リスト1　Pythonの仮想化環境の作成手順↓

```
sudo apt install python3-venv -y
python3 -m venv  tflite-env
source ~/tflite-env/bin/activate
pip3 install smbus2
pip install numpy==1.26.4
pip3 install --extra-index-url https://google-coral.github.io/py-repo/ tflite_runtime
```

リスト2　センサ・データの読み取りプログラム（read_sensor.py）↓
注：本章で紹介する異常検知のPythonプログラムは、すべてChatGPTとの対話を通じて作成しました。

```
import smbus2 as smbus
class ADXL345:
    def __init__(self, address=0x53, busnum=1):
        self.address = address
        self.bus = smbus.SMBus(busnum)
        self.bus.write_byte_data(self.address, 0x2D, 0x08)
    def read_axes(self):
        def read_word(reg):
            low = self.bus.read_byte_data(self.address, reg)
            high = self.bus.read_byte_data(self.address, reg + 1)
            val = (high << 8) | low
            return val - 65536 if val > 32767 else val
        x = read_word(0x32)
        y = read_word(0x34)
        z = read_word(0x36)
        return [x, y, z]
if __name__ == "__main__":
    sensor = ADXL345()
    while True:
        print(sensor.read_axes())
```

リスト3　データ収集プログラム（data_collect.py）↓

```
import os
import sys
import argparse
from datetime import datetime
import csv
import time
from read_sensor import ADXL345
: (省略)
sensor = ADXL345()
filename = output_file
duration = 60        # 記録する合計時間(秒)
interval = 0.1       # サンプリング間隔(秒)
log_interval = 5.0   # コンソール表示間隔(秒)
print(f"正常データ収集中({duration}秒)...")
with open(filename, mode='w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["timestamp", "x", "y", "z"])
    start = time.time()
    last_log = start
    while time.time() - start < duration:
        x, y, z = sensor.read_axes()
        ts = time.time()
        writer.writerow([ts, x, y, z])
        # 一定時間ごとに画面に表示
        if ts - last_log >= log_interval:
            print(f"{ts - start:.1f}s経過 - x={x}, y={y}, z={z}")
            last_log = ts
        time.sleep(interval)
print(f"正常データを {output_file} に保存しました。")
```

リスト4　リスト3の実行結果　↓

```
(tflite-env) xxxxxx@raspberrypi:~/dev/anomaly_detection/raspi5 $ python collect_data.py
・ 正常データ収集中(60秒)...
・ 5.0s経過 - x=-48, y=5, z=210
:
・ 55.3s経過 - x=-47, y=3, z=214
☑ 正常データを data/sensor_data_2508161414.csv に保存しました。
```

リスト5　データの前処理プログラム（prepare_data.py）↓

```
import os
import argparse
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
# 引数処理
: (省略)
# CSV読み込みと連結
all_data = []
```

```
for filename in sorted(os.listdir(input_dir)):
    if filename.endswith(".csv"):
        path = os.path.join(input_dir, filename)
        print(f"📥 読み込み中: {path}")
        df = pd.read_csv(path)
        if {'x', 'y', 'z'}.issubset(df.columns):
            xyz = df[['x', 'y', 'z']].values
            all_data.append(xyz)
        else:
            print(f"⚠ スキップ: {filename}(必要な列が不足)")
if not all_data:
    print("❌ 有効なCSVファイルが見つかりませんでした。")
    raise SystemExit(1)
xyz_all = np.concatenate(all_data, axis=0)
# 標準化(平均0・分散1)し、スケーラ統計を保存
scaler = StandardScaler()
xyz_scaled = scaler.fit_transform(xyz_all)
np.save("scaler_mean.npy", scaler.mean_)   # 形状: (3,)
np.save("scaler_std.npy", scaler.scale_)   # 形状: (3,)
print("☑ スケーラ統計を保存しました: scaler_mean.npy / scaler_std.npy")
# スライディングウィンドウ作成
X = []
for i in range(len(xyz_scaled) - window_size):
    window = xyz_scaled[i:i+window_size].flatten()
    X.append(window)
X = np.array(X)
# 学習用データを保存
np.save(output_file, X)
print(f"☑ {output_file} を保存しました(shape = {X.shape})")
```

