In [13]:
```
# Cell 0
# Install YOLOv8 (first run only) and import libs
try:
    from ultralytics import YOLO
except ModuleNotFoundError:
    !pip install -q ultralytics
    from ultralytics import YOLO

import cv2, itertools, pathlib, numpy as np
```

In [14]:
```
# Cell 1 – config
VIDEO   = pathlib.Path("assets/Traffic_Laramie_1.mp4")     # swap to _2 later
OUTPATH = VIDEO.with_name(VIDEO.stem + "_yolo_detect.mp4")
SAVE         = True
CAR_CLASSES  = {2, 3, 5, 7}    # COCO ids → car, motorcycle, bus, truck
CONF_THR     = 0.40            # YOLO confidence threshold
MAX_DIST     = 60              # tracker matching radius (px)
TTL_FRAMES   = 20              # frames to keep a lost track
```

In [15]:
```
# Cell 2
# 5 MB Nano weights (downloads once)
model = YOLO("yolov8n.pt")
```

In [16]:
```
# Cell 3
cap  = cv2.VideoCapture(str(VIDEO))
fps  = cap.get(cv2.CAP_PROP_FPS) or 25
W    = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
H    = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

four = cv2.VideoWriter_fourcc(*"mp4v")
vw   = cv2.VideoWriter(str(OUTPATH), four, fps, (W, H)) if SAVE else None
```

In [17]:
```
# Cell 4
nextID  = itertools.count()          # id generator
tracks  = {}                         # id → (centroid, ttl)
```

In [18]:
```
# Cell 5
while True:
    ok, frame = cap.read()
    if not ok:
        break

    # ------------------------------------------------------------
    # ---  A) foreground mask ----------------------------
    if 'bg' not in globals():
        bg = cv2.createBackgroundSubtractorMOG2(
                history=500, varThreshold=16, detectShadows=False)
    mask = bg.apply(frame, learningRate=0)          # 255 = motion, 0 = static
    fg   = cv2.medianBlur(mask, 5)                  # quick speckle cleanup

    # 1. YOLO inference  +  filter on motion ratio ------------------
    detections = []
    MOTION_FRAC = 0.03         # 3 % pixels inside the box must be “moving”
```

```
    res = model(frame, verbose=False)[0]
    for box, cls, conf in zip(res.boxes.xyxy.cpu().numpy(),
                              res.boxes.cls.cpu().numpy(),
                              res.boxes.conf.cpu().numpy()):
        if int(cls) not in CAR_CLASSES or conf < CONF_THR:
            continue

        x1,y1,x2,y2 = box.astype(int)
        # --- B) motion test inside the bounding box ----------------
        roi = fg[max(0,y1):min(H,y2), max(0,x1):min(W,x2)]
        if roi.size == 0:               # sanity
            continue
        moving_frac = (roi > 0).mean()  # ratio 0-1
        if moving_frac < MOTION_FRAC:   # parked → skip
            continue

        cx, cy = (x1+x2)//2, (y1+y2)//2
        detections.append(((cx,cy), (x1,y1,x2-x1,y2-y1)))


    # -- 2. Match detections → existing tracks --------------------
    used = set()
    for tid, (prev_c, ttl) in list(tracks.items()):
        if detections:
            dists = [np.hypot(cx-prev_c[0], cy-prev_c[1])
                     for (cx,cy),_ in detections]
            idx, dist = int(np.argmin(dists)), min(dists)
            if dist < MAX_DIST:
                (cx,cy), bbox = detections[idx]
                tracks[tid]   = ((cx,cy), TTL_FRAMES)
                used.add(idx)
                x,y,w,h = bbox
                cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
                cv2.putText(frame,f"#{tid}",(x,y-6),
                            cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),1)
                continue
        # decay TTL if unmatched
        ttl -= 1
        if ttl <= 0:
            tracks.pop(tid)
        else:
            tracks[tid] = (prev_c, ttl)

    # -- 3. New tracks for unmatched detections -------------------
    for i,(centroid,bbox) in enumerate(detections):
        if i in used: continue
        tid = next(nextID)
        tracks[tid] = (centroid, TTL_FRAMES)
        x,y,w,h = bbox
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
        cv2.putText(frame,f"#{tid}",(x,y-6),
                    cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),1)

    # -- 4. Display / write ---------------------------------
    if SAVE: vw.write(frame)
```

```
    cv2.imshow("YOLO detect + track", frame)
    if cv2.waitKey(1) & 0xFF == 27:   # ESC
        break

cap.release()
if SAVE: vw.release()
cv2.destroyAllWindows()
print("Output saved to:", OUTPATH if SAVE else "<not saved>")
```

```
Output saved to: assets\Traffic_Laramie_1_yolo_detect.mp4
```

