ДОДАТОК А

Лістинг програми автоматичного маркування метаданих VLM

```
      def classify_single_video(frames):
          prompt = """
              You are an expert in computer vision and military UAV
reconnaissance. You have been provided with a chronological sequence of
several frames from a SINGLE video recording.
              As this is a single video, the target (if present) is of the
same type in all frames.
              In some frames, the target may not yet be present or may be
invisible. Analyse ALL frames, find the one where the physical object in
the air is most clearly visible, and use this to draw your final
conclusion.

              Return the response STRICTLY in JSON format with the
following three fields (use lowercase only):

              1. "target_shape": the morphology (geometric shape) of a
military UAV.
              CRITICAL RULES REGARDING ARTIFACTS:
              - Ignore watermarks and logos! There may be logos on the
screen in the form of drone icons (e.g., yellow drawings). These are
graphics, not targets!
              - Ignore system crosshairs! The cross-shaped marker
(crosshairs) in the centre of the screen is part of the interface. It is
STRICTLY FORBIDDEN to confuse this cross with a drone of the "cross" shape.
              - The real target is a physical object in the sky.

              Select STRICTLY one option based on the silhouette SHAPE:
              - delta (delta-shaped silhouette, solid triangular wing,
arrow-shaped form without a distinct tubular fuselage. This class includes
Shahed-type kamikaze drones);
              - cross (cross-shaped silhouette, featuring characteristic X-
shaped wings at the front or rear. This class includes the Lancet and
similar models. Do not confuse with a sight!);
              - wing (classic straight wing, aeroplane-type airframe with a
pronounced long wingspan and narrow fuselage. This class includes
reconnaissance wings);
              - no_drone (if there is NO actual drone in the sky in ANY of
the frames: only clouds, land, a crosshair or watermarks);

              2. "camera_type": camera type.
              Select ONE option:
              - rgb (daytime colour camera, with natural colours and
shadows)
              - thermal (thermal imaging camera: monochrome image without
shadows; background is a uniform plane, target is a contrasting silhouette)

              3. "background": the visual complexity of the background
DIRECTLY BEHIND THE OBJECT.
              Assess the background only in the frame where the target is
most clearly visible. Assess the background area ONLY behind the target’s
silhouette.

              Select ONE option:
              - sky (the silhouette is entirely against a background of sky
or clouds)
              - uniform (the silhouette is entirely against a uniform
surface: water, sea, smooth snow)
```

```
            - field (the silhouette is entirely against a 2D geometric
background: agricultural fields, roads)
            - forest (the silhouette is entirely against an organic
texture: forest, trees)
            - urban (the silhouette is entirely against a hard geometric
background: urban buildings, infrastructure)
            - horizon (the background behind the silhouette is mixed: the
horizon line cuts through the background directly behind the object)

            Example response:
            {
                "target_shape": "cross",
                "camera_type": "rgb",
                "background": "field"
            }
        """
        try:
            response = client.models.generate_content(
                model='gemini-2.5-pro',
                contents=[prompt, *frames],
                config=types.GenerateContentConfig(
                    response_mime_type="application/json",
                ),
            )
            return json.loads(response.text)
        except Exception as e:
            tqdm.write(f"  [!] API Error: {e}")
            return None

    processed_count = 0
    skipped_count = 0
    error_count = 0

    existing_files = os.listdir(DESTINATION_DIR)

    print(f"Starting to scan the directory: {SOURCE_ROOT}...")

    all_videos = []
    for root, dirs, files in os.walk(SOURCE_ROOT):
        for file in files:
            if file.lower().endswith(('.mp4', '.mov', '.m4v')):
                all_videos.append((root, file))

    print(f"Videos found for processing: {len(all_videos)}")
    print(f"Previously processed files found: {len(existing_files)}\n" +
"-"*40)

    for root, file in tqdm(all_videos, desc="Video classification",
unit="file"):
        current_folder_name = os.path.basename(root)

        try:
            video_path = os.path.join(root, file)
            file_name_clean = os.path.splitext(file)[0]
            file_ext = os.path.splitext(file)[1].lower()

            expected_suffix =
f"_{current_folder_name}_{file_name_clean}{file_ext}"
            already_processed = any(f.endswith(expected_suffix) for f in
existing_files)

            if already_processed:
                skipped_count += 1
                continue
```

```
                frames = extract_keyframes_from_video(video_path)
                if not frames:
                    tqdm.write(f" [!] Unable to extract frames: {file}")
                    error_count += 1
                    continue

                if SAVE_DEBUG_FRAMES:
                    debug_base_dir = "debug_frames"
                    video_debug_dir = os.path.join(debug_base_dir,
file_name_clean)
                    os.makedirs(video_debug_dir, exist_ok=True)
                    for i, pil_img in enumerate(frames):
                        pil_img.save(os.path.join(video_debug_dir,
f"frame_{i:03d}.png"))

                classification = classify_single_video(frames)
                if not classification:
                    tqdm.write(f" [!] API response empty: {file}")
                    error_count += 1
                    continue

                target = classification.get("target_shape", "unknown")
                camera = classification.get("camera_type", "unknown")
                bg = classification.get("background", "unknown")

                new_file_name =
f"{target}_{camera}_{bg}_{current_folder_name}_{file_name_clean}{file_ext}"
                new_file_path = os.path.join(DESTINATION_DIR, new_file_name)

                shutil.copy2(video_path, new_file_path)

                existing_files.append(new_file_name)
                processed_count += 1

                time.sleep(4.0)

            except Exception as e:
                tqdm.write(f" [CRITICAL ERROR] Error with file {file}: {e}")
                error_count += 1
                continue
```

