```
import os
import cv2
import glob
import argparse
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage.feature import greycomatrix, greycoprops
from skimage import io, img_as_ubyte
from scipy.stats import entropy
from sklearn.ensemble import IsolationForest


def extract_features(image_path):
  """Compute basic thermal texture features for one image."""
  img = io.imread(image_path, as_gray=True)
  img_uint8 = img_as_ubyte(img)

  mean = np.mean(img)
  std = np.std(img)
  min_val = np.min(img)
```

```
    max_val = np.max(img)
    ent = entropy(np.histogram(img, bins=256)[0] + 1e-6)

    # Grey-level co-occurrence matrix (texture)
    glcm = greycomatrix(img_uint8, distances=[1], angles=[0], symmetric=True, normed=True)
    contrast = greycoprops(glcm, 'contrast')[0, 0]
    homogeneity = greycoprops(glcm, 'homogeneity')[0, 0]

    return [mean, std, min_val, max_val, ent, contrast, homogeneity]


def extract_frames(video_path, output_folder, step=10):
    """Extract every Nth frame from a video and save to folder."""
    os.makedirs(output_folder, exist_ok=True)
    cap = cv2.VideoCapture(video_path)
    frame_id = 0
    saved_frames = []

    if not cap.isOpened():
        raise IOError(f"Error: Could not open video file: {video_path}")

    print(f"[INFO] Extracting frames from {video_path} ...")
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if frame_id % step == 0:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            fname = os.path.join(output_folder, f"frame_{frame_id:05d}.png")
```

```
            cv2.imwrite(fname, gray)
            saved_frames.append(fname)
        frame_id += 1
    cap.release()
    print(f"[INFO] Saved {len(saved_frames)} frames.")
    return saved_frames


def main(video_path, frame_step, contamination):
    frames_dir = "frames_extracted"
    frame_files = extract_frames(video_path, frames_dir, step=frame_step)

    # Compute features
    print("[INFO] Computing features for frames ...")
    features = [extract_features(f) for f in frame_files]
    df = pd.DataFrame(features, columns=["mean", "std", "min", "max", "entropy", "contrast",
"homogeneity"])

    # Isolation Forest
    print("[INFO] Running Isolation Forest ...")
    model = IsolationForest(contamination=contamination, random_state=42)
    df["anomaly"] = model.fit_predict(df)

    anomalies = df[df["anomaly"] == -1]
    print(f"[INFO] Detected {len(anomalies)} anomalous frames out of {len(df)}")

    # Plot
    plt.figure(figsize=(10,4))
    plt.plot(df["mean"], label="Mean Temperature", color="blue")
```

```
  plt.scatter(anomalies.index, anomalies["mean"], color="red", label="Anomalies")
  plt.xlabel("Frame Index")
  plt.ylabel("Mean Pixel Value")
  plt.legend()
  plt.title("Thermal Anomaly Detection via Isolation Forest")
  plt.tight_layout()
  plt.show()

  # Save results
  anomalies_out = pd.DataFrame({
      "frame_index": anomalies.index,
      "frame_path": [frame_files[i] for i in anomalies.index]
  })
  anomalies_out.to_csv("anomalous_frames.csv", index=False)
  print("[INFO] Saved anomalous frame list to anomalous_frames.csv")

  print("\nSample anomalous frames:")
  print(anomalies_out.head())


if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.add_argument("--video", type=str, required=True, help="Path to the thermal video (.mp4)")
  parser.add_argument("--frame-step", type=int, default=10, help="Extract every Nth frame
(default: 10)")
  parser.add_argument("--contamination", type=float, default=0.02, help="Expected anomaly
fraction (default: 0.02)")
  args = parser.parse_args()
  main(args.video, args.frame_step, args.contamination)
```