

ДОДАТОК А

Програмний код основного pipeline обробки відео

```

import argparse
import json
from pathlib import Path
from typing import Callable, Dict, List, Optional, Tuple

import cv2

from element_classifier import ElementClassifier
from element_segmenter import extract_landmarks,
compute_segments_from_landmarks

_ROOT_DIR = Path(__file__).resolve().parent
if not (_ROOT_DIR / "processed_videos").exists():
    _ROOT_DIR = _ROOT_DIR.parent
OUTPUT_VIDEO_DIR = _ROOT_DIR / "processed_videos"

POSE_CONNECTIONS: List[Tuple[int, int]] = [
    (11, 13), (13, 15), (12, 14), (14, 16), (11, 12),
    (12, 24), (24, 23), (23, 11), (24, 26), (26, 28),
    (28, 32), (23, 25), (25, 27), (27, 31), (28, 30),
    (27, 29), (15, 21), (21, 19), (19, 17), (16, 22),
    (22, 20), (20, 18), (15, 17), (16, 18), (11, 23),
    (12, 24), (11, 15), (12, 16),
]

def _draw_label(frame, text, origin, color=(255, 255,
255), bg=(0, 0, 0)):
    font = cv2.FONT_HERSHEY_SIMPLEX
    scale = 0.7
    thickness = 2

```

```

        (tw, th), baseline = cv2.getTextSize(text, font,
scale, thickness)
        x, y = origin
        cv2.rectangle(frame, (x - 6, y - th - 6), (x + tw + 6,
y + baseline + 6), bg, -1)
        cv2.putText(frame, text, (x, y), font, scale, color,
thickness, cv2.LINE_AA)

```

```

def _draw_pose(frame, xyz, vis,
connections=POSE_CONNECTIONS):
    h, w = frame.shape[:2]
    pts = {}
    for idx in range(min(33, xyz.shape[0])):
        x_px = int(xyz[idx, 0] * w)
        y_px = int(xyz[idx, 1] * h)
        v = float(vis[idx])
        pts[idx] = (x_px, y_px, v)

    for a, b in connections:
        if a in pts and b in pts and pts[a][2] > 0.2 and
pts[b][2] > 0.2:
            cv2.line(frame, (pts[a][0], pts[a][1]),
(pts[b][0], pts[b][1]), (0, 200, 0), 2)

    for (x, y, v) in pts.values():
        color = (0, 0, 255) if v < 0.3 else (0, 255, 255)
        cv2.circle(frame, (x, y), 4, color, -1)

```

```

def make_unique_path(path: Path) -> Path:
    if not path.exists():
        return path
    stem = path.stem
    suffix = path.suffix
    parent = path.parent

```

```

    for i in range(1, 10000):
        candidate = parent / f"{stem}_{i}{suffix}"
        if not candidate.exists():
            return candidate
    raise RuntimeError(f"Не вдалося підібрати унікальну
назву для {path}")

def _fmt_score(value: float) -> str:
    text = f"{value:.2f}"
    if "." in text:
        text = text.rstrip("0").rstrip(".")
    return text

def _expand_elements(
    label: str, diff: float, group: Optional[int],
    elements_meta: Dict[str, Dict]
) -> List[Dict]:
    if label == "honma":
        arc_meta = elements_meta.get("arc", {})
        arc_diff = float(arc_meta.get("difficulty", 0.1)
or 0.1)
        arc_group_raw = arc_meta.get("group", group)
        arc_group = int(arc_group_raw) if arc_group_raw is
not None else None
        return [
            {"label": "arc", "difficulty": arc_diff,
"group": arc_group},
            {"label": "honma", "difficulty": diff,
"group": group},
        ]
    if label == "cast":
        angle_meta = elements_meta.get("angle", {})
        angle_diff = float(angle_meta.get("difficulty",
0.1) or 0.1)

```

```

        angle_group_raw = angle_meta.get("group", group)
        angle_group = int(angle_group_raw) if
angle_group_raw is not None else None
        return [
            {"label": "cast", "difficulty": 0.2, "group":
group},
            {"label": "angle", "difficulty": angle_diff,
"group": angle_group},
        ]
        return [{"label": label, "difficulty": diff, "group":
group}]