ДОДАТОК Б

Програмна реалізація алгоритму напівавтоматичної розмітки даних

```
import os
import shutil
import torch
from transformers import Sam3VideoModel, Sam3VideoProcessor
from transformers.video_utils import load_video


def create_cvat_zip(outputs_per_frame, video_filename, output_dir):
    base_dir = os.path.join(output_dir, "cvat_mot_tracking")
    gt_dir = os.path.join(base_dir, "gt")
    os.makedirs(gt_dir, exist_ok=True)

    with open(os.path.join(base_dir, "labels.txt"), "w") as f:
        f.write("drone\n")

    with open(os.path.join(gt_dir, "gt.txt"), "w") as f:
        for frame_idx, data in outputs_per_frame.items():
            mot_frame_idx = frame_idx + 1
            if "boxes" in data and len(data["boxes"]) > 0:
                for i, box in enumerate(data["boxes"]):
                    track_id = i + 1
                    x1, y1, x2, y2 = box.tolist()
                    width = x2 - x1
                    height = y2 - y1

f.write(f"{mot_frame_idx},{track_id},{x1:.2f},{y1:.2f},{width:.2f},{height:
.2f},1,1,1\n")

    zip_path = os.path.join(output_dir,
os.path.splitext(os.path.basename(video_filename))[0] + "_mot")
    return shutil.make_archive(zip_path, 'zip', base_dir)


def generate_pseudo_annotations(video_path, prompt_text, output_dir,
device="cuda"):
    model = Sam3VideoModel.from_pretrained("facebook/sam3",
torch_dtype=torch.float16).to(device)
    processor = Sam3VideoProcessor.from_pretrained("facebook/sam3")
```

```
        video_frames = load_video(video_path)[0]
        inference_session = processor.init_video_session(
            video=video_frames,
            inference_device=device,
            processing_device="cpu",
            video_storage_device="cpu",
            dtype=torch.float16
        )
        processor.add_text_prompt(inference_session=inference_session,
text=prompt_text)

        outputs_per_frame = {}
        for model_outputs in
model.propagate_in_video_iterator(inference_session=inference_session):
            processed_outputs =
processor.postprocess_outputs(inference_session, model_outputs)
            outputs_per_frame[model_outputs.frame_idx] =
processed_outputs

        zip_file = create_cvat_zip(outputs_per_frame, video_path,
output_dir)
        return zip_file
```