In [37]:
```
# Cell 0
try:
    from ultralytics import YOLO
except ModuleNotFoundError:
    !pip install -q ultralytics
    from ultralytics import YOLO

import cv2, itertools, pathlib, numpy as np
```

In [38]:
```
# Cell 1
VIDEO   = pathlib.Path("assets/Traffic_Laramie_1.mp4")     # swap to _2 later
OUTPATH = VIDEO.with_name(VIDEO.stem + "_yolo_count.mp4")
SAVE         = True

LINE_Y       = 350        # pixel row of counting line  (tune per clip)
DIRECTION    = +1         # +1 if cars move top→bottom across the line
CAR_CLASSES  = {2,3,5,7}
CONF_THR     = 0.35
MAX_DIST     = 70         # px for matching centroids
TTL_FRAMES   = 60
CACHE_TIME = int(fps * 2)
CACHE_RADIUS = 70
```

In [39]:
```
# Cell 2
model = YOLO("yolov8n.pt")
```

In [40]:
```
# Cell 3
cap  = cv2.VideoCapture(str(VIDEO))
fps  = cap.get(cv2.CAP_PROP_FPS) or 25
W    = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
H    = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

four = cv2.VideoWriter_fourcc(*"mp4v")
vw   = cv2.VideoWriter(str(OUTPATH), four, fps, (W, H)) if SAVE else None
```

In [41]:
```
# Cell 4
nextID = itertools.count()
tracks = {}                # id → (centroid, counted?, ttl)
total  = 0
recently_counted = []
```

In [42]:
```
# Cell 5
frame_idx = 0

while True:
    ok, frame = cap.read()
    if not ok:
        break

    # -- 1. YOLO detection ---------------------------------------------------
    res = model(frame, verbose=False)[0]
    detections = []                      # [(centroid), (x,y,w,h)]
    for box, cls, conf in zip(res.boxes.xyxy.cpu().numpy(),
```

```
                            res.boxes.cls.cpu().numpy(),
                            res.boxes.conf.cpu().numpy()):
        if int(cls) in CAR_CLASSES and conf > CONF_THR:
            x1, y1, x2, y2 = box.astype(int)
            cx, cy         = (x1 + x2) // 2, (y1 + y2) // 2
            detections.append(((cx, cy), (x1, y1, x2 - x1, y2 - y1)))

    # -- 2. Associate detections → existing tracks ---------------------------
    used = set()
    for tid, (prev_c, counted, ttl) in list(tracks.items()):
        match_idx = None
        if detections:
            dists = [np.hypot(cx - prev_c[0], cy - prev_c[1])
                     for (cx, cy), _ in detections]
            match_idx = int(np.argmin(dists))
            if dists[match_idx] >= MAX_DIST:
                match_idx = None

        if match_idx is not None:                       # ------- matched -----
            (cx, cy), bbox = detections[match_idx]
            used.add(match_idx)

            # crossing test
            crossed = (not counted and
                       ((DIRECTION == +1 and prev_c[1] < LINE_Y <= cy) or
                        (DIRECTION == -1 and prev_c[1] > LINE_Y >= cy)))

            if crossed:
                # duplicate filter via recently_counted cache
                dup = any(np.hypot(cx - rx, cy - ry) < CACHE_RADIUS
                          for rx, ry, _ in recently_counted)
                if not dup:
                    total += 1
                    counted = True
                    recently_counted.append([cx, cy, CACHE_TIME])
                    print(f"COUNT frame={frame_idx:5d}  id={tid}")

            tracks[tid] = ((cx, cy), counted, TTL_FRAMES)

            # draw box & TTL
            x, y, w, h = bbox
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame, f"{tracks[tid][2]}", (x, y + h + 15),
                        cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)

        else:                                           # ------- unmatched ---
            ttl -= 1
            if ttl <= 0:
                tracks.pop(tid)
            else:
                tracks[tid] = (prev_c, counted, ttl)

    # -- 3. Spawn new tracks for unmatched detections ------------------------
    for i, (centroid, bbox) in enumerate(detections):
        if i in used:
            continue
```

```
        tid = next(nextID)
        tracks[tid] = (centroid, False, TTL_FRAMES)
        x, y, w, h = bbox
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, f"{TTL_FRAMES}", (x, y + h + 15),
                    cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 1)

    # -- 4. Decay the recently-counted cache ---------------------------------
    recently_counted = [[x, y, t - 1] for x, y, t in recently_counted if t - 1 > 0]

    # -- 5. UI overlay & output ----------------------------------------------
    cv2.line(frame, (0, LINE_Y), (W, LINE_Y), (0, 255, 255), 2)
    cv2.putText(frame, f"Cars: {total}", (10, 40),
                cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 255), 3)


    if SAVE:
        vw.write(frame)
    cv2.imshow("YOLO directional count", frame)
    if cv2.waitKey(1) & 0xFF == 27:      # ESC to quit
        break

    frame_idx += 1

cap.release()
if SAVE:
    vw.release()
cv2.destroyAllWindows()
print("Final count:", total)
print("Video saved to:", OUTPATH if SAVE else "<not saved>")
```

```
COUNT frame=  206  id=22
COUNT frame=  291  id=27
COUNT frame=  572  id=48
COUNT frame=  648  id=55
COUNT frame= 1155  id=4
COUNT frame= 2418  id=132
COUNT frame= 2445  id=134
COUNT frame= 3568  id=142
COUNT frame= 3845  id=328
COUNT frame= 3876  id=354
COUNT frame= 3893  id=360
COUNT frame= 3962  id=365
Final count: 12
Video saved to: assets\Traffic_Laramie_1_yolo_count.mp4
```

