          await analyzeFrame();

          fpsEl.innerText =
            'FPS: ' + Math.round(1000 / (performance.now() -
start));

          setTimeout(() => {
            rafId = requestAnimationFrame(loop);
          }, 50);

          if (threatLevel >= 2) {
            startSiren();
          } else {
            stopSiren();
          }
          updateUIWarning();
        }

        // ===== BUTTON =====
        realtimeToggle.onclick = () => {
          realtime = !realtime;
          realtimeToggle.innerText = realtime ? '■ Стоп' : '▶
Старт';

          if (realtime) loop();
          else cancelAnimationFrame(rafId);
        };
      });
    </script>
  </body>
</html>
```