Network Traffic Classification Using Machine Learning and Deep Learning Approaches

Liu, Yucheng

LEHIGH UNIVERSITY

Research Background

Project Objectives

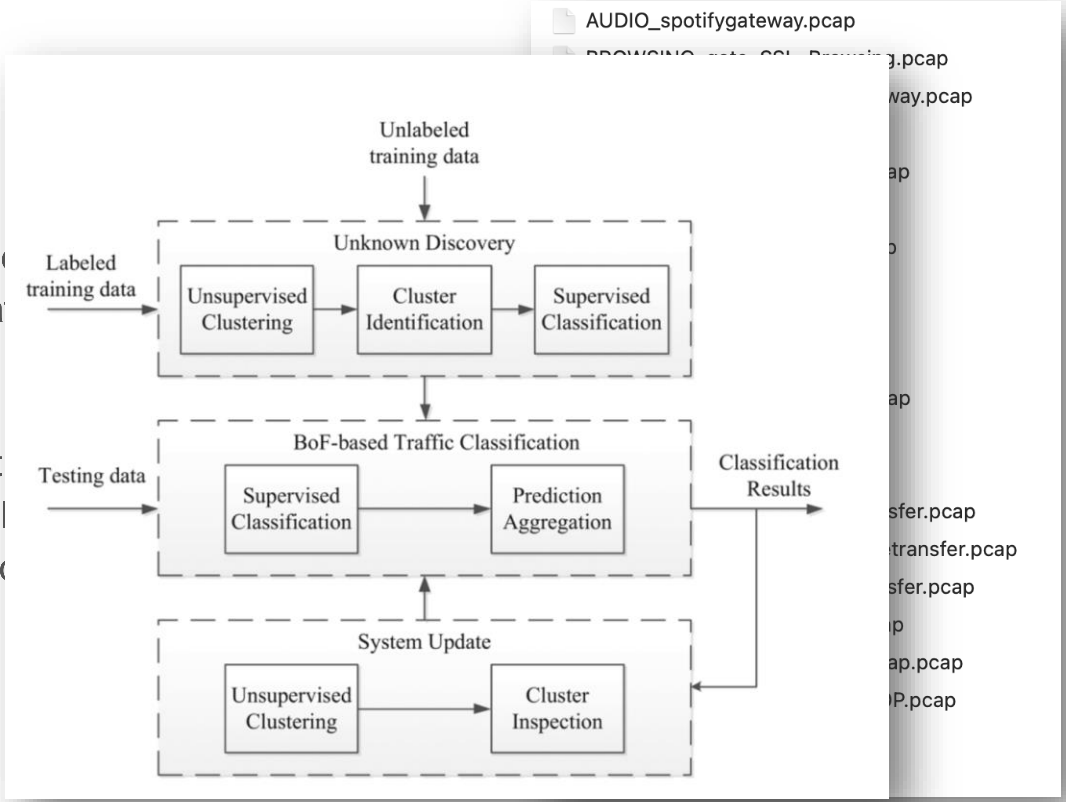
- **Flow classification goal**: map raw encrypted traffic → application labels (e.g. Chat vs. Video vs. P2P)
- **End-to-end pipeline**: PCAP ⟶ feature extraction ⟶ ML/DL model ⟶ real-time inference
- **Key questions**:
- Can lightweight ML (K-NN, RF) match deep models?
- Does temporal modeling (CNN+RNN) improve accuracy?
- What is the training time vs. accuracy trade-off?

AUDIO_spotifygateway.pcap
BROWSING_gate_SSL_Browsing.pcap
BROWSING_ssl_browsing_gateway.pcap
CHAT_aimchatgateway.pcap
CHAT_facebookchatgateway.pcap
CHAT_gate_AIM_chat.pcap
CHAT_gate_facebook_chat.pcap
CHAT_gate_hangout_chat.pcap
CHAT_gate_ICQ_chat.pcap
CHAT_gate_skype_chat.pcap
CHAT_hangoutschatgateway.pcap
CHAT_icqchatgateway.pcap
CHAT_skypechatgateway.pcap
FILE-TRANSFER_gate_FTP_transfer.pcap
FILE-TRANSFER_gate_SFTP_filetransfer.pcap
MAIL_gate_Email_IMAP_filetransfer.pcap
MAIL_gate_POP_filetransfer.pcap
MAIL_Gateway_Thunderbird_Imap.pcap
MAIL_Gateway_Thunderbird_POP.pcap
torFacebook.pcap
torGoogle.pcap

LEHIGH UNIVERSITY

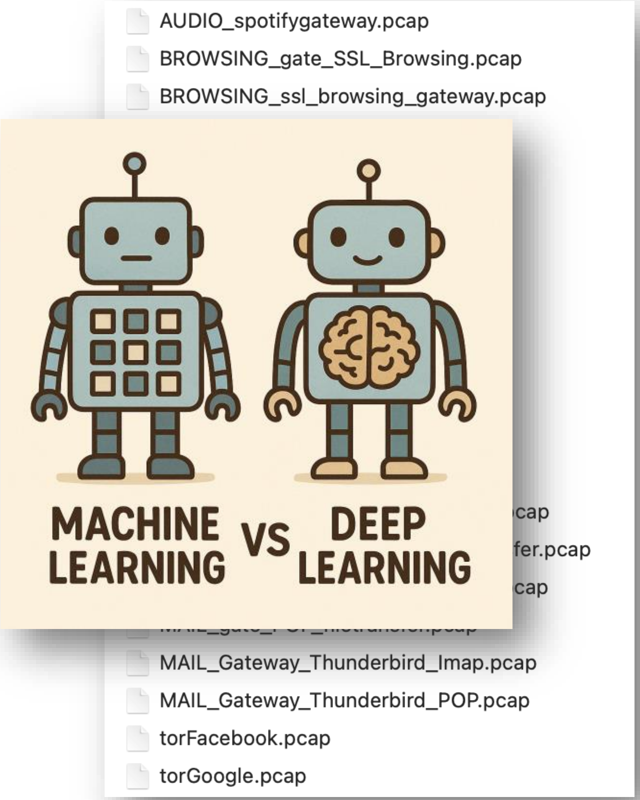
Project Objectives

- **Flow classification goal**: map raw
  application labels (e.g. Chat vs. Vid
- **End-to-end pipeline**: PCAP ⟶ fea
  model ⟶ real-time inference
- **Key questions**:
- Can lightweight ML (K-NN, RF) mat
- Does temporal modeling (CNN+RN
- What is the training time vs. accura



