

ДОДАТОК А

Програмний модуль кінематично-адаптивного просторово-часового проріджування відеоряду за методом KASFraD

```
import numpy as np
import cv2

def scale_factor(prev_gray: np.ndarray, curr_gray: np.ndarray)
-> float:
    """
    Розрахунок коефіцієнта масштабування в афінній моделі руху
    для оцінювання динаміки наближення камери до об'єкта.
    """
    pts = cv2.goodFeaturesToTrack(prev_gray, maxCorners=250,
    qualityLevel=0.01, minDistance=10, blockSize=7)
    if pts is None or len(pts) < 8:
        return 1.0
    next_pts, status, _ = cv2.calcOpticalFlowPyrLK(prev_gray,
    curr_gray, pts, None)
    if next_pts is None or status is None:
        return 1.0
    ok_prev = pts[status.ravel() == 1]
    ok_next = next_pts[status.ravel() == 1]
    if len(ok_prev) < 6:
        return 1.0
    matrix, _ = cv2.estimateAffinePartial2D(ok_prev, ok_next,
    method=cv2.RANSAC, ransacReprojThreshold=3.0)
    if matrix is None:
        return 1.0
    return float((np.hypot(matrix[0, 0], matrix[0, 1]) +
    np.hypot(matrix[1, 0], matrix[1, 1])) * 0.5)
```

ДОДАТОК Б

Програмний модуль оркестрації хмарних репозиторіїв та автоматизованого імпорту завдань у CVAT REST API

```
import json
import re
import subprocess

def create_cvat_task(task_name: str, local_zip_path: str,
labels: list) -> str:
    """
    Автоматизоване створення нового завдання у CVAT та імпорт
    підготовленого архіву з кадрами і псевдоанотаціями через
    CLI.
    """
    labels_str = json.dumps(labels).replace("'", '\\\\"')
    create_cmd = (
        f'cvat-cli --server-host {CVAT_HOST} --server-port
{CVAT_PORT} '
        f'--auth {CVAT_USER}:{CVAT_PASS} create "{task_name}"
',
        f'--labels "{labels_str}" local "{local_zip_path}"'
    )

    # Виконання команди операційної системи для взаємодії з
    CVAT CLI
    try:
        result = subprocess.run(create_cmd, shell=True,
check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
text=True)
        output = result.stdout
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"Помилка виконання команди CVAT
CLI: {e.stderr}")

    match = re.search(r'(?i)ID[\\s:]*(\\d+)', output)
    if not match:
        raise RuntimeError(f"Не вдалося отримати ID завдання
для {task_name}. Вихідні дані: {output}")

    return match.group(1)
```

ДОДАТОК В

Програмний модуль евристичної дедуплікації зображень та оцінки розмиття

```

from pathlib import Path
from typing import Tuple
import numpy as np
from PIL import Image

def compute_metrics_worker(img_path: Path) -> Tuple[Path, int,
float]:
    """
    Обчислення різницевого хешу dHash та дисперсії Лапласіана
    для оцінки розмиття окремого зображення.
    """
    try:
        # 1. Обчислення різницевого хешу dHash (розмірність
9x8)
        with Image.open(img_path) as img:
            img_gray = img.convert('L').resize((9, 8),
Image.Resampling.LANCZOS)
            arr = np.array(img_gray, dtype=np.uint8)

            # Порівняння суміжних пікселів по рядках
            diff = arr[:, :-1] > arr[:, 1:]
            decimal_value = 0
            for bit in diff.ravel():
                decimal_value = (decimal_value << 1) | int(bit)

            # 2. Оцінка ступеня розмиття через фільтр Лапласа
            with Image.open(img_path) as img:
                img_gray = img.convert('L')
                arr = np.array(img_gray, dtype=np.float32)

            # Наближене обчислення згортки з ядром Лапласа
            laplacian = (arr[1:-1, 2:] + arr[1:-1, :-2] + arr[2:,
1:-1] + arr[:-2, 1:-1] - 4 * arr[1:-1, 1:-1])
            variance = float(laplacian.var())

            return img_path, decimal_value, variance
    except Exception:
        return img_path, 0, 1.0

```

ДОДАТОК Г

Програмна конфігурація навчання та валідації без'якорного детектора

YOLO26-nano

```

from ultralytics import YOLO

def run_training_pipeline(yaml_path_baseline: str):
    """
    Ініціалізація та запуск оптимізованого процесу навчання
    детектора YOLO26-nano.
    Конфігурація оптимізована для детекції дрібних об'єктів з
    БпЛА без NMS.
    """
    # Завантаження базової конфігурації моделі YOLO26-nano
    model_4 = YOLO("yolov26n.yaml")

    # Запуск процесу навчання з оптимізованими
    гіперпараметрами
    results_4 = model_4.train(
        data=yaml_path_baseline,
        epochs=100,
        imgsz=640,
        batch=-1,
        patience=25,
        device=0,
        project='/kaggle/working/runs/detect',
        name='yolov26_musgd_e2e_experiment4',
        save=True,
        end2end=True,          # Активація режиму детекції без
NMS (One-to-One head)
        optimizer="MuSGD",    # Спеціалізований оптимізатор для
дуальних градієнтів
        hsv_s=0.3, hsv_v=0.3, # Стійкість до тіней та змін
освітленості
        fliplr=0.5, flipud=0.5, # Компенсація просторової
упередженості ракурсу БпЛА
        translate=0.1, degrees=15.0, # Стабілізація по кутах
нахилу камери
        scale=0.0, shear=0.0, perspective=0.0 # Захист
геометричної цілісності дрібних цілей
    )
    return results_4

