```python
import cv2
import numpy as np
import torch
from PIL import Image
import tqdm as tqdm
import torch
from diffusers import AutoPipelineForInpainting
from diffusers.utils import load_image, make_image_grid

pipeline = AutoPipelineForInpainting.from_pretrained(
    "runwayml/stable-diffusion-inpainting", torch_dtype=torch.float16,variant="fp16"
)
pipeline.enable_model_cpu_offload()

def extract_frames(video_path):
    cap = cv2.VideoCapture(video_path)
    frames = []
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            frames.append(frame)
        else:
            break
    cap.release()
    return frames


def pad_frame(frame, target_width=1280, target_height=720):
    org_height, org_width = frame.shape[:2]
    scale = min(target_height / org_height, target_width / org_width)

    new_width = int(org_width * scale)
    new_height = int(org_height * scale)

    resized_frame = cv2.resize(frame, (new_width, new_height))
    new_frame = np.zeros((target_height, target_width, 3), dtype=np.uint8)

    y_offset = (target_height - new_height) // 2
    x_offset = (target_width - new_width) // 2
    new_frame[y_offset:y_offset + new_height, x_offset:x_offset + new_width] = resized_frame

    return new_frame, x_offset, new_width


def inpainting_frame(frame, mask, pipeline):
    # frame and mask are resized to 512x512 for inpainting
    frame_resized = cv2.resize(frame, (512, 512))
    mask_resized = cv2.resize(mask, (512, 512))

    # Convert to PIL Image
    frame_image = Image.fromarray(frame_resized)
    mask_image = Image.fromarray(mask_resized)

    try:
        # Perform inpainting
        inpainted_frame = pipeline(prompt=" ", image=frame_image, mask_image=mask_image).images[0]

        # Resize inpainted frame back to original size
        inpainted_frame = inpainted_frame.resize((frame.shape[1], frame.shape[0]))
        return np.array(inpainted_frame)
    except Exception as e:
        print(f"Error in inpainting frame: {e}")
        return frame

def assemble_frames_to_video(frames, output_video_path, fps=3):
    height, width, _ = frames[0].shape
```

```
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_video_path, fourcc, fps, (width, height))
    for frame in frames:
        out.write(frame)
    out.release()


def HD_Video_Conversion(input_video, output_video, frame_rate_reduction=1):

    frames = extract_frames(input_video)

    inpainted_frames = []
    for i, frame in enumerate(frames):
        if i % frame_rate_reduction == 0:  # Process frames based on reduction
            padded_frame, start_x, new_width = pad_frame(frame)
            mask = np.zeros_like(padded_frame[:, :, 0])
            mask[:, :start_x] = 1
            mask[:, start_x + new_width:] = 1
            inpainted_frame = inpainting_frame(padded_frame, mask, pipeline)
            inpainted_frames.append(inpainted_frame)

    if not output_video.endswith('.mp4'):
        output_video += '.mp4'
    assemble_frames_to_video(inpainted_frames, output_video)
    return output_video
```