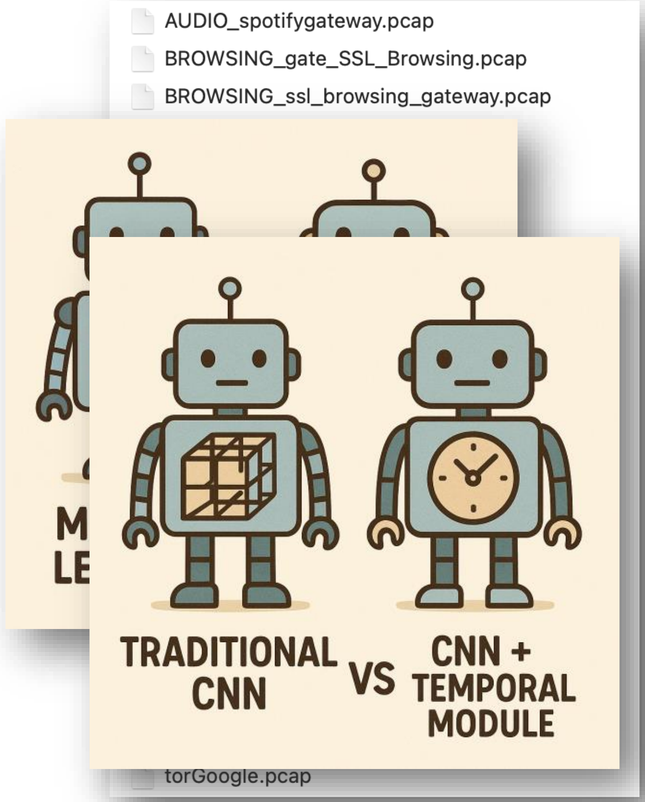
Project Objectives

- **Flow classification goal**: map raw encrypted traffic → application labels (e.g. Chat vs. Video vs. P2P)
- **End-to-end pipeline**: PCAP ⟶ feature extraction ⟶ ML/DL model ⟶ real-time inference
- **Key questions**:
- Can lightweight ML (K-NN, RF) match deep models?
- Does temporal modeling (CNN+RNN) improve accuracy?
- What is the training time vs. accuracy trade-off?



Project Objectives

- **Flow classification goal**: map raw encrypted traffic → application labels (e.g. Chat vs. Video vs. P2P)
- **End-to-end pipeline**: PCAP ⟶ feature extraction ⟶ ML/DL model ⟶ real-time inference
- **Key questions**:
- Can lightweight ML (K-NN, RF) match deep models?
- Does temporal modeling (CNN+RNN) improve accuracy?
- What is the training time vs. accuracy trade-off?



Key Challenges

- **Raw PCAP complexity:** packet parsing, flow reassembly, noise removal
- **Feature selection:** hundreds of statistical metrics; choosing meaningful subset
- **Capturing temporal patterns:**per-flow aggregates lose intra-flow dynamics / need sub-window sequence representation

LEHIGH UNIVERSITY

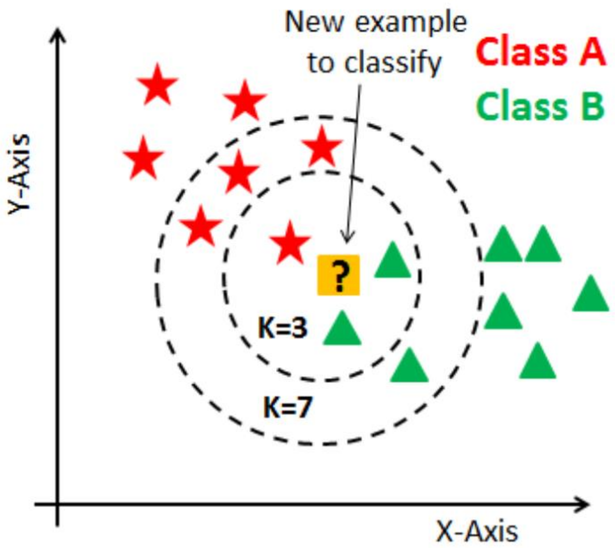
Key Challenges

- **Raw PCAP complexity:** packet parsing, flow reassembly, noise removal
- **Feature selection:** hundreds of statistical metrics; choosing meaningful subset
- **Capturing temporal patterns:**per-flow aggregates lose intra-flow

```
Source IP, Source Port, Destination IP, Destination Port, Protocol, Flow Duration,
Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min,
n,Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min,Bwd IAT Mean, Bwd IAT Std,
Bwd IAT Max, Bwd IAT Min,Active Mean, Active Std, Active Max, Active Min,Idle Mean,
, Idle Std, Idle Max, Idle Min,label
```

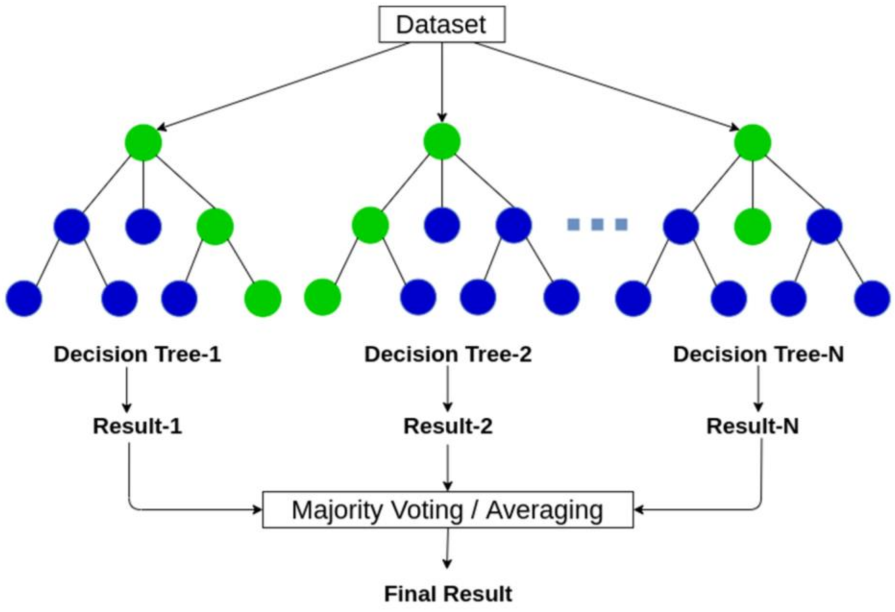
Review – Traditional ML

- **K-Nearest Neighbors (K-NN)**
  - simple, non-parametric; labels by nearest feature vectors
  - *Pros*: no training phase, interpretable
  - *Cons*: slow at inference, sensitive to feature scaling



Review – Traditional ML

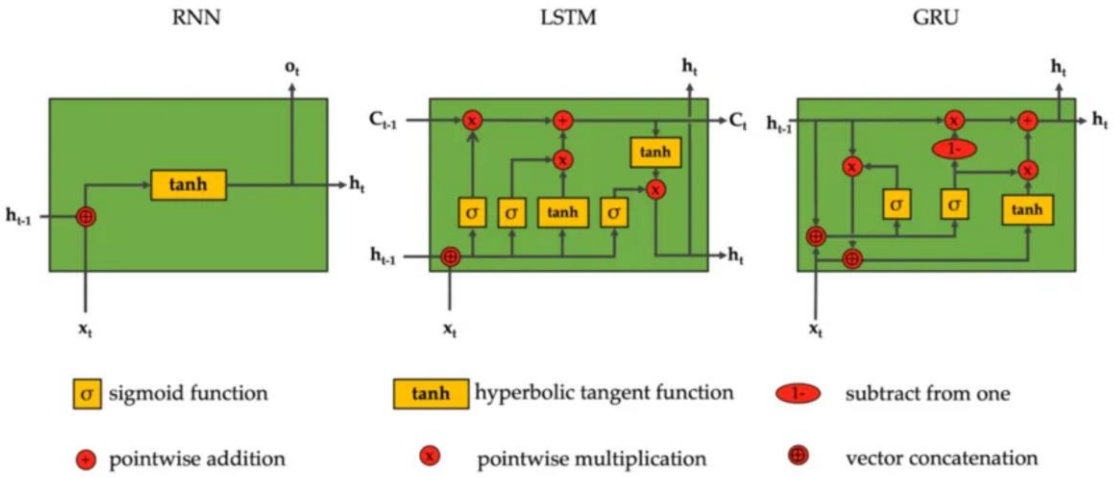
- ## Random Forest (RF)
    - Ensemble of decision trees trained on random subsets of samples & features
    - **Pros**:Robust to noise and overfitting, Handles high-dimensional data well
    - **Cons**:Large model size, Longer training time as number of trees grows