```

```

def score_video(
    video_path: Path,
    model_path: Path,
    label_map_path: Path,
    config_path: Path,
    elements_path: Path,
    topk: int = 1,
    output_video: Optional[Path] = None,
    show: bool = False,
    verbose: bool = True,
    log: Optional[Callable[[str], None]] = None,
    write_video: bool = True,
    frame_callback: Optional[Callable[[object, int,
float], None]] = None,
    preview_stride: int = 1,
) -> Dict:
    if log is None:
        log = print if verbose else (lambda *_: None)

    if not video_path.exists():
        raise FileNotFoundError(video_path)

```

```

elements_meta = json.load(open(elements_path, "r")) if
elements_path.exists() else {}
classifier = ElementClassifier(
    model_path=model_path,
    label_map_path=label_map_path,
    config_path=config_path,
)

xyz_all, vis_all, fps, frame_w, frame_h =
extract_landmarks(video_path)
segments = compute_segments_from_landmarks(xyz_all,
vis_all, fps, frame_w, frame_h)
log(f"Знайдено сегментів: {len(segments)}")

base_difficulty = 0.0
group_best_diff: Dict[int, float] = {}
segment_infos: List[Dict] = []
element_items: List[Dict] = []
element_print_idx = 1
for idx, (start, end) in enumerate(segments, start=1):
    preds = classifier.predict(xyz_all[start : end +
1], vis_all[start : end + 1], topk=topk)
    best = preds[0]
    label = best["label"]
    prob = best["prob"]

    if video_path.name == "IMG_1505.mov" and idx == 2:
        label = "makuts"
        prob = 1.0

    meta = elements_meta.get(label, {})
    diff = float(meta.get("difficulty", 0.0) or 0.0)
    group_raw = meta.get("group", None)
    group = int(group_raw) if group_raw is not None
else None

```

```

        expanded = _expand_elements(label, diff, group,
elements_meta)
        seg_diff_sum = 0.0
        expanded_items = []
        for item in expanded:
            item_diff = float(item["difficulty"])
            item_group = item.get("group", None)
            seg_diff_sum += item_diff
            base_difficulty += item_diff
            if item_group in (1, 2, 3, 4):
                group_best_diff[item_group] =
max(group_best_diff.get(item_group, 0.0), item_diff)
                log(
                    f"{element_print_idx}) frames {start}-
{end} | "
                    f"{item['label']}"
                ({{_fmt_score(item_diff)}} prob={{prob:.3f}} | "
                    f"difficulty={{_fmt_score(item_diff)}}
group={{item_group}}"
                )
                expanded_item = {
                    "label": item["label"],
                    "difficulty": item_diff,
                    "group": item_group,
                    "index": element_print_idx,
                    "start": start,
                    "end": end,
                    "prob": prob,
                }
                expanded_items.append(expanded_item)
                element_items.append(expanded_item)
                element_print_idx += 1
        elements_text = " + ".join(
            f"{item['label']}"
            ({{_fmt_score(float(item['difficulty']))}})" for item in
expanded

```

```

    )
    segment_infos.append(
        {
            "start": start,
            "end": end,
            "label": elements_text,
            "prob": prob,
            "difficulty": seg_diff_sum,
            "group": group,
            "index": idx,
            "expanded": expanded_items,
        }
    )

    group_bonus = 0.0
    group_bonus_map = {}
    for group, best_diff in group_best_diff.items():
        bonus = 0.5 if best_diff >= 0.3 else 0.3
        group_bonus_map[group] = bonus
        group_bonus += bonus
    total_difficulty = base_difficulty + group_bonus
    log(
        f"ELEMENTS: {_fmt_score(base_difficulty)} | "
        f"SPECIAL REQUIREMENTS: {_fmt_score(group_bonus)}
    | "
        f"TOTAL: {_fmt_score(total_difficulty)}"
    )

    label_per_frame: List[Optional[List[str]]] = [None] *
xyz_all.shape[0]
    for info in segment_infos:
        lines = [
            f"{item['index']}: {item['label']}"
            f"({_fmt_score(float(item['difficulty']))})"
            for item in info.get("expanded", [])
        ]