ДОДАТОК В

Програмна реалізація алгоритму дедуплікації за метрикою IoU

```
def calculate_iou(box1, box2):
    b1_x1, b1_y1 = box1[0] - box1[2]/2, box1[1] - box1[3]/2
    b1_x2, b1_y2 = box1[0] + box1[2]/2, box1[1] + box1[3]/2
    b2_x1, b2_y1 = box2[0] - box2[2]/2, box2[1] - box2[3]/2
    b2_x2, b2_y2 = box2[0] + box2[2]/2, box2[1] + box2[3]/2

    ix1, iy1 = max(b1_x1, b2_x1), max(b1_y1, b2_y1)
    ix2, iy2 = min(b1_x2, b2_x2), min(b1_y2, b2_y2)

    if ix2 <= ix1 or iy2 <= iy1:
        return 0.0

    iarea = (ix2 - ix1) * (iy2 - iy1)
    b1_area = box1[2] * box1[3]
    b2_area = box2[2] * box2[3]
    return iarea / (b1_area + b2_area - iarea + 1e-7)


def read_bbox(label_path):
    with open(label_path, 'r') as f:
        parts = f.readline().strip().split()
    return list(map(float, parts[1:5]))


def run_iou_deduplication(pos_dir, trash_dir, iou_threshold=0.95):
    pos_dir = Path(pos_dir)
    trash_dir = Path(trash_dir)

    all_labels = sorted(glob.glob(str(pos_dir / "*.txt")))

    video_groups = {}
    for lbl in all_labels:
        v_id = "_f".join(os.path.basename(lbl).split("_f")[:-1])
        video_groups.setdefault(v_id, []).append(lbl)

    for v_id in video_groups:
        video_groups[v_id].sort()
```

```
        trash_dir.mkdir(parents=True, exist_ok=True)

        for video_name, labels in video_groups.items():
            if not labels:
                continue
            anchor_box = read_bbox(labels[0])

            for lbl_path in labels[1:]:
                curr_box = read_bbox(lbl_path)

                if calculate_iou(anchor_box, curr_box) > iou_threshold:
                    shutil.move(lbl_path, trash_dir /
os.path.basename(lbl_path))
                    img_path = Path(lbl_path).with_suffix('.jpg')
                    if img_path.exists():
                        shutil.move(str(img_path), trash_dir /
img_path.name)
                else:
                    anchor_box = curr_box
```

ДОДАТОК Г

Програмна реалізація алгоритму семантичної дедуплікації даних

```
import os
import glob
import shutil
import re
from pathlib import Path
import torch
import numpy as np


def k_center_greedy_threshold(embeddings: np.ndarray, theta: float) -
> list[int]:
    n = len(embeddings)
    if n == 0:
        return []
    if n == 1:
        return [0]

    sim = embeddings @ embeddings.T
    sim = np.clip(sim, -1.0, 1.0)
    dists = 1.0 - sim

    selected_centers = [0]
    min_dists = dists[0].copy()

    while True:
        farthest_idx = np.argmax(min_dists)
        max_dist = min_dists[farthest_idx]

        if max_dist < theta:
            break

        selected_centers.append(farthest_idx)
        min_dists = np.minimum(min_dists, dists[farthest_idx])

    return selected_centers
```

```
      def move_files_to_trash(label_path: Path, trash_dir: Path,
is_positive: bool):
          try:
              img_path = label_path.with_suffix('.jpg')
              if is_positive and label_path.exists():
                  shutil.move(str(label_path), trash_dir / label_path.name)
              if img_path.exists():
                  shutil.move(str(img_path), trash_dir / img_path.name)
          except OSError:
              pass


      def run_kcenter_deduplication(target_data_dir: Path, emb_path: Path,
trash_dir: Path, theta: float, is_positive: bool):
          try:
              embeddings_dict = torch.load(emb_path, map_location="cpu",
weights_only=False)
          except Exception:
              return

          search_pattern = "*.txt" if is_positive else "*.jpg"
          all_files = sorted(glob.glob(str(target_data_dir /
search_pattern)))

          video_groups = {}
          for filepath_str in all_files:
              filepath = Path(filepath_str)
              v_id = re.sub(r'_f\d+$', '', filepath.stem)
              video_groups.setdefault(v_id, []).append(filepath)

          for v_id in video_groups:
              video_groups[v_id].sort()

          trash_dir.mkdir(parents=True, exist_ok=True)

          for video_name, files in video_groups.items():
              valid_files = []
              valid_embs = []

              for file_path in files:
                  img_name = file_path.with_suffix(".jpg").name
                  if img_name in embeddings_dict:
                      valid_files.append(file_path)
```

```
                valid_embs.append(embeddings_dict[img_name])

        if len(valid_files) < 2:
            continue

        valid_embs = np.stack(valid_embs)

        if not is_positive:
            valid_embs = valid_embs[:, :768]

        keep_indices = set(k_center_greedy_threshold(valid_embs,
theta))

        for idx, file_path in enumerate(valid_files):
            if idx not in keep_indices:
                move_files_to_trash(file_path, trash_dir,
is_positive)
```