LEHIGH UNIVERSITY

Review – Deep Learning Method

- **RNN family (LSTM / GRU)**
  - model long/short-term temporal dependencies across windows
  - handle variable-length flows, resistant to jitter

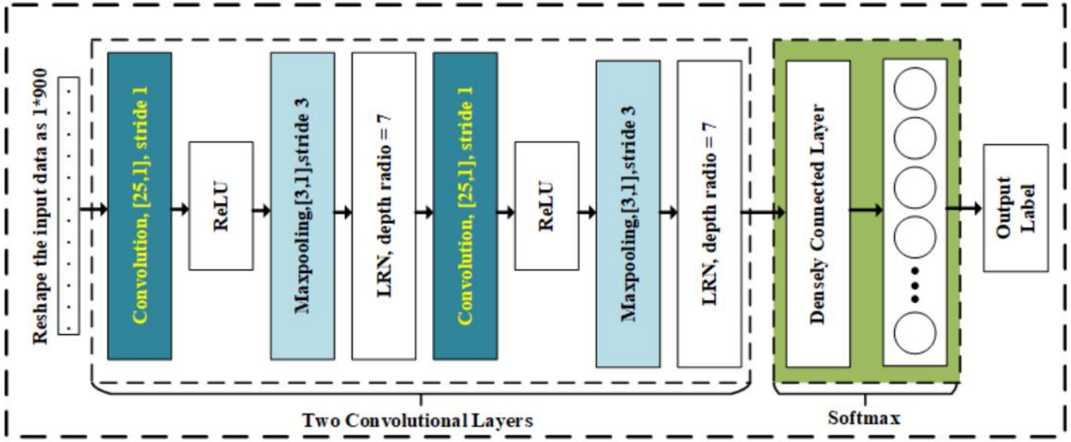


LEHIGH UNIVERSITY

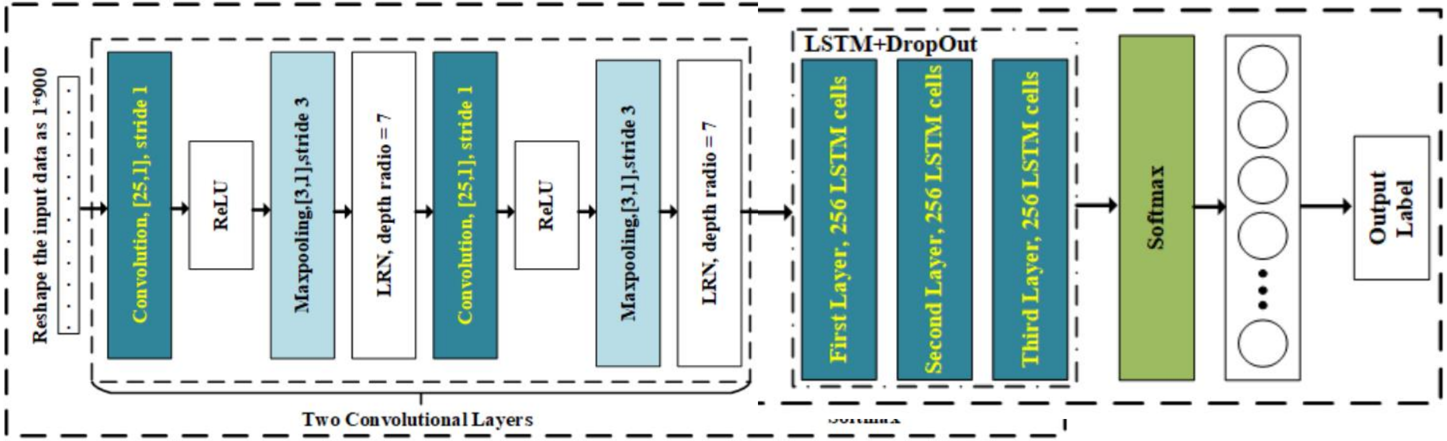
Review – Deep Learning Method

- **Hybrid architectures**
  - **CNN + LSTM/GRU** pipeline (Zeng et al., IEEE Access 2019 “Deep-Full-Range”)
  - integrate spatial & temporal modules for best of both worlds

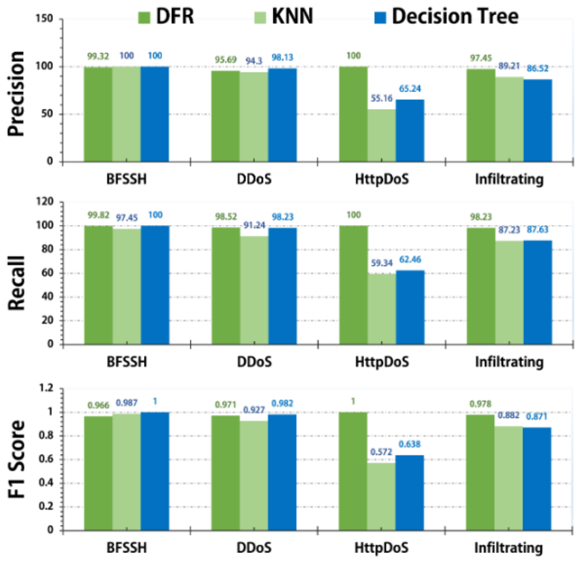
Review – Deep Learning Method



Review – Deep Learning Method



Review – Deep Learning Method



Deep-Full-Range: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework

Yi Zeng; Huaxi Gu

Algorithm 5 Online-Fashioned DFR Algorithm

Input:
$RT^1, RT^2, \ldots, RT^t, \ldots, RT^T$

Output:
The DFR model that best fits with the current traffic environment

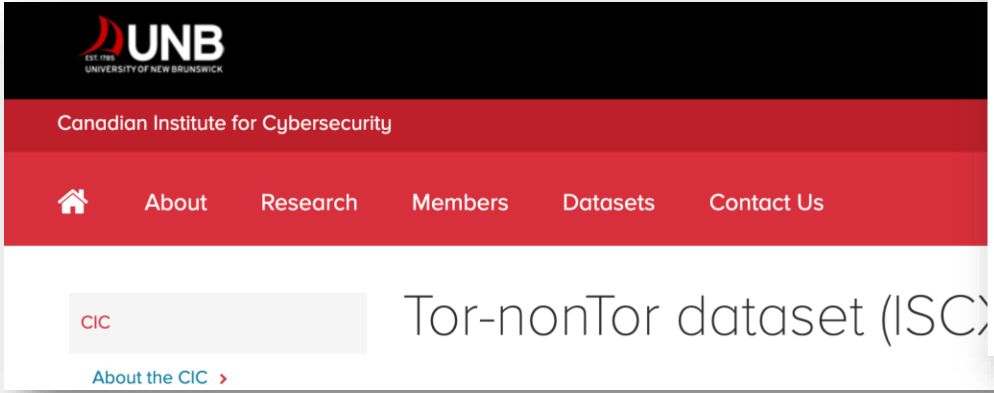
```
for each t in (1, T) do
    Apply Alg. 1 to gain G(1, 2, ...j, ...J)
    Randomly separate G(1, 2, ...j, ...J) according to 9:1;
    Use G_Train to train the three DFRs;
    Use G_Test to select the highest-accuracy DFR;
    Run the current-using DFR on the same G_Test;
    if the current-using DFR's accuracy is smaller then
        Save the new DFR;
        Transmit the new DFR to other units
    end if
end for
```

