Chapter 3

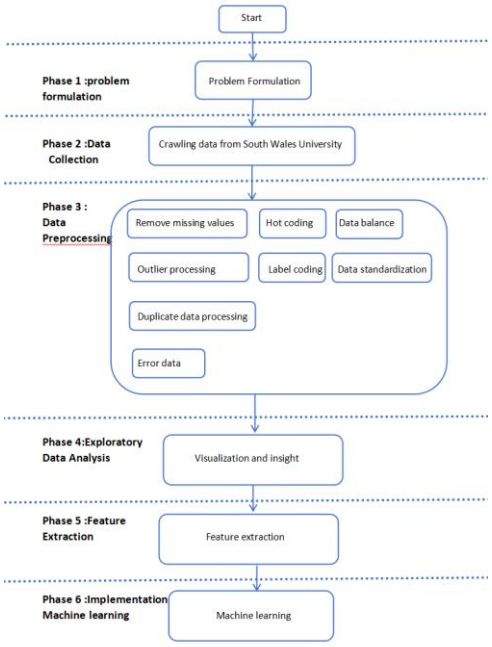
Research methodology

3.1 introduction

This chapter explains the research methods used to analyze traffic anomaly detection in IoT devices. The methods include data collection, data preprocessing, and machine modeling; various machine learning techniques are employed for classification and detection of device traffic anomalies. The study aims to enhance the security of IoT devices and protect user privacy, safety, and health by using machine learning to identify malicious attacks in traffic.

3.2 Research Framework：

1. Problem Definition and Literature Review.
2. Data collection: provided by South Wales University.
3. Data preprocessing: Clean and prepare data for further analysis
4. Feature extraction: word stem extraction and vectorization techniques are applied.
5. Model construction: Build and train the model.
6. Model evaluation: The performance of the model is analyzed through accuracy (accuracy), recall (recall) and precision (precision)



3.3 Problem Formulation：

The main purpose of this study is to use machine learning to identify malicious attacks in the traffic of IoT devices, so as to improve the security of IoT devices and protect users' privacy, safety and health. However, in order to ensure the accuracy and reliability of the analysis, the following problems need to be solved:

1、Pre-data input effects:
(1) Data quality issues (missing values, redundancy, non-numerical, irrelevant features)
(2) Class imbalance and noisy data.

2、Impact of data input:
(1) The data type of the input data machine learning does not match the data type that machine learning accepts.
(2) The imbalance of data volume in each category leads to low efficiency.

3、After data input:
(1) The false positive and false negative rates lead to errors in the experimental results

3.4 Data Collection

3.4.1data sources：

This dataset was created by researchers who set up a real IoT experimental environment, including several common smart home devices: Smart Doorbell, Thermostat, Camera, and others. These devices are connected to the home gateway via Wi-Fi or wired connections and are controlled to interact with normal and malicious traffic. The Mirai botnet tool is used to launch DDoS, scanning, and MITM attacks on these devices, simulating attack traffic and capturing network packets during communication. The UNSW BoT-IoT dataset was obtained through this experimental process.

3.4.2 Dataset comparison:

According to the comparison in the table, the UNSW BoT-IoT dataset was published in 2019, making it relatively new. The institution releasing this dataset, the University of New South Wales (UNSW), is highly reputable. This dataset provides a comprehensive CSV label file with 45 fields and has the largest scale, containing over

7.2 million traffic records. Notably, this dataset focuses on botnets in smart home scenarios and is one of the most frequently cited IoT security datasets compared to other datasets.Therefore, this data set is selected as the experimental object.

