```python
import cv2
import numpy as np
import logging
from typing import Generator, Tuple
from src.config import AppConfig

logger = logging.getLogger(__name__)

def parse_roi(roi_str: str) -> Tuple[int, int, int, int]:
    """
    Parses 'x,y,w,h' string into a tuple of integers.
    """
    try:
        parts = [int(p.strip()) for p in roi_str.split(',')]
        if len(parts) != 4:
            raise ValueError
        return tuple(parts) # type: ignore
    except (ValueError, AttributeError):
        raise ValueError(f"Invalid ROI format. Expected 'x,y,w,h', got '{roi_str}'")

class VideoLoader:
    def __init__(self, config: AppConfig):
        self.config = config

    def get_frames(self) -> Generator[np.ndarray, None, None]:
        """
        Yields preprocessed frames from the video file.
        """
        video_path = str(self.config.video_path)
        logger.info(f"Opening video file: {video_path}")
        cap = cv2.VideoCapture(video_path)

        try:
            if not cap.isOpened():
                raise IOError(f"Cannot open video file: {self.config.video_path}")

            total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
            logger.info(f"Video opened successfully. Total frames: {total_frames}")

            while True:
                ret, frame = cap.read()
                if not ret:
                    logger.info("End of video reached or failed to read frame.")
                    break

                # Check if frame is valid
                if frame is None or frame.size == 0:
                    logger.warning("Empty frame read.")
                    continue

                processed_frame = self._preprocess(frame)
                yield processed_frame

        finally:
            cap.release()
            logger.info("VideoLoader released.")

    def _preprocess(self, frame: np.ndarray) -> np.ndarray:
        """
        Applies ROI crop, Resize (if needed), and Color conversion.
        """
        x, y, w, h = self.config.roi

        # 1. ROI Crop (Safe slicing)
        # Note: frame shape is (height, width, channels)
        # Slicing: frame[y:y+h, x:x+w]
        roi_frame = frame[y:y+h, x:x+w]

        if roi_frame.size == 0:
             # ROI가 이미지 범위를 벗어난 경우 빈 프레임 반환 가능성 대비
             # 실제로는 설정 단계에서 막아야 하지만 안전장치
             return roi_frame

        # 2. Resize Logic (Match C++ implementation)
        # if (roi.cols >= 320) ...
        rows, cols = roi_frame.shape[:2]
        if cols >= 320:
            target_x = 320.0
            scale = target_x / cols
            # cv2.resize takes (width, height)
            new_width = int(cols * scale)
            new_height = int(rows * scale)
            roi_frame = cv2.resize(roi_frame, (new_width, new_height), interpolation=cv2.INTER_LINEAR)

        # 3. BGR to RGB (MediaPipe requires RGB)
        # C++ 코드에서는 BGR 그대로 MP로 넘기는 듯 했으나(OpenCV 기본),
        # MediaPipe Python API는 명시적으로 RGB 입력을 권장합니다.
```

```python
        # 원본 분석 결과: dst_img = pDlg->m_cap_img.clone(); ... cvtColor(..., COLOR_BGRA2BGR)
        # C++ 코드는 BGRA 처리만 있고 BGR->RGB 변환이 명시적으로 안 보일 수 있으나,
        # Python MediaPipe는 RGB를 기대하므로 여기서 변환하는 것이 안전합니다.
        rgb_frame = cv2.cvtColor(roi_frame, cv2.COLOR_BGR2RGB)

        return rgb_frame

def calculate_optical_flow_value(prev_gray: np.ndarray, curr_gray: np.ndarray) -> float:
    """
    Calculates the average motion magnitude between two grayscale frames using Sparse Optical Flow (Lucas-Kanade).
    Matches the C++ implementation to ensure consistent thresholding behavior.

    Input: prev_gray, curr_gray (uint8 grayscale images)
    Output: float (average motion magnitude of tracked points)
    """
    if prev_gray is None or curr_gray is None:
        return 0.0

    if prev_gray.shape != curr_gray.shape:
        return 0.0

    # 1. Detect feature points to track (Sparse)
    # Parameters match typical C++ OpenCV defaults or common usage if not specified
    feature_params = dict(maxCorners=300,
                          qualityLevel=0.01,
                          minDistance=7,
                          blockSize=7)

    p0 = cv2.goodFeaturesToTrack(prev_gray, mask=None, **feature_params)

    # If no features found, motion is 0
    if p0 is None:
        return 0.0

    # 2. Calculate Optical Flow (Lucas-Kanade)
    lk_params = dict(winSize=(15, 15),
                     maxLevel=2,
                     criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

    p1, st, err = cv2.calcOpticalFlowPyrLK(prev_gray, curr_gray, p0, None, **lk_params)

    # 3. Calculate Average Motion of Valid Points
    if p1 is not None:
        # Select good points
        good_new = p1[st == 1]
        good_old = p0[st == 1]

        if len(good_new) == 0:
            return 0.0

        # Calculate Euclidean distances
        displacements = np.linalg.norm(good_new - good_old, axis=1)

        # Average magnitude
        avg_motion = np.mean(displacements)
        return float(avg_motion)

    return 0.0
```