```

ДОДАТОК Д

Програмний модуль трекінгу цілей та обчислення просторового відхилення

```

import queue
import numpy as np

def process_tracking_stream(self):
    """
    Паралельне обчислення вектора відхилення геометричного
    центру цілі
    від фокального центру зображення в потоці обробки відео.
    """
    while not self.stopped or not self.capture_queue.empty():
        try:
            frame_id, frame =
self.capture_queue.get(timeout=1.0)
            except queue.Empty:
                continue

            results = self.model(frame, conf=0.01, imgsz=(360,
640), rect=True, verbose=False)[0]
            boxes = results.boxes
            tracker_input = []
            if boxes is not None:
                xyxy = boxes.xyxy.cpu().numpy()
                confs = boxes.conf.cpu().numpy()
                for i in range(len(xyxy)):
                    tracker_input.append([xyxy[i][0], xyxy[i][1],
xyxy[i][2], xyxy[i][3], confs[i], 0])

            tracker_input = np.array(tracker_input) if
len(tracker_input) > 0 else np.empty((0, 6))
            online_targets = self.tracker.update(tracker_input,
frame)

            frame_results = []
            for target in online_targets:
                x1, y1, x2, y2, track_id = target[:5]
                cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)
                dx = cx - self.screen_cx # Помилка утримання по
горизонталі (відхилення)
                dy = cy - self.screen_cy # Помилка утримання по
вертикалі (відхилення)
                frame_results.append({"track_id": int(track_id),
"vector": [dx, dy]})

            self.output_queue.put((frame_id, frame_results))

```

ДОДАТОК Е

Програмний модуль низькорівневої оптимізації та PTQ-компіляції моделі

```

from ultralytics import YOLO

def export_optimized_ptq_model(resolved_weights_path: str,
                               yaml_path_baseline: str):
    """
    Виконання низькорівневої оптимізації моделі за технологією
    PTQ.
    Експорт у формат TensorRT INT8 із FP16 Fallback та
    стабільний ONNX FP16.
    """
    ptq_model = YOLO(resolved_weights_path)

    # 1. Експорт у формат TensorRT з калібруванням INT8 та
    # автоматичним FP16 Fallback
    ptq_engine_path = ptq_model.export(
        format="engine",
        int8=True,
        half=True, # Дозволяє автоматичний відкат (Fallback)
        data=yaml_path_baseline,
        device=0,
        opset=12
    )
    до FP16 для нестабільних шарів

    # 2. Експорт у стабільний та апаратно-незалежний формат
    # ONNX FP16
    ptq_onnx_path = ptq_model.export(
        format="onnx",
        half=True, # Переклад усіх ваг та операцій у формат
        напівточності FP16
        opset=12
    )

    return ptq_engine_path, ptq_onnx_path

```

ДОДАТОК Ж

Конфігураційні параметри датасету

```
path: /kaggle/working/dataset_v25_640
train: train/images
val: val/images
test: test/images

names:
  0: military_object
```

ДОДАТОК И

Скрипт порівняльної оцінки точності моделей у форматах FP32 та FP16

```
# Оцінка детектора у форматі FP32
model_fp32 = YOLO(weights_path_fp32)
metrics_fp32 = model_fp32.val(
    data=yaml_path, split='test', imgsz=640, batch=16,
    device=0 if torch.cuda.is_available() else 'cpu',
    rect=True, half=False, verbose=False
)

# Оцінка детектора у форматі FP16
model_fp16 = YOLO(weights_path_fp16)
metrics_fp16 = model_fp16.val(
    data=yaml_path, split='test', imgsz=640, batch=16,
    device=0 if torch.cuda.is_available() else 'cpu',
    rect=True, half=True, verbose=False
)
```

ДОДАТОК К

Алгоритм розрахунку координат центру та вектора відхилення об'єкта

```
# x1, y1, x2, y2 – координати знайденої рамки; w, h – її ширина  
та висота  
w = x2 - x1  
h = y2 - y1  
  
# Координати геометричного центру об'єкта  
cx = int(x1 + w / 2)  
cy = int(y1 + h / 2)  
  
# Розрахунок відхилення від оптичного центру кадру (screen_cx,  
screen_cy)  
dx = cx - screen_cx  
dy = cy - screen_cy
```

ДОДАТОК Л

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		103 с.	