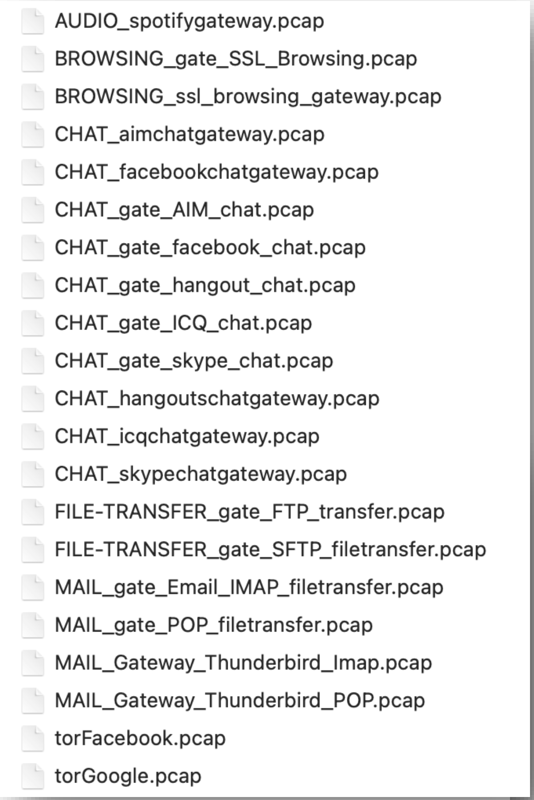
Preparation

./pcap_analysis.py

Data Processing & Feature Extraction

- **PCAP File Collection**
  - Collect network traffic data in PCAP format
  - Extract traffic labels from filenames





Data Processing

- **Group packets by 5-t**
  (*src_ip, src_port, dst_i*
  protocol)
  - Split flows into 10-seco
    for consistent analysis
  - Separate forward and backward traffic directions

```
for packet in packets:
    if IP in packet:
        if TCP in packet:
            proto = 'TCP'
            src_port = packet[TCP].sport
            dst_port = packet[TCP].dport
```

```
# Separate forward and backward packets
fwd_packets = [p for p in packets if p[IP].src == src_ip]
bwd_packets = [p for p in packets if p[IP].src == dst_ip]

# Calculate forward IAT statistics
fwd_times = [p.time for p in fwd_packets]
fwd_iats = [fwd_times[i+1] - fwd_times[i] for i in range(len(fwd_times)-1)]
```

```
        if src_ip < dst_ip:
            flow_key = (src_ip, src_port, dst_ip, dst_port, proto)
        else:
            flow_key = (dst_ip, dst_port, src_ip, src_port, proto)

        flows[flow_key].append(packet)

return flows
```

Feature Extraction – Basic Features

- Flow Duration, Flow Bytes
- IAT Statistics (Mean, Std,
- Overall Flow
- Forward Direction
- Backward Direction
- Active/Idle Time Statistics

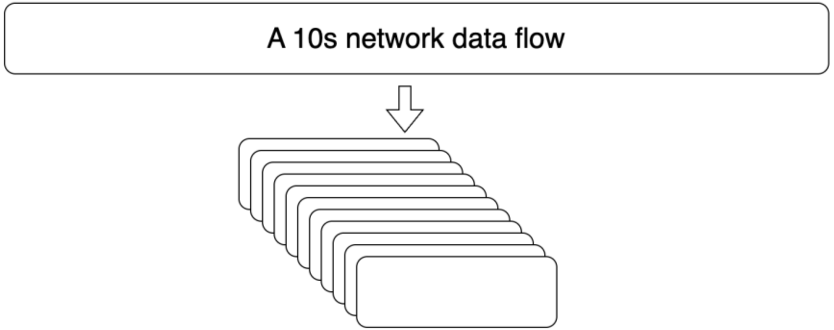
```
# Calculate flow bytes and packets per second
total_bytes = sum(len(p) for p in packets)
tot
byt
pac
```

```
# Calculate packet inter-arrival times (IAT)
packet_times = [p.time for p in packets]
iats = [packet_times[i+1] - packet_times[i] for i in range(len(packet_times)-1)]

# Ca
if i
```

```
for i in range(len(iats)):
    if iats[i] > idle_threshold:
        # End of active period
        active_time = packet_times[i] - current_active_start
        active_times.append(active_time)
        idle_times.append(iats[i])
        current_active_start = packet_times[i+1]
```

Feature Extraction - Temporal Features

- **Advanced Temporal Features (80 features)**
- 10 time windows × 8 features per window:
- Packets_Count, Bytes_Count
- IAT_Mean, IAT_Std
- Fwd_IAT_Mean, Fwd_IAT_Std
- Bwd_IAT_Mean, Bwd_IAT_Std



Data Processing

- **Preprocessing Steps:**
- Remove IP addresses and ports
- Handle missing values
- Replace infinite values
- Standardize features (zero mean, unit variance)
- Split data (80% training, 20% testing)

My Model Architecture

./base_model.py
./advanced_model.py

KNN

- **Hyperparameter Optimization**: GridSearchCV with 5-fold cross-validation
- **K values tested**: 3, 5, 7, 9, 11, 13, 15
- **Distance Metric**: Euclidean (default)
- **Weight Function**: Uniform weighting (default)

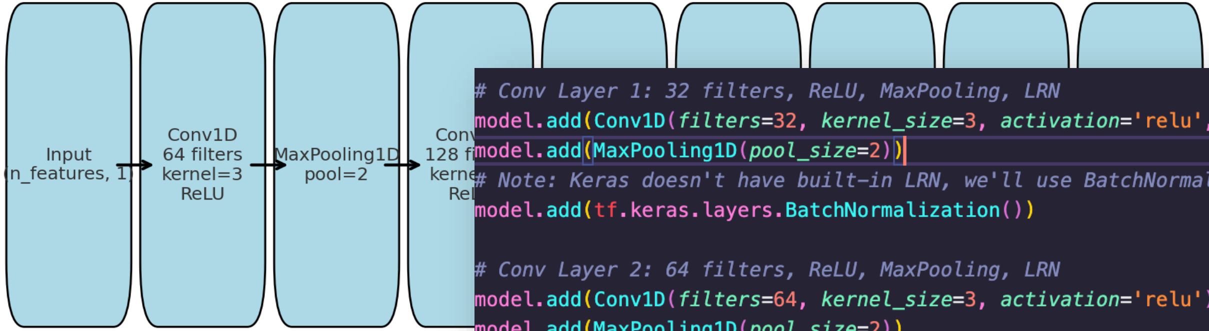
```python
def train_knn(X_train, y_train, X_test, y_test, feature_names):
    """Train and evaluate KNN model"""
    print("Training KNN model...")

    # Use grid search to find the best K value
    param_grid = {'n_neighbors': [3, 5, 7, 9, 11, 13, 15]}
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Get the best model
    best_knn = grid_search.best_estimator_
    best_k = grid_search.best_params_['n_neighbors']
    print(f"Best K value: {best_k}")
```