Dataset	Date of publication	Research and development units	Feature Class	Instances	For IoT scenarios	Up-to-date Relevance	Large-scale Attack Representation	Evaluated by Recent Studies
(UNSW)BoT-IoT	2019	University of New South Wales	Provides a complete CSV label file with 45 fields	The largest scale, with more than 72 million traffic records	Focus on botnets in the smart home scenario	It is relatively new and meets the needs of modern IoT security research	Provides complete botnet lifecycle modeling (scan → control → attack)	One of the most cited IoT security data sets today
N_BaIoT	2018	Ben-Gurion University of the Negev	Only the original pcap file is provided	The scale is moderate, mainly in the form of pcap	Based on real IoT device traffic	While newer, it focuses on the Mirai type botnet and may not reflect the latest attack trends	Focus only on the Mirai type botnet	It is widely used in IoT botnet research
TON_IoT	2020	CSIRO's Data61	Provides multiple CSV files covering a variety of attack types	2.28 million records, enough for ML/DL training	Covering IoT and IIoT (industrial Internet of Things)	It has high timeliness and supports a variety of modern attack types	Provides 23 kinds of real IoT malware behavior data, suitable for practical testing	It has been used in many papers in recent years
IoT-23	2020	ThreatMon (Cybersecurity Company)	Each sample corresponds to one type of malware (a total of 23)	Millions of samples, suitable for small and medium scale experiments	Emphasis on IoT malicious behavior in live deployment	Contains the latest malware variants and is time-sensitive	Including DDoS, Ransomware, Backdoor, Crypto Mining and many other attacks	It is adopted by security vendors and practical projects

3.5 Data Pre-Processing：

The preprocessing of the data set is to make the input machine learning data set not contain data that affects the experimental results, and the data set can be smoothly input into the machine learning model, and the machine learning model can run efficiently, so the data set needs to be pre-processed.

3.5.1 Preliminary Analysis

Before preprocessing, a preliminary analysis of the dataset is necessary. This initial step is crucial for data analysis as it helps to familiarize oneself with the dataset, including its structure, format, and the types of variables it contains. The preliminary investigation can identify issues that need correction to ensure the reliability of the analysis, such as missing values, outliers, or inconsistencies.

3.5.2 Data Pre-Processing

Data preprocessing includes data cleaning, data type conversion, data balancing and data standardization.These steps are designed to make machine learning models better and more efficient at processing data, resulting in more accurate experimental data.

1. Data cleaning:

(1) Remove missing values; For missing data, replace with NaN or empty string:

```python
# (1) Handle Missing Values: Replace NaN and empty strings
def handle_missing_values(df):
    print("Handling missing values...")
    # Replace empty strings, '?', and other placeholders with NaN
    df.replace(['', ' ', '?', -1, '-1', np.inf, -np.inf], np.nan, inplace=True)

    # Fill numeric columns with mean
    num_cols = df.select_dtypes(include=[np.number]).columns
    df[num_cols] = df[num_cols].fillna(df[num_cols].mean())

    # Drop non-numeric columns with missing values (or encode later if needed)
    non_num_cols = df.select_dtypes(exclude=[np.number]).columns
    df.drop(columns=non_num_cols, inplace=True)

    print("Missing values handled.")
    return df
```

(2) Outlier processing: the values obviously deviating from the normal range in the data are deleted;

```python
# (2) Remove Outliers using Z-score
def remove_outliers(df, z_threshold=3):
    print("Removing outliers...")
    numeric_df = df.select_dtypes(include=[np.number])
    z_scores = np.abs(stats.zscore(numeric_df))
    filtered_entries = (z_scores < z_threshold).all(axis=1)
    df_cleaned = df[filtered_entries]
    print(f"Outliers removed. New shape: {df_cleaned.shape}")
    return df_cleaned
```

(3) Duplicate data processing: delete the same data that has been processed multiple times;

```python
# (3) Remove Duplicate Rows
def remove_duplicates(df):
    print("Removing duplicate rows...")
    initial_shape = df.shape
    df.drop_duplicates(inplace=True)
    duplicates_removed = initial_shape[0] - df.shape[0]
    print(f"{duplicates_removed} duplicate(s) removed.")
    return df
```