リスト6　リスト5の実行結果↓

```
python .¥prepare_data.py
📥 読み込み中: data¥sensor_data_2508022026.csv
:
📥 読み込み中: data¥sensor_data_2508161414.csv
☑ スケーラ統計を保存しました: scaler_mean.npy / scaler_std.npy
☑ X_train.npy を保存しました(shape = (3500, 30))
```

リスト7　モデルの学習処理プログラム（train_autoencoder.py）↓

```
: (省略)
def main(in_fn: str, epochs: int, batch_size: int):
    # 前処理済みの学習データ(.npy)を読み込み
    print(f"📥 前処理済みデータを読み込み: {in_fn}")
    X = np.load(in_fn)
    if X.ndim != 2:
        raise ValueError(f"入力配列は2次元である必要がありますが、形状: {X.shape}")
    print(f"  shape = {X.shape}")
    # Autoencoder 構築
    input_layer = Input(shape=(X.shape[1],))
    encoded = Dense(16, activation="relu")(input_layer)
    decoded = Dense(X.shape[1], activation="linear")(encoded)
    autoencoder = Model(inputs=input_layer, outputs=decoded)
    autoencoder.compile(optimizer='adam', loss='mse')
    print("🎓 学習開始...")
    autoencoder.fit(X, X, epochs=epochs, batch_size=batch_size, shuffle=True)
    # Keras (.h5) 形式で保存
    print("💾 Keras (.h5) 形式で保存中...")
    autoencoder.save("autoencoder_model.h5")
    print("☑ モデルを autoencoder_model.h5 に保存しました。")
    # TFLite 形式に変換・保存
    print("💾 TFLite 形式に変換中...")
    converter = tf.lite.TFLiteConverter.from_keras_model(autoencoder)
    tflite_model = converter.convert()
    with open("autoencoder_model.tflite", "wb") as f:
        f.write(tflite_model)
    print("☑ モデルを autoencoder_model.tflite に保存しました。")
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="前処理済みデータ(.npy)からAutoencoderを学習")
    parser.add_argument("-in_fn", default="X_train.npy", help="入力npyファイル(既定: X_train.npy)")
    parser.add_argument("--epochs", type=int, default=50, help="学習エポック数(既定: 50)")
    parser.add_argument("--batch_size", type=int, default=32, help="バッチサイズ(既定: 32)")
    args = parser.parse_args()
    main(args.in_fn, args.epochs, args.batch_size)
```

リスト8　リスト7の実行結果↓

```
python .¥train_autoencoder.py
:
📥 前処理済みデータを読み込み: X_train.npy
  shape = (3500, 30)
:
🎓 学習開始...
Epoch 1/50
110/110 [==============================] - 0s 411us/step - loss: 0.5997
Epoch 2/50
110/110 [==============================] - 0s 376us/step - loss: 0.0914
```

```
: (省略)
Epoch 50/50
110/110 [==============================] - 0s 365us/step - loss: 0.0022
💾 Keras (.h5) 形式で保存中...
☑ モデルを autoencoder_model.h5 に保存しました。
💾 TFLite 形式に変換中...
:
☑ モデルを autoencoder_model.tflite に保存しました。
```

リスト9　しきい値の計算用プログラム（calculate_threshold.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 = "autoencoder_model.tflite"
THRESHOLD_FILE = "threshold.txt"
SCALER_MEAN_PATH = "scaler_mean.npy"
SCALER_STD_PATH = "scaler_std.npy"
PERCENTILE = 95
print("📦 X_train.npy 読み込み中...")
X_train = np.load(X_TRAIN_PATH)
# --- スケーラ読み込み ---
if os.path.exists(SCALER_MEAN_PATH) and os.path.exists(SCALER_STD_PATH):
    scaler_mean = np.load(SCALER_MEAN_PATH)
    scaler_std = np.load(SCALER_STD_PATH)
    print("☑ スケーラ読み込み成功")
else:
    print("✕ スケーラファイルが見つかりません(scaler_mean.npy / scaler_std.npy)")
    exit(1)
print("📦 TFLite モデル読み込み中...")
interpreter = tf.lite.Interpreter(model_path=MODEL_PATH)
interpreter.allocate_tensors()
input_index = interpreter.get_input_details()[0]['index']
output_index = interpreter.get_output_details()[0]['index']
print("🔎 MSE 計算中...")
errors = []
for i, x in enumerate(X_train):
    window = x.reshape((10, 3))
    normed = (window - scaler_mean) / (scaler_std + 1e-8)
    input_data = normed.flatten().reshape(1, -1).astype(np.float32)
    interpreter.set_tensor(input_index, input_data)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_index)
    mse = np.mean((input_data - output_data) ** 2)
    errors.append(mse)
threshold = np.percentile(errors, PERCENTILE)
print(f"☑ {PERCENTILE}パーセンタイルのしきい値: {threshold:.4f}")
with open(THRESHOLD_FILE, "w") as f:
    f.write(f"{threshold:.6f}")
print(f"💾 しきい値を {THRESHOLD_FILE} に保存しました。")
```

リスト10　異常検知を実行するプログラム（detect_anomaly.py）↓