Random Forest

- Trees in Forest: 10, 50, or 100 (optimized via GridSearchCV)
- Max Tree Depth: None , 10, or 20
- Min Samples to Split: 2, 5, or 10
- Criterion: Gini impurity (default)
- Bootstrap: True (default)

```python
def train_random_forest(X_train, y_train, X_test, y_test, feature_names):
    """Train and evaluate Random Forest model"""
    print("Training Random Forest model...")

    # Define parameter grid for grid search
    param_grid = {
        'n_estimators': [10, 50, 100],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10]
    }

    # Create Random Forest classifier
    rf = RandomForestClassifier(random_state=42)

    # Perform grid search with cross-validation
    grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)

    # Get best model
    best_rf = grid_search.best_estimator_
    best_params = grid_search.best_params_
    print(f"Best parameters: {best_params}")
```

1D-CNN




```
# Conv Layer 1: 32 filters, ReLU, MaxPooling, LRN
model.add(Conv1D(filters=32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
# Note: Keras doesn't have built-in LRN, we'll use BatchNormalization as an alternative
model.add(tf.keras.layers.BatchNormalization())

# Conv Layer 2: 64 filters, ReLU, MaxPooling, LRN
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(tf.keras.layers.BatchNormalization())

# Flatten and Dense layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Adding dropout for regularization
model.add(Dense(num_classes, activation='softmax'))
```

CNN + LSTM

CNN Layers:
Conv1D: 64 filters, kernel size 3, ReLU activation, same padding
MaxPooling1D: pool size 2
Conv1D: 128 filters, kernel size 3, ReLU activation, same padding
MaxPooling1D: pool size 2

RNN Layers:
Bidirectional LSTM: 128 units, return sequences=True
Dropout: 30%
LSTM: 64 units
Dropout: 30%

CNN + GRU (simplified version LSTM)

CNN Layers:
Conv1D: 64 filters, kernel size 3, ReLU activation, same padding
MaxPooling1D: pool size 2
Conv1D: 128 filters, kernel size 3, ReLU activation, same padding
MaxPooling1D: pool size 2

RNN Layers:
Bidirectional LSTM: 128 units, return sequences=True
Dropout: 30%
GRU: 64 units
Dropout: 30%

LSTM vs GRU

Internal Architecture

LSTM: Has 3 gates (input, forget, output)

GRU: Has 2 gates (update, reset)

Parameter Count

LSTM: 4 weight matrices (more parameters)

GRU: 3 weight matrices (fewer parameters)

CNN-LSTM Model:

```
# LSTM part
x = Bidirectional(LSTM(128, return_sequences=True))(x)
x = Dropout(0.3)(x)
x = LSTM(64)(x)
x = Dropout(0.3)(x)
```

CNN-GRU Model:

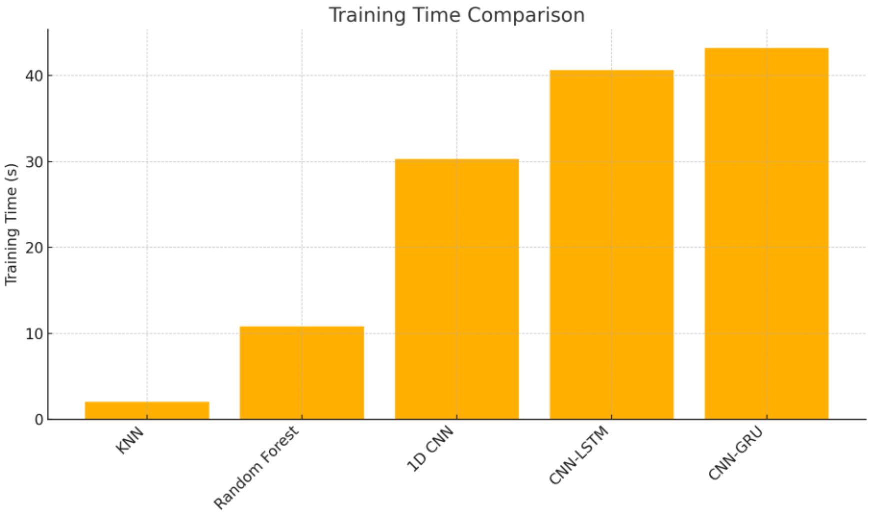
```
# GRU part
x = Bidirectional(GRU(128, return_sequences=True))(x)
x = Dropout(0.3)(x)
x = GRU(64)(x)
x = Dropout(0.3)(x)
```