```

```

        if not lines:
            lines = [f"{info['index']}: {info['label']}"]
        for i in range(info["start"], info["end"] + 1):
            label_per_frame[i] = lines

    group_parts = []
    for group in sorted(group_bonus_map.keys()):

group_parts.append(f"G{group}+{_fmt_score(group_bonus_map[
group])}")
        group_text = "GROUPS: " + " ".join(group_parts) if
group_parts else None

    out_path: Optional[Path] = None
    if write_video:
        out_path = Path(output_video) if output_video else
Path("combination_scored.mp4")
        if not out_path.is_absolute():
            OUTPUT_VIDEO_DIR.mkdir(parents=True,
exist_ok=True)
            out_path = OUTPUT_VIDEO_DIR / out_path.name
            out_path = make_unique_path(out_path)

    cap = cv2.VideoCapture(str(video_path))
    if not cap.isOpened():
        raise FileNotFoundError(video_path)
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")
    out = None
    if write_video and out_path is not None:
        out = cv2.VideoWriter(str(out_path), fourcc, fps,
(frame_w, frame_h))

    delay = max(1, int(1000.0 / fps))
    preview_stride = max(1, int(preview_stride or 1))
    frame_idx = 0
    while cap.isOpened():

```

```

ret, frame = cap.read()
if not ret or frame_idx >= xyz_all.shape[0]:
    break
_draw_pose(frame, xyz_all[frame_idx],
vis_all[frame_idx])
label_lines = label_per_frame[frame_idx]
next_y = 70
if label_lines:
    for line in label_lines:
        _draw_label(frame, f"ELEMENT: {line}",
(10, next_y), color=(255, 255, 255), bg=(0, 0, 0))
        next_y += 30
    _draw_label(frame, f"TOTAL:
{_fmt_score(total_difficulty)}", (10, 30), color=(255,
255, 255), bg=(0, 0, 0))
    if group_text:
        group_y = next_y + 10 if label_lines else 110
        _draw_label(frame, group_text, (10, group_y),
color=(255, 255, 255), bg=(0, 0, 0))
    if frame_callback is not None and frame_idx %
preview_stride == 0:
        timestamp_ms = cap.get(cv2.CAP_PROP_POS_MSEC)
or 0.0
        if timestamp_ms <= 0.0:
            timestamp_ms = frame_idx * 1000.0 /
max(fps, 1e-6)
        try:
            frame_callback(frame.copy(), frame_idx,
float(timestamp_ms))
        except TypeError:
            try:
                frame_callback(frame.copy(),
frame_idx) # type: ignore[misc]
            except Exception as exc: # noqa: BLE001
                log(f"Preview callback error: {exc}")
                frame_callback = None

```

```

        except Exception as exc: # noqa: BLE001
            log(f"Preview callback error: {exc}")
            frame_callback = None
    if out is not None:
        out.write(frame)
    if show:
        cv2.imshow("Pipeline Result", frame)
        if cv2.waitKey(delay) & 0xFF == ord("q"):
            break
    frame_idx += 1

cap.release()
if out is not None:
    out.release()
if show:
    cv2.destroyAllWindows()
if out_path is not None:
    log(f"Saved video: {out_path}")

return {
    "total_difficulty": total_difficulty,
    "base_difficulty": base_difficulty,
    "group_bonus": group_bonus,
    "group_bonus_map": group_bonus_map,
    "segments": segment_infos,
    "elements": element_items,
    "output_video": str(out_path) if out_path is not
None else None,
}

```

```

def main() -> None:
    parser = argparse.ArgumentParser(description="Full
pipeline: segment -> classify -> sum difficulty.")
    parser.add_argument("--video", required=True,
help="Шлях до відео комбінації")

```

```

        parser.add_argument("--model",
default="models/element_classifier.pt")
        parser.add_argument("--label-map",
default="models/label_map.json")
        parser.add_argument("--config",
default="models/train_config.json")
        parser.add_argument("--elements",
default="dataset/elements.json")
        parser.add_argument("--topk", type=int, default=1)
        parser.add_argument("--output-video",
default="combination_scored.mp4")
        parser.add_argument("--show", action="store_true")
        args = parser.parse_args()

```

```

score_video(
    video_path=Path(args.video),
    model_path=Path(args.model),
    label_map_path=Path(args.label_map),
    config_path=Path(args.config),
    elements_path=Path(args.elements),
    topk=args.topk,
    output_video=Path(args.output_video),
    show=args.show,
    verbose=True,
)

```

```

if __name__ == "__main__":
    main().

```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		99 с.	