ДОДАТОК Д

Програмна реалізація конвеєра навчання та тестування моделі детекції

```
from pathlib import Path
from ultralytics import YOLO

DEFAULT_MODEL_TYPE = "yolo26n.pt"
IMGSZ = 1280
EPOCHS = 50
BATCH_SIZE = 8
PATIENCE = 15


def run_yolo_training(yaml_path: Path, run_name: str, output_dir: Path)
-> Path:
    model = YOLO(DEFAULT_MODEL_TYPE)

    model.train(
        data=str(yaml_path),
        epochs=EPOCHS,
        imgsz=IMGSZ,
        batch=BATCH_SIZE,
        device=0,
        optimizer="auto",
        mosaic=0.0,
        mixup=0.0,
        copy_paste=0.0,
        scale=0.0,
        fliplr=0.0,
        degrees=0.0,
        shear=0.0,
        translate=0.0,
        hsv_h=0.0,
        hsv_s=0.0,
        hsv_v=0.0,
        rect=True,
        cache="disk",
        workers=2,
        seed=42,
        deterministic=True,
        patience=PATIENCE,
```

```
            project=str(output_dir),
            name=run_name,
            exist_ok=True,
            save=True,
            plots=True,
        )

        best_weights = Path(output_dir) / run_name / "weights" / "best.pt"
        return best_weights


    def evaluate_model(weights_path: Path, yaml_path: Path, run_name: str,
output_dir: Path):
        model = YOLO(str(weights_path))

        metrics = model.val(
            data=str(yaml_path),
            split="test",
            imgsz=IMGSZ,
            batch=BATCH_SIZE,
            device=0,
            rect=True,
            project=str(output_dir),
            name=f"{run_name}_test_evaluation",
            exist_ok=True,
            save=True,
            plots=True
        )

        return metrics
```

ДОДАТОК Е

Лістинг програми компіляції моделі YOLO26n та вимірювання швидкодії інференсу

```
import os
import time
from pathlib import Path
import numpy as np
import torch
from ultralytics import YOLO

IMGSZ = 1280

def compile_model_to_tensorrt(weights_path: Path) -> Path:
    model_fp32 = YOLO(str(weights_path)).to("cuda")

    engine_path_str = model_fp32.export(
        format="engine",
        imgsz=IMGSZ,
        half=True,
        workspace=4,
        dynamic=False
    )
    return Path(engine_path_str)


def benchmark_inference(model_path: Path, num_benchmark=100, num_warmup=10):
    model = YOLO(str(model_path), task="detect")
    dummy_img = np.random.randint(0, 255, (IMGSZ, IMGSZ, 3), dtype=np.uint8)

    for _ in range(num_warmup):
        model(dummy_img, verbose=False, device="cuda")
    torch.cuda.synchronize()

    times = []
    for _ in range(num_benchmark):
        torch.cuda.synchronize()
        t0 = time.perf_counter()
        model(dummy_img, verbose=False, device="cuda")
```

```
        torch.cuda.synchronize()
        times.append(time.perf_counter() - t0)

    times_ms = np.array(times) * 1000
    mean_ms = np.mean(times_ms)
    fps = 1000.0 / mean_ms
    return fps, mean_ms
```