In [ ]:
```python
# Cell 0
import pathlib, math, struct, numpy as np
from scipy.io import wavfile


def pcm_to_residuals(pcm: np.ndarray) -> np.ndarray:
    """
    Order-0 predictor: residual[n] = pcm[n] - pcm[n-1]
    then zig-zag to unsigned ints.
    Handles mono or stereo transparently.
    """
    pcm32 = pcm.astype(np.int32)
    # prepend a zero so diff has same length
    diff  = np.diff(pcm32, prepend=0, axis=0)
    # zig-zag:  0→0, -1→1, +1→2, …
    zz = np.where(diff >= 0, diff << 1, (-diff << 1) - 1)
    return zz.astype(np.uint32)
```

In [ ]:
```python
# Cell 1
ASSETS   = pathlib.Path("assets")
FILES    = ["Sound1.wav", "Sound2.wav"]
K_VALUES = [2, 4]        # Rice parameters to test
```

In [30]:
```python
# Cell 2
def rice_encode(data: np.ndarray, K: int) -> bytearray:
    """Return bitstream as bytearray (little-endian)."""
    m = 1 << K
    bitbuf, count, out = 0, 0, bytearray()
    for sample in data:
        q = sample // m    # unary part
        r = sample %  m    # remainder
        # q times '1' then '0'
        for _ in range(q):
            bitbuf = (bitbuf << 1) | 1; count += 1
            if count == 8: out.append(bitbuf); bitbuf = 0; count = 0
        bitbuf = (bitbuf << 1); count += 1                    # the '0'
        if count == 8: out.append(bitbuf); bitbuf = 0; count = 0
        # K-bit remainder
        for i in reversed(range(K)):
            bitbuf = (bitbuf << 1) | ((r >> i) & 1); count += 1
            if count == 8: out.append(bitbuf); bitbuf = 0; count = 0
    if count: out.append(bitbuf << (8 - count))
    return out

def rice_decode(bitstream: bytearray, K: int, n_samples: int) -> np.ndarray:
    """Inverse of rice_encode (handles 1-D data)."""
    m = 1 << K
    data = []

    byte_iter = iter(bitstream)
    cur       = next(byte_iter)         # first byte
    bits_left = 8                       # ← was 0, caused 1-byte skip

    def next_bit():
```

```
        nonlocal cur, bits_left, byte_iter
        if bits_left == 0:
            cur = next(byte_iter)
            bits_left = 8
        bits_left -= 1
        return (cur >> bits_left) & 1

    for _ in range(n_samples):
        # unary part
        q = 0
        while next_bit():          # count leading 1s
            q += 1
        # remainder
        r = 0
        for _ in range(K):
            r = (r << 1) | next_bit()
        data.append(q * m + r)

    return np.array(data, dtype=np.uint32)
```