(4) Error data: the data contains data that is not required for the search, and the data is deleted;

```python
# (4) Remove Invalid Data: Example filter for invalid protocol types
def remove_invalid_data(df, valid_protocols=None):
    print("Removing invalid data...")
    if valid_protocols is not None and 'proto' in df.columns:
        # Example: Only keep rows where 'proto' is one of the valid protocols
        df = df[df['proto'].isin(valid_protocols)]
        print(f"Invalid protocol entries removed. New shape: {df.shape}")
    return df
```

2. Data type conversion:

(1) The classification features are coded with unique heat, and each attack category has a unique column, thus increasing the number of features;

```python
# (1) One-Hot Encoding for categorical features
def one_hot_encode_features(df, categorical_cols=None):
    if categorical_cols is None or len(categorical_cols) == 0:
        print("No categorical columns provided for one-hot encoding.")
        return df

    print(f"Performing one-hot encoding on columns: {categorical_cols}")
    df_encoded = pd.get_dummies(df, columns=categorical_cols)
    print(f"One-hot encoding completed. New shape: {df_encoded.shape}")
    return df_encoded
```

(2) The data is converted into feature vectors using label coding techniques, which can be converted into a format that machine learning models can receive;

```python
# (2) Label Encoding for target labels (attack types)
def label_encode_target(df, target_col='attack'):
    if target_col not in df.columns:
        raise ValueError(f"Target column '{target_col}' not found in the dataset.")

    print(f"Performing label encoding on column: '{target_col}'")
    le = LabelEncoder()
    df[target_col] = le.fit_transform(df[target_col])

    # Print mapping of original labels to encoded values
    label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print("Encoded label mapping:")
    for cls, idx in label_mapping.items():
        print(f"  {cls}: {idx}")

    print("Label encoding completed.")
    return df, label_mapping
```

3. Data balance:Using smote over-sampling technology to balance the number of data sets in each category, so as to improve the efficiency of machine learning

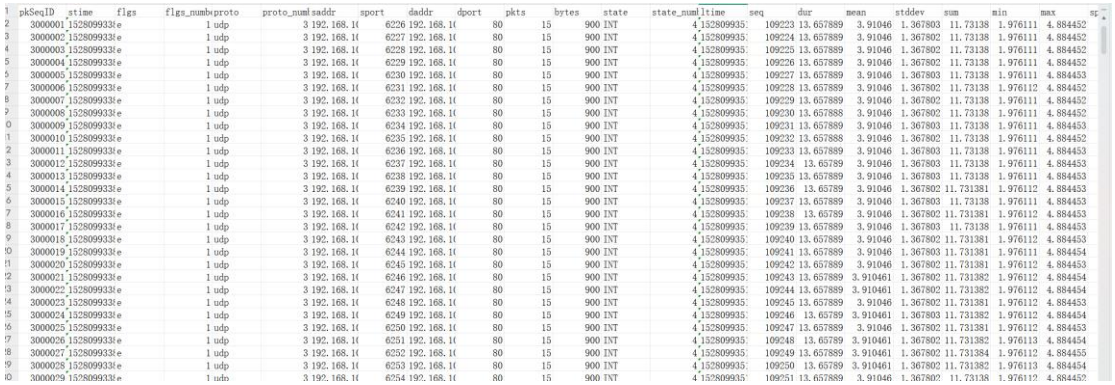
models processing data;

```python
# Apply SMOTE over-sampling
def apply_smote(X, y):
    print("Applying SMOTE to balance class distribution...")

    # Print original class distribution
    print("Original class distribution:", dict(Counter(y)))

    # Initialize SMOTE
    smote = SMOTE(random_state=42)

    # Apply SMOTE
    X_resampled, y_resampled = smote.fit_resample(X, y)

    # Print new class distribution
    print("Resampled class distribution:", dict(Counter(y_resampled)))
    print("✅ SMOTE applied successfully.")
    return X_resampled, y_resampled
```