ДОДАТОК Ж

Лістинг програми розрахунку карт активації EigenCAM

```
import cv2
import numpy as np
import torch
import matplotlib.pyplot as plt
from pathlib import Path
from ultralytics import YOLO

def find_sppf_layer(pytorch_model):
    for i, layer in enumerate(pytorch_model.model):
        if "SPPF" in type(layer).__name__:
            return layer
    return pytorch_model.model[9]

def manual_eigencam(pytorch_model, target_layer, img_tensor):
    activations = {}

    def hook(model, input, output):
        activations["feat"] = output.detach()

    handle = target_layer.register_forward_hook(hook)
    with torch.no_grad():
        _ = pytorch_model(img_tensor)
    handle.remove()

    feat = activations["feat"].cpu().squeeze(0).numpy()
    reshaped = feat.reshape(feat.shape[0], -1)
    _, _, Vt = np.linalg.svd(reshaped, full_matrices=False)

    cam_map = Vt[0].reshape(feat.shape[1], feat.shape[2])
    if cam_map.mean() < 0:
        cam_map = -cam_map

    cam_map = np.maximum(cam_map, 0)
    cam_map -= cam_map.min()
    if cam_map.max() > 0:
        cam_map /= cam_map.max()
    return cam_map
```

```
    def overlay_heatmap(img_rgb, cam_map, alpha=0.4):
        h, w = img_rgb.shape[:2]
        heatmap = cv2.resize(cam_map, (w, h))
        heatmap = np.uint8(255 * heatmap)
        heatmap_color = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
        heatmap_color = cv2.cvtColor(heatmap_color, cv2.COLOR_BGR2RGB)
        return cv2.addWeighted(img_rgb, 1 - alpha, heatmap_color, alpha,
0)


    def preprocess_image(img_path: str, device: torch.device):
        img = cv2.imread(img_path)
        if img is None:
            return None, None, None
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img_resized = cv2.resize(img_rgb, (640, 640))
        tensor = torch.from_numpy(img_resized).permute(2, 0, 1).float() /
255.0
        tensor = tensor.unsqueeze(0).to(device)
        return img_rgb, img_resized, tensor


    def visualize_attention(model_path: str, img_path: str, save_path: str,
device_str="cuda"):
        device = torch.device(device_str)
        model = YOLO(model_path)
        pytorch_model = model.model
        pytorch_model.eval()
        pytorch_model.to(device)

        target_layer = find_sppf_layer(pytorch_model)
        img_rgb, _, tensor = preprocess_image(img_path, device)

        cam_map = manual_eigencam(pytorch_model, target_layer, tensor)
        overlay = overlay_heatmap(img_rgb, cam_map)

        fig, axes = plt.subplots(1, 2, figsize=(12, 6))
        axes[0].imshow(img_rgb)
        axes[0].set_title("Original Image")
        axes[0].axis("off")

        axes[1].imshow(overlay)
```

```
axes[1].set_title("EigenCAM Attention")
axes[1].axis("off")

plt.tight_layout()
plt.savefig(save_path, dpi=120, bbox_inches="tight")
plt.close()
```