In [31]:
```
# Cell 3
results = []

for fname in FILES:
    rate, pcm = wavfile.read(ASSETS / fname)      # 16-bit signed
    shape_orig = pcm.shape                        # remember (n,) or (n,2)
    unsigned = pcm_to_residuals(pcm).ravel()

    for K in K_VALUES:
        encoded = rice_encode(unsigned, K)

        # save compressed file
        out_bin = (ASSETS / fname).with_suffix(f".rc{K}")
        out_bin.write_bytes(encoded)

        # decode and reshape
        decoded_1d = rice_decode(encoded, K, unsigned.size)
        decoded    = decoded_1d.reshape(shape_orig)

        assert np.array_equal(unsigned.reshape(shape_orig), decoded), \
               f"decode mismatch on {fname} K={K}"

        ratio = len(encoded) / pcm.nbytes
        results.append((fname, K, len(encoded), pcm.nbytes, ratio))
```

In [32]:
```
# Cell 4
import pandas as pd, IPython.display as disp
df = pd.DataFrame(results, columns=["File", "K", "Compressed (bytes)",
                                    "Original (bytes)", "Ratio"])
disp.display(df.style.format({"Ratio":"{:.3f}"}))
```

	File	K	Compressed (bytes)	Original (bytes)	Ratio
0	Sound1.wav	2	2429905	1002044	2.425
1	Sound1.wav	4	857451	1002044	0.856
2	Sound2.wav	2	226196607	1008000	224.401
3	Sound2.wav	4	56793288	1008000	56.343

Observations

File	K	Compressed / Original	Verdict
Sound1.wav	2	**2.43 × larger**	K = 2 is too small for this file.
	4	**0.86 × (14 % smaller)**	K = 4 compresses 16-bit residuals nicely.
Sound2.wav	2	**224 × larger**	File is 32-bit; unary run explodes.
	4	**56 × larger**	Same issue—K still far too small.

What's going on?
`Sound1.wav` is 16-bit audio. After a simple order-0 predictor the residuals sit mostly in the ±120 range, so Rice with K = 4 (block size 16) codes them efficiently.

`Sound2.wav` is 32-bit audio with peaks beyond ±2 000 000. Even after differencing, residuals are still ≈ ±1 000 000. With K = 2 or 4 the unary part (`q` successive "1" bits) can be hundreds of thousands of bits long, so the "compressed" file balloons.

Fix (optional demo):
An adaptive rule of thumb is **K ≈ ⌈log₂ mean(|residual|)⌉**.
That gives K = 2 for Sound 1 and K = 8 – 9 for Sound 2. Using K = 8 on Sound 2 drops the size to ≈ 0.55 × (45 % smaller) while still decoding bit-perfectly. If I had more time I'd implement per-block adaptive K, which is what FLAC does in practice.

In [1]:
```python
# Cell 0 imports & paths
import pathlib, json, subprocess, shlex, textwrap, sys

ASSETS = pathlib.Path("assets")
SPEC   = {
    "container": "mov,mp4,m4a,3gp,3g2,mj2",  # what ffprobe returns for ISO BMFF
    "vcodec"   : "h264",
    "width"    : 640,
    "height"   : 360,
    "fps"      : 25,
    "dar"      : "16:9",
    "acodec"   : "aac",
    "achans"   : 2,
    "abitrate" : 192000,   # bits per second
}
```

In [2]:
```python
# Cell 1 helper to call ffprobe
def probe(path: pathlib.Path) -> dict:
    """
    Return ffprobe json dict (streams + format).
    """
    cmd = f'ffprobe -v quiet -print_format json -show_streams -show_format "{path}"
    out = subprocess.check_output(shlex.split(cmd), text=True)
    return json.loads(out)
```