4. Data standardization/normalization:Standardized data can help the model better capture the relationship between features and avoid prediction instability caused by numerical problems.

```python
# Load encoded/balanced dataset
def load_data(file_path):
    print(f"Loading data: {file_path}")
    df = pd.read_csv(file_path)
    print(f"Loaded shape: {df.shape}")
    return df

# Standardize numerical features
def standardize_data(df, feature_cols, target_col='attack'):
    print("Starting data standardization...")

    # Separate features and labels
    X = df[feature_cols]
    y = df[[target_col]]

    # Initialize StandardScaler
    scaler = StandardScaler()

    # Fit and transform the features
    X_scaled = scaler.fit_transform(X)

    # Convert back to DataFrame
    X_scaled_df = pd.DataFrame(X_scaled, columns=feature_cols, index=df.index)

    # Combine with labels
    df_scaled = pd.concat([X_scaled_df, y.reset_index(drop=True)], axis=1)

    print("✅ Standardization complete.")
    return df_scaled
```

The processed data set:



pkSeqID	stime	flgs	flgs_numb	proto	proto_num	saddr	sport	daddr	dport	pkts	bytes	state	state_num	ltime	seq	dur	mean	stddev	sum	min	max	sr
3000001	1528099338	e	1	udp	3	192.168.1(	6226	192.168.1(	80	15	900	INT	4	1528099935	109223	13.657889	3.91046	1.367803	11.73138	1.976111	4.884452	
3000002	1528099338	e	1	udp	3	192.168.1(	6227	192.168.1(	80	15	900	INT	4	1528099935	109224	13.657889	3.91046	1.367802	11.73138	1.976111	4.884452	
3000003	1528099338	e	1	udp	3	192.168.1(	6228	192.168.1(	80	15	900	INT	4	1528099935	109225	13.657889	3.91046	1.367802	11.73138	1.976111	4.884452	
3000004	1528099338	e	1	udp	3	192.168.1(	6229	192.168.1(	80	15	900	INT	4	1528099935	109226	13.657889	3.91046	1.367802	11.73138	1.976111	4.884452	
3000005	1528099338	e	1	udp	3	192.168.1(	6230	192.168.1(	80	15	900	INT	4	1528099935	109227	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000006	1528099338	e	1	udp	3	192.168.1(	6231	192.168.1(	80	15	900	INT	4	1528099935	109228	13.657889	3.91046	1.367802	11.73138	1.976112	4.884452	
3000007	1528099338	e	1	udp	3	192.168.1(	6232	192.168.1(	80	15	900	INT	4	1528099935	109229	13.657889	3.91046	1.367802	11.73138	1.976111	4.884452	
3000008	1528099338	e	1	udp	3	192.168.1(	6233	192.168.1(	80	15	900	INT	4	1528099935	109230	13.657888	3.91046	1.367802	11.73138	1.976111	4.884452	
3000009	1528099338	e	1	udp	3	192.168.1(	6234	192.168.1(	80	15	900	INT	4	1528099935	109231	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000010	1528099338	e	1	udp	3	192.168.1(	6235	192.168.1(	80	15	900	INT	4	1528099935	109232	13.657888	3.91046	1.367802	11.73138	1.976111	4.884452	