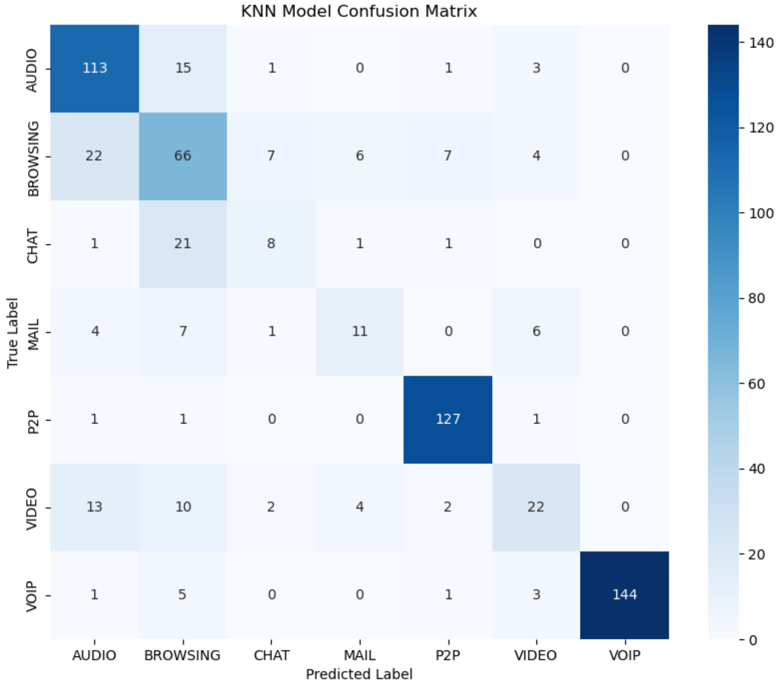
Result Analysis

LEHIGH UNIVERSITY

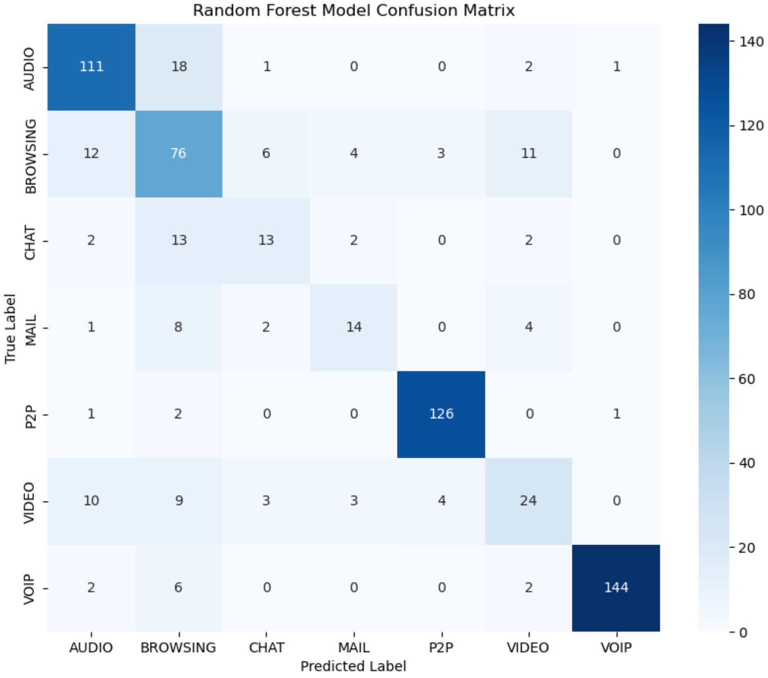
Accuracy

Time Spent



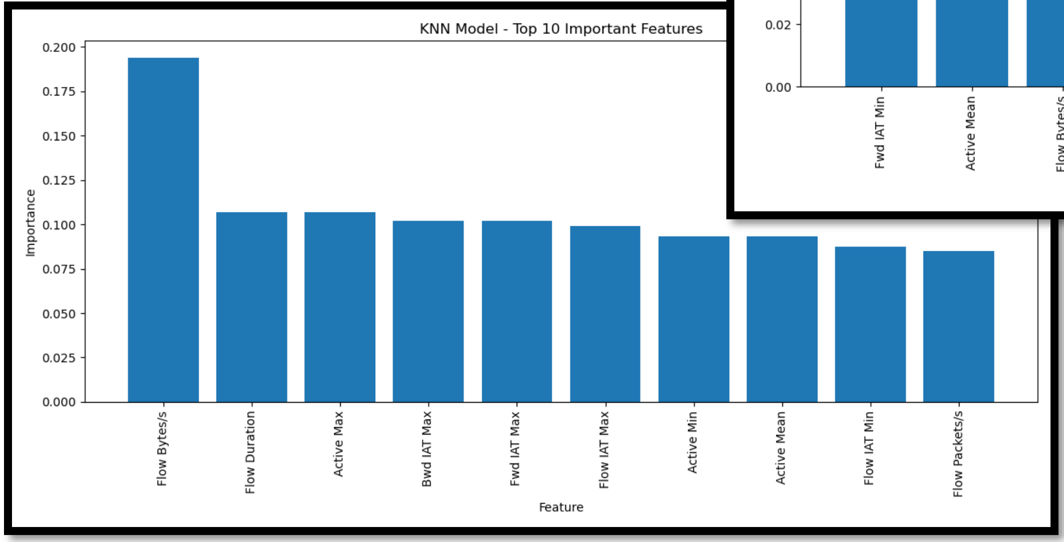

KNN Model Confusion Matrix
True Label
Predicted Label
AUDIO 113 15 1 0 1 3 0
BROWSING 22 66 7 6 7 4 0
CHAT 1 21 8 1 1 0 0
MAIL 4 7 1 11 0 6 0
P2P 1 1 0 0 127 1 0
VIDEO 13 10 2 4 2 22 0
VOIP 1 5 0 0 1 3 144
AUDIO BROWSING CHAT MAIL P2P VIDEO VOIP
140 120 100 80 60 40 20 0

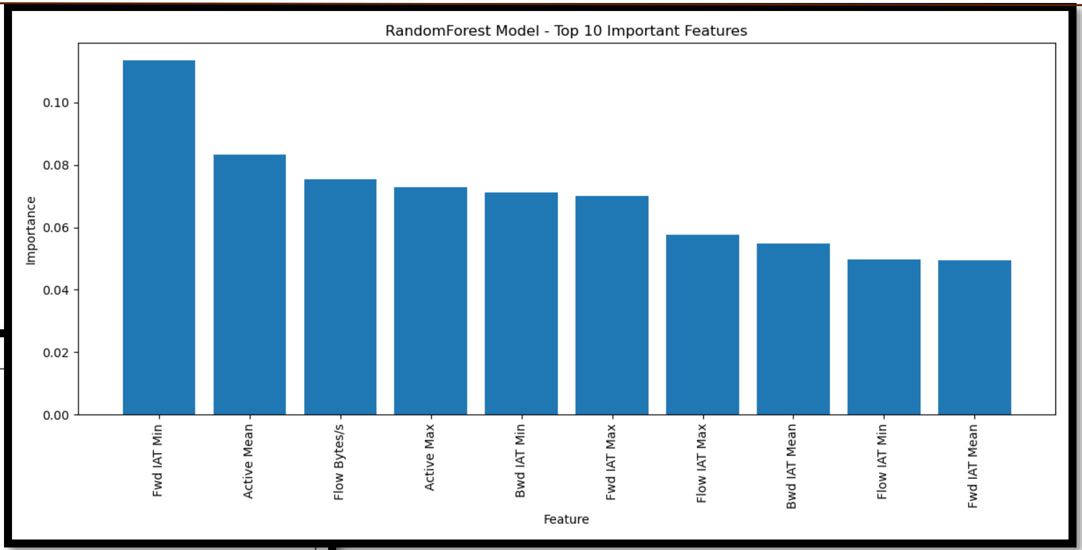
KNN


Random Forest Model Confusion Matrix
True Label
Predicted Label
AUDIO 111 18 1 0 0 2 1
BROWSING 12 76 6 4 3 11 0
CHAT 2 13 13 2 0 2 0
MAIL 1 8 2 14 0 4 0
P2P 1 2 0 0 126 0 1
VIDEO 10 9 3 3 4 24 0
VOIP 2 6 0 0 0 2 144
AUDIO BROWSING CHAT MAIL P2P VIDEO VOIP
140 120 100 80 60 40 20 0

