リスト1　モデル学習プログラムtrain_autoencoder.py（抜粋）↓

```
: (省略)
    # ===== 新規: TFLite INT8(入出力ともにint8)を追加出力 =====
    def rep_ds():
        # 代表データは学習時の Z スコア空間([-K, K] へクリップ)で与える
        for i in range(min(len(X), 2048)):  # 上限で軽量化
            x = X[i].astype(np.float32).copy()
            # 念のためクリップ(学習作成済みXがZスコア済みである前提)
            x = np.clip(x, -K_CLIP, K_CLIP)
            yield [x.reshape(1, -1)]

    converter_i8 = tf.lite.TFLiteConverter.from_keras_model(autoencoder)
    converter_i8.optimizations = [tf.lite.Optimize.DEFAULT]
    converter_i8.representative_dataset = rep_ds
    converter_i8.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
    converter_i8.inference_input_type  = tf.int8
    converter_i8.inference_output_type = tf.int8
    tflite_model_i8 = converter_i8.convert()
    with open("autoencoder_model_int8.tflite", "wb") as f:
        f.write(tflite_model_i8)
    print("☑ モデルを autoencoder_model_int8.tflite に保存しました。")
```

リスト2　しきい値決定プログラムcalculate_threshould.py↓

```
import numpy as np
import tensorflow as tf
import os
# ファイル設定
X_TRAIN_PATH = "X_train.npy"
MODEL_PATH_FLOAT = "autoencoder_model.tflite"
MODEL_PATH_INT8  = "autoencoder_model_int8.tflite"
THRESHOLD_FP32_FILE = "threshold_fp32.txt"
THRESHOLD_INT8_FILE = "threshold_int8.txt"
SCALER_MEAN_PATH = "scaler_mean.npy"
SCALER_STD_PATH  = "scaler_std.npy"
PERCENTILE = 99
WINDOW_SIZE = 10
FEAT_DIM = 3
K_CLIP = 3.0  # Z-score clipping bound (symmetric)
: (省略)
def compute_threshold_fp32(X):
: (省略)
def compute_threshold_int8(X):
: (省略)
    interpreter = tf.lite.Interpreter(model_path=MODEL_PATH_INT8)
    interpreter.allocate_tensors()
    in_d = interpreter.get_input_details()[0]
    out_d = interpreter.get_output_details()[0]
: (省略)
    in_scale, in_zp = in_d['quantization']
    out_scale, out_zp = out_d['quantization']
: (省略)
    errors = []
    for i, x in enumerate(X):
        z = np.clip(x.astype(np.float32), -K_CLIP, K_CLIP).reshape(1, -1)
        q_in = np.round(z / in_scale + in_zp).astype(np.int8)
        interpreter.set_tensor(in_d['index'], q_in)
        interpreter.invoke()
        q_out = interpreter.get_tensor(out_d['index']).astype(np.int8)
        # dequantize to Z-score space then compute MSE
        y = out_scale * (q_out.astype(np.int32) - out_zp)
        x_deq = in_scale * (q_in.astype(np.int32) - in_zp)
        mse = np.mean((x_deq  - y) ** 2)
        errors.append(float(mse))
    thr = np.percentile(errors, PERCENTILE)
: (省略)
    return thr

def main():
: (省略)
    _ = load_scaler()  # 存在チェックのみ
    X = load_data()
: (省略)
    fp32_thr = compute_threshold_fp32(X)
    int8_thr  = compute_threshold_int8(X)
: (省略)
```

リスト3　異常検知プログラムdetect_anomaly.py（抜粋）↓

```
: 省略
# --- 定数 ---
WINDOW_SIZE = 10
FEAT_DIM = 3
THRESHOLD_FP32_FILE = "threshold_fp32.txt"
THRESHOLD_INT8_FILE = "threshold_int8.txt"
MODEL_PATH_FLOAT = "autoencoder_model.tflite"
```

```
8  MODEL_PATH_INT8  = "autoencoder_model_int8.tflite"
9  SCALER_MEAN_PATH = "scaler_mean.npy"
10  SCALER_STD_PATH  = "scaler_std.npy"
11  K_CLIP = 3.0  # Z-score clipping bound (symmetric)
12  def load_threshold(datatype: str) -> float:
13  : (省略)
14  def load_model(datatype: str):
15      model_path = MODEL_PATH_INT8 if datatype == "int8" else MODEL_PATH_FLOAT
16  : (省略)
17  def load_scaler():
18  : (省略)
19  def main(datatype: str):
20      THRESHOLD = load_threshold(datatype)
21      interpreter, in_d, out_d, model_path = load_model(datatype)
22      in_scale, in_zp = in_d['quantization']
23      out_scale, out_zp = out_d['quantization']
24  : (省略)
25      scaler_mean, scaler_std = load_scaler()
26      sensor = ADXL345()
27      queue = deque(maxlen=WINDOW_SIZE)
28  : (省略)
29      try:
30          # 0.1秒ごとの周期実行(処理時間を考慮)
31          next_time = time.time()
32          while True:
33              # 1) Ctrl-C ポーリングで停止指示を確認
34              # 2) ADXL345 から生データ取得
35              x, y, z = sensor.read_axes()
36              queue.append([x, y, z])
37              if len(queue) == WINDOW_SIZE:
38                  data = np.array(queue, dtype=np.float32)  # (W, 3)
39                  # 3) Z スコア正規化 → クリップ
40                  zscore = (data - scaler_mean) / (scaler_std + 1e-8)
41                  z_clip = np.clip(zscore, -K_CLIP, K_CLIP).flatten().reshape(1, -1).astype(np.float32)
42
43                  if datatype == "int8":
44                      # 4) INT8 へ量子化し、バッファへプッシュ
45                      q_in = np.round(z_clip / in_scale + in_zp).astype(np.int8)
46                      interpreter.set_tensor(in_d['index'], q_in)
47                      # 5) DNN 推論 (dnn_compute)
48                      interpreter.invoke()
49                      q_out = interpreter.get_tensor(out_d['index']).astype(np.int8)
50                      # 6) 出力を逆量子化して MSE 算出
51                      y_hat = out_scale * (q_out.astype(np.int32) - out_zp)
52                      x_deq = in_scale  * (q_in.astype(np.int32) - in_zp)
53                      mse = np.mean((x_deq - y_hat) ** 2)
54                  else:
55                      interpreter.set_tensor(in_d['index'], z_clip.astype(np.float32))
56                      # 5) DNN 推論 (dnn_compute)
57                      interpreter.invoke()
58                      # 6) 出力を逆量子化して MSE 算出
59                      y_hat = interpreter.get_tensor(out_d['index']).astype(np.float32)
60                      mse = np.mean((z_clip - y_hat) ** 2)
61                  # 7) しきい値で NORMAL / ANOMALY 判定 → シリアル出力 & LED 制御
62                  status = "ANOMALY" if mse > THRESHOLD else "NORMAL"
63                  print(f"{status} | MSE={mse:.6f}")
64              # --- 周期制御: 0.1秒ごとに実行 ---
65  : (省略)
```

リスト4　INT8データ形式の異常検知の実行結果↓

```
python detect_anomaly.py -datatype=int8
・ 異常検知開始 | datatype=int8 | model=autoencoder_model_int8.tflite | threshold=0.141297
☑ Normal | MSE=0.094837
:
☑ Normal | MSE=0.087547
・ Anomaly | MSE=0.422446
:
・ Anomaly | MSE=0.313000
☑ Normal | MSE=0.044899
:
```