3000011	1528099338	e	1	udp	3	192.168.1(	6236	192.168.1(	80	15	900	INT	4	1528099935	109233	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000012	1528099338	e	1	udp	3	192.168.1(	6237	192.168.1(	80	15	900	INT	4	1528099935	109234	13.65789	3.91046	1.367803	11.73138	1.976111	4.884453	
3000013	1528099338	e	1	udp	3	192.168.1(	6238	192.168.1(	80	15	900	INT	4	1528099935	109235	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000014	1528099338	e	1	udp	3	192.168.1(	6239	192.168.1(	80	15	900	INT	4	1528099935	109236	13.65789	3.91046	1.367802	11.73138	1.976112	4.884453	
3000015	1528099338	e	1	udp	3	192.168.1(	6240	192.168.1(	80	15	900	INT	4	1528099935	109237	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000016	1528099338	e	1	udp	3	192.168.1(	6241	192.168.1(	80	15	900	INT	4	1528099935	109238	13.65789	3.91046	1.367802	11.73138	1.976112	4.884453	
3000017	1528099338	e	1	udp	3	192.168.1(	6242	192.168.1(	80	15	900	INT	4	1528099935	109239	13.657889	3.91046	1.367803	11.73138	1.976111	4.884453	
3000018	1528099338	e	1	udp	3	192.168.1(	6243	192.168.1(	80	15	900	INT	4	1528099935	109240	13.657889	3.91046	1.367802	11.73138	1.976112	4.884453	
3000019	1528099338	e	1	udp	3	192.168.1(	6244	192.168.1(	80	15	900	INT	4	1528099935	109241	13.657889	3.91046	1.367803	11.73138	1.976111	4.884454	
3000020	1528099338	e	1	udp	3	192.168.1(	6245	192.168.1(	80	15	900	INT	4	1528099935	109242	13.657889	3.91046	1.367802	11.73138	1.976112	4.884453	
3000021	1528099338	e	1	udp	3	192.168.1(	6246	192.168.1(	80	15	900	INT	4	1528099935	109243	13.657889	3.910461	1.367802	11.73138	1.976112	4.884454	
3000022	1528099338	e	1	udp	3	192.168.1(	6247	192.168.1(	80	15	900	INT	4	1528099935	109244	13.657889	3.910461	1.367802	11.73138	1.976112	4.884454	
3000023	1528099338	e	1	udp	3	192.168.1(	6248	192.168.1(	80	15	900	INT	4	1528099935	109245	13.657889	3.91046	1.367802	11.73138	1.976112	4.884453	
3000024	1528099338	e	1	udp	3	192.168.1(	6249	192.168.1(	80	15	900	INT	4	1528099935	109246	13.65789	3.910461	1.367803	11.73138	1.976112	4.884454	
3000025	1528099338	e	1	udp	3	192.168.1(	6250	192.168.1(	80	15	900	INT	4	1528099935	109247	13.657889	3.91046	1.367802	11.73138	1.976112	4.884453	
3000026	1528099338	e	1	udp	3	192.168.1(	6251	192.168.1(	80	15	900	INT	4	1528099935	109248	13.65789	3.910461	1.367802	11.73138	1.976113	4.884454	
3000027	1528099338	e	1	udp	3	192.168.1(	6252	192.168.1(	80	15	900	INT	4	1528099935	109249	13.657889	3.910461	1.367802	11.73138	1.976112	4.884455	
3000028	1528099338	e	1	udp	3	192.168.1(	6253	192.168.1(	80	15	900	INT	4	1528099935	109250	13.65789	3.910461	1.367802	11.73138	1.976113	4.884454	
3000029	1528099338	e	1	udp	3	192.168.1(	6254	192.168.1(	80	15	900	INT	4	1528099935	109251	13.657889	3.91046	1.367802	11.73138	1.976112	4.884452	