```
import tflite_runtime.interpreter as tflite
import numpy as np
from read_sensor import ADXL345
from collections import deque
import time
import os
# --- 設定 ---
WINDOW_SIZE = 10
THRESHOLD_FILE = "threshold.txt"
MODEL_PATH = "autoencoder_model.tflite"
SCALER_MEAN_PATH = "scaler_mean.npy"
SCALER_STD_PATH = "scaler_std.npy"
# --- しきい値の読み込み ---
if os.path.exists(THRESHOLD_FILE):
    with open(THRESHOLD_FILE, "r") as f:
        THRESHOLD = float(f.read().strip())
: (省略)
print(f"☑ 使用するしきい値: {THRESHOLD:.4f}")
# --- スケーラの読み込み ---
if os.path.exists(SCALER_MEAN_PATH) and os.path.exists(SCALER_STD_PATH):
    scaler_mean = np.load(SCALER_MEAN_PATH)
    scaler_std = np.load(SCALER_STD_PATH)
    print(f"☑ スケーラ読み込み成功")
: (省略)
# --- TFLiteモデルの読み込み ---
print("📦 モデル読み込み中...")
interpreter = tflite.Interpreter(model_path=MODEL_PATH)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

```
# --- センサー初期化 ---
sensor = ADXL345()
queue = deque(maxlen=WINDOW_SIZE)
print("🚀 異常検知を開始します(Ctrl+C で停止)")
# --- メインループ ---
try:
    while True:
        x, y, z = sensor.read_axes()
        queue.append([x, y, z])
        if len(queue) == WINDOW_SIZE:
            data = np.array(queue, dtype=np.float32)
            normed = (data - scaler_mean) / (scaler_std + 1e-8)
            input_data = np.array([normed.flatten()], dtype=np.float32)
            interpreter.set_tensor(input_details[0]['index'], input_data)
            interpreter.invoke()
            output = interpreter.get_tensor(output_details[0]['index'])[0]
            mse = np.mean((input_data[0] - output) ** 2)
            result = "🚨 Anomaly" if mse > THRESHOLD else "☑ Normal"
            print(f"{result} | MSE = {mse:.4f}")
        time.sleep(0.1)
: (省略)
```

リスト11　リスト10の実行結果↓

```
(tflite-env) xxxxxx@raspberrypi:~/dev/anomaly_detection/raspi5 $ python detect_anomaly.py
☑ 使用するしきい値: 0.1958
☑ スケーラ読み込み成功
・ モデル読み込み中...
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
・ 異常検知を開始します(Ctrl+C で停止)
☑ Normal | MSE = 0.0003
:
☑ Normal | MSE = 0.0003
・ Anomaly | MSE = 0.8567
:
・ Anomaly | MSE = 3.0140
☑ Normal | MSE = 0.0004
☑ Normal | MSE = 0.0003
:
```

リスト12　しきい値の妥当性を視覚的に確認するプログラム（check_mse_distribution.py）↓

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

X = np.load("X_train.npy")
interpreter = tf.lite.Interpreter(model_path="autoencoder_model.tflite")
interpreter.allocate_tensors()
input_index = interpreter.get_input_details()[0]['index']
output_index = interpreter.get_output_details()[0]['index']

errors = []
for x in X:
    x = x.reshape(1, -1).astype(np.float32)
    interpreter.set_tensor(input_index, x)
    interpreter.invoke()
    output = interpreter.get_tensor(output_index)
    mse = np.mean((x - output) ** 2)
    errors.append(mse)
plt.hist(errors, bins=50)
plt.title("Reconstruction Error Histogram (X_train.npy)")
plt.xlabel("MSE")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

print("Min MSE:", np.min(errors))
print("Max MSE:", np.max(errors))
print("95 percentile:", np.percentile(errors, 95))
```