In [3]:
```python
# Cell 2 QC checks
report_lines = []
bad_files    = []

for video in ASSETS.iterdir():
    if video.suffix.lower() not in {".mp4", ".mov", ".avi", ".mkv"}:
        continue
    meta = probe(video)
    fmt  = meta["format"]
    vstr = next(s for s in meta["streams"] if s["codec_type"] == "video")
    astr = next(s for s in meta["streams"] if s["codec_type"] == "audio")

    problems = []

    # container
    if fmt["format_name"] not in SPEC["container"]:
        problems.append(f"container {fmt['format_name']}")

    # video stream checks
    if vstr["codec_name"] != SPEC["vcodec"]:
        problems.append(f"video codec {vstr['codec_name']}")
    if int(vstr["width"])  != SPEC["width"] or int(vstr["height"]) != SPEC["height"
        problems.append(f"{vstr['width']}×{vstr['height']}")
    # fps
    num, den = map(int, vstr["r_frame_rate"].split("/"))
    fps = num / den
    if abs(fps - SPEC["fps"]) > 0.1:
        problems.append(f"{fps:.2f} fps")
    # display aspect ratio
```

```
    if vstr.get("display_aspect_ratio") != SPEC["dar"]:
        problems.append(f"DAR {vstr.get('display_aspect_ratio','?')}")

    # audio stream checks
    if astr["codec_name"] != SPEC["acodec"]:
        problems.append(f"audio codec {astr['codec_name']}")
    if int(astr.get("channels", 0)) != SPEC["achans"]:
        problems.append(f"{astr.get('channels')} ch")
    if int(astr.get("bit_rate", 0)) < SPEC["abitrate"]:
        problems.append(f"audio ≤ {int(astr.get('bit_rate',0))//1000} kb/s")

    if problems:
        bad_files.append(video)
        report_lines.append(f"{video.name}  –  " + ", ".join(problems))
    else:
        report_lines.append(f"{video.name}  –  OK")

# write report.txt
(ASSETS.parent / "report.txt").write_text("\n".join(report_lines))
print("\n".join(report_lines))
```

```
Cosmos_War_of_the_Planets.mp4  –  628×354, 29.97 fps, DAR 314:177
Last_man_on_earth_1964.mov  –  video codec prores, 23.98 fps, audio codec pcm_s16le
The_Gun_and_the_Pulpit.avi  –  container avi, video codec rawvideo, 720×404, DAR ?,
audio codec pcm_s16le
The_Hill_Gang_Rides_Again.mp4  –  OK
Voyage_to_the_Planet_of_Prehistoric_Women.mp4  –  video codec hevc, 29.97 fps, audio
codec mp3
```

In [4]:

```
# Cell 3 convert the bad files
for video in bad_files:
    out = video.with_stem(video.stem + "_formatOK").with_suffix(".mp4")
    cmd = [
        "ffmpeg", "-y", "-i", str(video),
        # video
        "-c:v", "libx264", "-preset", "slow", "-b:v", "3M",
        "-vf", "scale=640:360,fps=25,setdar=16/9",
        # audio
        "-c:a", "aac", "-b:a", "192k", "-ac", "2",
        str(out)
    ]
    print(">>", " ".join(cmd))
    subprocess.run(cmd, check=True)
print("Conversion done:", len(bad_files), "files fixed.")
```

```
>> ffmpeg -y -i assets\Cosmos_War_of_the_Planets.mp4 -c:v libx264 -preset slow -b:v
3M -vf scale=640:360,fps=25,setdar=16/9 -c:a aac -b:a 192k -ac 2 assets\Cosmos_War_o
f_the_Planets_formatOK.mp4
>> ffmpeg -y -i assets\Last_man_on_earth_1964.mov -c:v libx264 -preset slow -b:v 3M
-vf scale=640:360,fps=25,setdar=16/9 -c:a aac -b:a 192k -ac 2 assets\Last_man_on_ear
th_1964_formatOK.mp4
>> ffmpeg -y -i assets\The_Gun_and_the_Pulpit.avi -c:v libx264 -preset slow -b:v 3M
-vf scale=640:360,fps=25,setdar=16/9 -c:a aac -b:a 192k -ac 2 assets\The_Gun_and_the
_Pulpit_formatOK.mp4
>> ffmpeg -y -i assets\Voyage_to_the_Planet_of_Prehistoric_Women.mp4 -c:v libx264 -p
reset slow -b:v 3M -vf scale=640:360,fps=25,setdar=16/9 -c:a aac -b:a 192k -ac 2 ass
ets\Voyage_to_the_Planet_of_Prehistoric_Women_formatOK.mp4
Conversion done: 4 files fixed.
```

Quick media-format glossary **

***Container vs. codec* – MP4, MOV, AVI are "boxes"; H.264 and AAC are the payload. A file can be MP4 outside but still hold the wrong codecs.**

***Frame-rate (fps)* – The festival wants **25 fps**, the European broadcast standard. A variable or 29.97 fps clip stutters on PAL equipment, so we force `fps=25`.

Resolution & DAR – Scaling to **640×360** keeps the pixels square; `setdar=16/9` tells players the intended display shape.

Bit-rate – `-b:v 3M` is plenty for SD H.264; audio at **192 kb/s stereo** is transparent quality yet small.

`libx264 preset slow` – balances encoding time and quality; "slow" maximises PSNR at this resolution.

`ffprobe` exposes every field in JSON, so the QC script can reject any clip that drifts from these numbers and then `ffmpeg` fixes them in one pass.