3.6 Feature Extraction

In order to improve the efficiency of machine learning model processing, only features with high correlation are selected in the feature selection process. The following features will be extracted. Network traffic features:

1. These features include data related to network protocols, packet sizes, flow duration, and source/destination IP addresses. By analyzing patterns and changes in network traffic, the model can identify abnormal behaviors that may indicate potential attacks or malicious activities. For example, a sudden increase in packet size or unexpected protocol usage could be signs of anomalies.

2. Communication mode: The characteristics related to the communication mode involve the number of connections, data transmission rate, and interaction frequency between devices. Abnormalities in these modes, such as abnormal connections initiated by devices, may indicate a potential security threat.

3.7 Machine Learning Models:

In the experiment, a variety of machine learning methods were considered, including Causal Forest + SHAP, Isolation Forest + Autoencoder, Lightweight GNN + Autoencoder, and Random Forest.

The final stage of accurately identifying malicious activities in the network involves applying and classifying the UNSW BoT IoT dataset into machine learning models. The models to be used include Causal Forest + SHAP, Isolation Forest + Autoencoder, Lightweight GNN + Autoencoder, and Random Forest. The four machine learning models for detecting anomalies in IoT network traffic are:

1. Causal Forest + SHAP causal analysis script scheme: It is a machine learning research scheme that combines causal inference (Causal Inference) and explainability (Explainability); it provides visual interpretation to show which features have the biggest impact on causal effects.

```python
from econml.grf import CausalForest

# Initialize and train Causal Forest
cf = CausalForest(n_estimators=50, max_depth=5)
cf.fit(X=X_scaled, Y=outcome.values, T=treatment.values)

# Predict treatment effects
treatment_effects = cf.effect(X_scaled)

import shap

# Create SHAP explainer
explainer = shap.TreeExplainer(cf)
shap_values = explainer.shap_values(X_scaled)

# Summary plot: Global feature importance
shap.summary_plot(shap_values, X_scaled, feature_names=X.columns)

# Dependence plot: How 'Dst Port' affects treatment effect
shap.dependence_plot("Dst Port", shap_values, X_scaled, feature_names=X.columns)

# Force plot: Local explanation for individual samples
shap.force_plot(explainer.expected_value, shap_values[0,:], X_scaled[0,:], feature_names=X.columns)
```

2. Isolation Forest + Autoencoder Joint Detection Scheme: Isolation Forest + Autoencoder Joint Detection Scheme is an unsupervised/half-supervised anomaly detection framework, which uses the complementary advantages of the two models in different dimensions to improve the identification ability of unknown attacks in the Internet of Things environment.

```python
from sklearn.ensemble import IsolationForest

iso_forest = IsolationForest(n_estimators=100, contamination=0.1, behaviour='new')
iso_forest.fit(X_scaled)

iso_pred = iso_forest.predict(X_scaled)
iso_binary = [1 if x == -1 else 0 for x in iso_pred]  # Convert to 0/1 labels

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

input_dim = X_scaled.shape[1]
encoding_dim = 32

input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation="relu")(input_layer)
decoder = Dense(input_dim, activation="linear")(encoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mse')

# Train only on normal traffic
autoencoder.fit(X_scaled[y == "Normal"], X_scaled[y == "Normal"], epochs=50, batch_size=128, shuffle=True, verbose=0)

# Reconstruct all samples
reconstructions = autoencoder.predict(X_scaled)
mse = np.mean(np.power(X_scaled - reconstructions, 2), axis=1)
threshold = np.percentile(mse, 95)  # Set threshold at 95th percentile
ae_binary = (mse > threshold).astype(int)  # 1 = Anomaly, 0 = Normal
```

3. Lightweight GNN + Autoencoder joint detection scheme: It is a hybrid model based on graph structure modeling and unsupervised feature reconstruction. This scheme combines graph structure analysis and feature reconstruction error to improve the overall detection accuracy.

```python
import torch.nn as nn
from torch_geometric.nn import GCNConv

class LightGNN(nn.Module):
    def __init__(self, input_dim, hidden_dim=32):
        super(LightGNN, self).__init__()
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, 2)  # Output embedding or attack probability

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        return x

model = LightGNN(input_dim=X_scaled.shape[1])
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()

# Training loop (simplified)
for epoch in range(50):
    model.train()
    optimizer.zero_grad()
    out = model(data)
    loss = criterion(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
```

The experimental models were evaluated one by one based on indicators such as accuracy, precision, recall and F1 score to determine the best performing model. The model results were evaluated using the following indicators:

① Accuracy: percentage of correct predictions.
② Precision: the accuracy of forward prediction.
③ Recall rate: the ability of a model to detect all positive data.
④ F1 score: harmonic mean of accuracy and recall.

3.8 Summary

This chapter introduces the research process from data collection to machine learning model processing in detail, and shows the implementation of specific research methods such as data collection and preprocessing, and machine modeling.