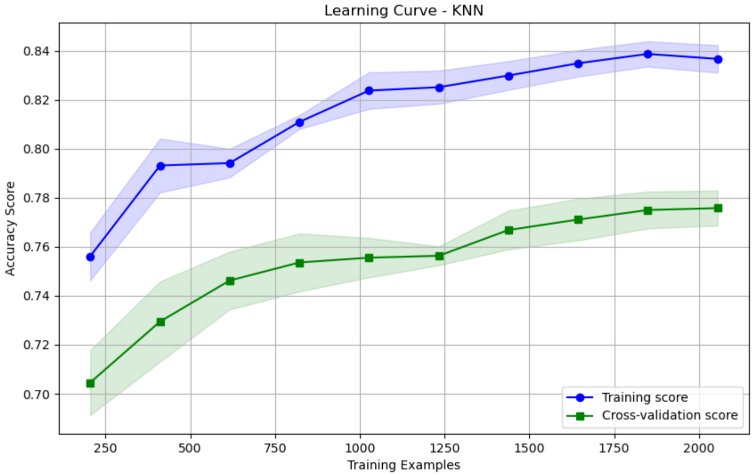
RF

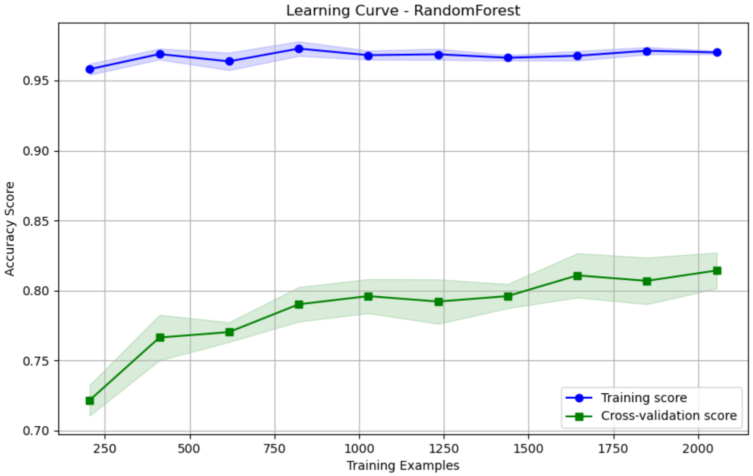
KNN





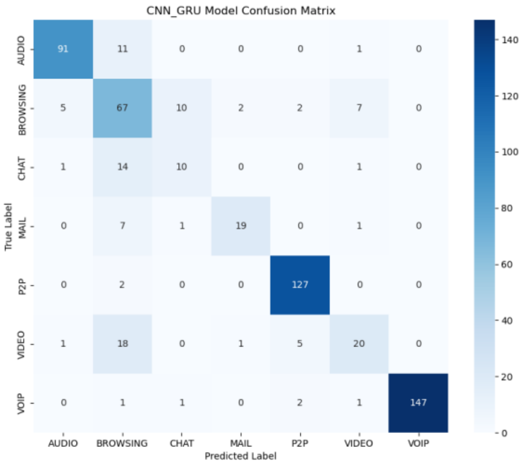
RandomForest



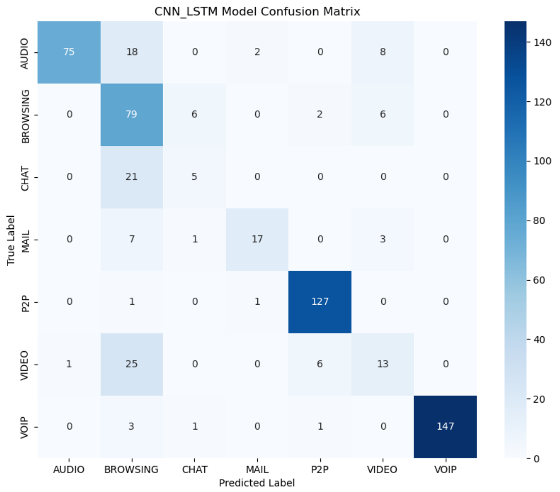


Learning_Curve - KNN

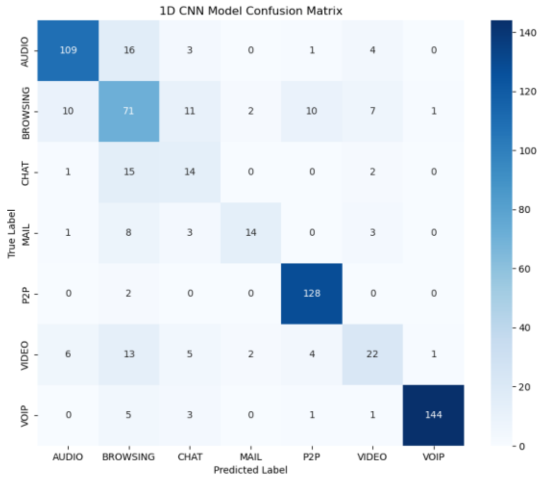
Learning_Curve - RF



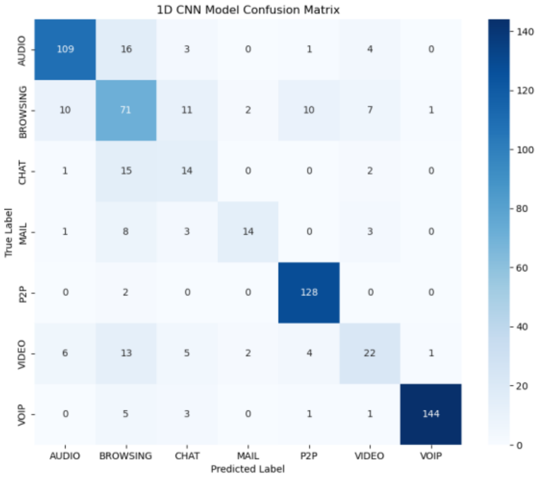
CNN_GRU

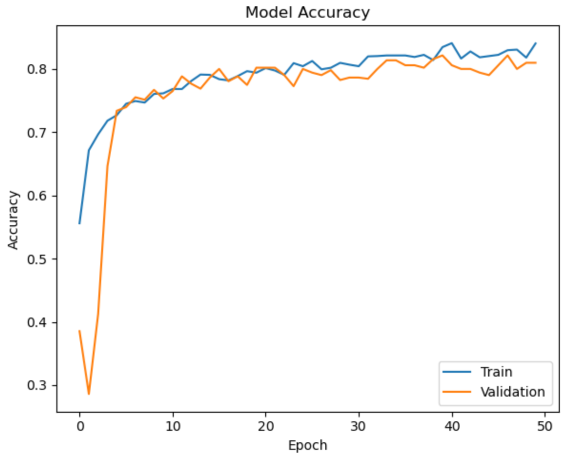


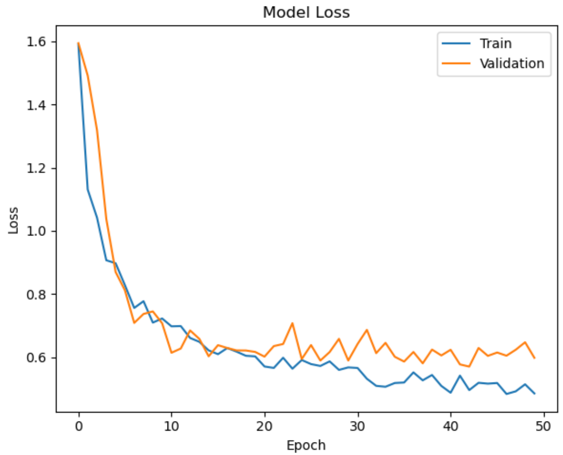
CNN_LSTM



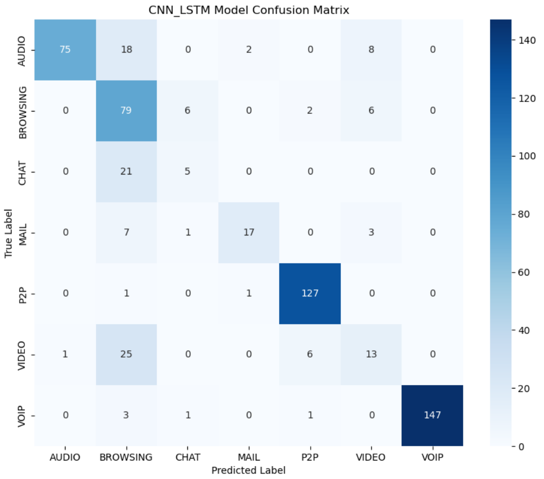
1D_CNN

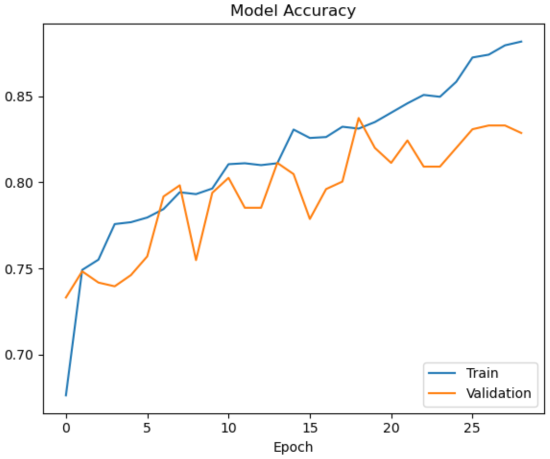

1D CNN Model Confusion Matrix
True Label
Predicted Label
AUDIO
BROWSING
CHAT
MAIL
P2P
VIDEO
VOIP

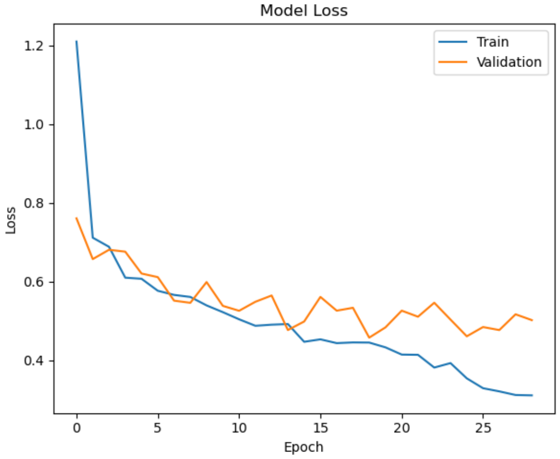

Model Accuracy
Accuracy
Epoch
Train
Validation


Model Loss
Loss
Epoch
Train