ДОДАТОК И

Лістинг програми порівняльного тестування та вимірювання затримок трекерів ByteTrack та OC-SORT

```
import configparser
import time
from pathlib import Path
import numpy as np
from boxmot.trackers.bytetrack.bytetrack import ByteTrack
from boxmot.trackers.ocsort.ocsort import OcSort


def get_configured_tracker(tracker_name: str, config: dict, fps:
int):
    if tracker_name.lower() == "bytetrack":
        return ByteTrack(
            track_thresh=config.get("track_thresh", 0.5),
            match_thresh=config.get("match_thresh", 0.8),
            track_buffer=int(config.get("track_buffer", fps))
        )
    elif tracker_name.lower() == "ocsort":
        return OcSort(
            det_thresh=config.get("det_thresh", 0.5),
            max_age=int(config.get("max_age", fps)),
            iou_threshold=config.get("iou_threshold", 0.3)
        )
    else:
        raise ValueError(f"Unknown tracker type: {tracker_name}")


def run_tracking_pipeline(dataset_dir: Path, out_dir: Path,
split_name: str,
                          tracker_name: str, tracker_config: dict,
                          run_dir_name: str, detections_dir: Path =
None):
    split_dir = Path(dataset_dir) / f"drone-{split_name}"
    if not split_dir.exists():
        return

    sequences = [d for d in split_dir.iterdir() if d.is_dir()]
```

```
        global_total_time = 0.0
        global_total_frames = 0

        for seq in sequences:
            seq_name = seq.name

            config = configparser.ConfigParser()
            config.read(seq / "seqinfo.ini")
            width = int(config['Sequence']['imWidth'])
            height = int(config['Sequence']['imHeight'])
            fps = int(config['Sequence']['frameRate'])
            seq_length = int(config['Sequence']['seqLength'])

            tracker = get_configured_tracker(tracker_name,
tracker_config, fps)

            if detections_dir is None:
                det_file = seq / "gt/gt.txt"
            else:
                det_file = Path(detections_dir) / f"{seq_name}.txt"

            if not det_file.exists():
                continue

            if det_file.stat().st_size > 0:
                detections = np.loadtxt(det_file, delimiter=',')
                if detections.ndim == 1:
                    detections = np.array([detections])
            else:
                detections = np.empty((0, 9))

            results = []
            dummy_frame = np.zeros((height, width, 3), dtype=np.uint8)

            seq_tracker_time = 0.0
            seq_total_frames = 0

            for frame_id in range(1, seq_length + 1):
                if len(detections) > 0:
                    frame_dets = detections[detections[:, 0] == frame_id]
                else:
                    frame_dets = np.empty((0, 9))
```

```
                if frame_dets.ndim == 1:
                    frame_dets = np.array([frame_dets])

                tracker_input = []
                if len(frame_dets) > 0:
                    for det in frame_dets:
                        left, top, w, h = det[2:6]
                        conf = det[6] if (detections_dir is not None and
det[6] > 0) else 1.0
                        cls = 1
                        tracker_input.append([left, top, left + w, top +
h, conf, cls])

                tracker_input = np.array(tracker_input) if
len(tracker_input) > 0 else np.empty((0, 6))

                start_time = time.perf_counter()
                online_targets = tracker.update(tracker_input,
dummy_frame)
                end_time = time.perf_counter()

                seq_tracker_time += (end_time - start_time)
                seq_total_frames += 1

                for target in online_targets:
                    x1, y1, x2, y2, track_id, conf, cls = target[:7]
                    results.append([frame_id, track_id, x1, y1, x2 - x1,
y2 - y1, conf, -1, -1, -1])

            global_total_time += seq_tracker_time
            global_total_frames += seq_total_frames

            out_path = Path(out_dir) / f"drone-{split_name}" /
run_dir_name / "data"
            out_path.mkdir(parents=True, exist_ok=True)

            res_file = out_path / f"{seq_name}.txt"
            if len(results) > 0:
                np.savetxt(res_file, np.array(results),
fmt='%d,%d,%.2f,%.2f,%.2f,%.2f,%.2f,%d,%d,%d')
            else:
                open(res_file, 'w').close()
```

```
        if global_total_frames > 0:
            global_latency_ms = (global_total_time / global_total_frames)
* 1000
            global_fps = global_total_frames / global_total_time
            return global_latency_ms, global_fps
        return 0.0, 0.0
```

ДОДАТОК К

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	139 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Книш А.О.			Донавчання та оптимізація нейромережевої системи детекції БПЛА для вбудованих обчислювачів засобів перехоплення	Шифр групи	Код напр./спец.
Перевірив		Головянко М.В.				ІТШІ-22-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					