Validation

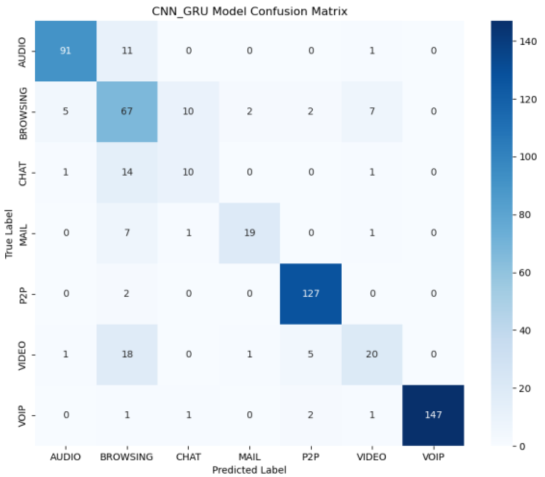
1D_CNN

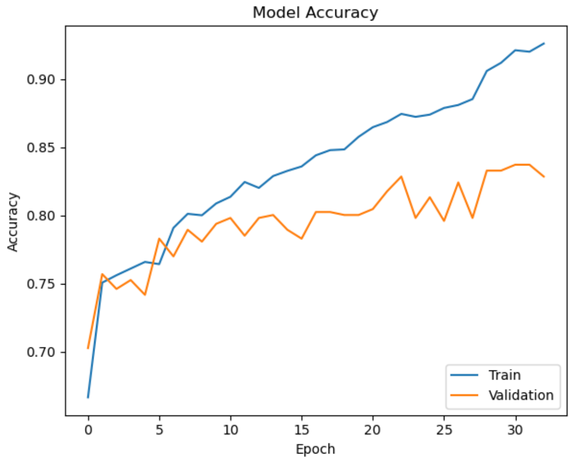

CNN_LSTM Model Confusion Matrix
True Label
Predicted Label
AUDIO
BROWSING
CHAT
MAIL
P2P
VIDEO
VOIP
75 18 0 2 0 8 0
0 79 6 0 2 6 0
0 21 5 0 0 0 0
0 7 1 17 0 3 0
0 1 0 1 127 0 0
1 25 0 0 6 13 0
0 3 1 0 1 0 147
0
20
40
60
80
100
120
140

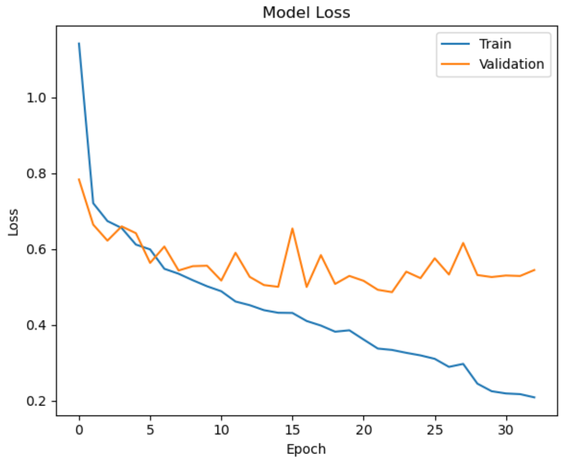

Model Accuracy
Epoch
Train
Validation
0.70
0.75
0.80
0.85
0
5
10
15
20
25


Model Loss
Loss
Epoch
Train
Validation
0.4
0.6
0.8
1.0
1.2
0
5
10
15
20
25

CNN_LSTM

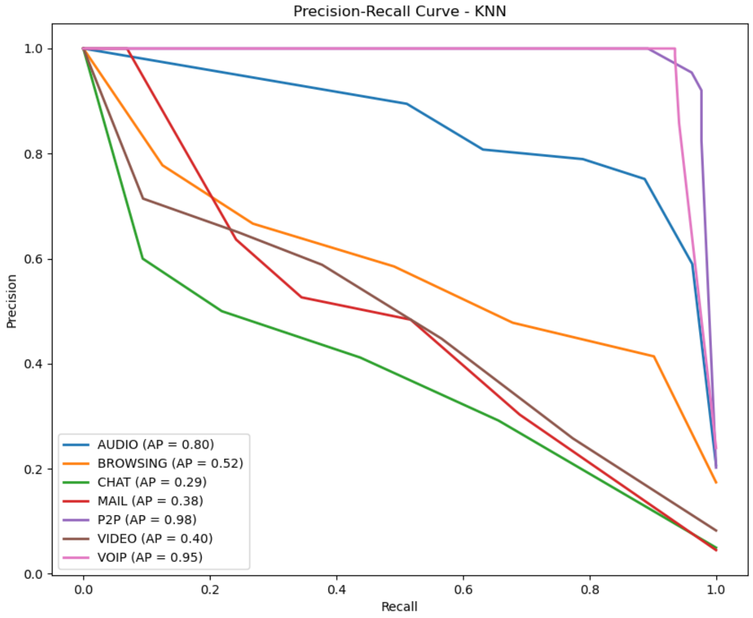

CNN_GRU Model Confusion Matrix
True Label
Predicted Label
AUDIO
BROWSING
CHAT
MAIL
P2P
VIDEO
VOIP


Model Accuracy
Accuracy
Epoch
Train
Validation

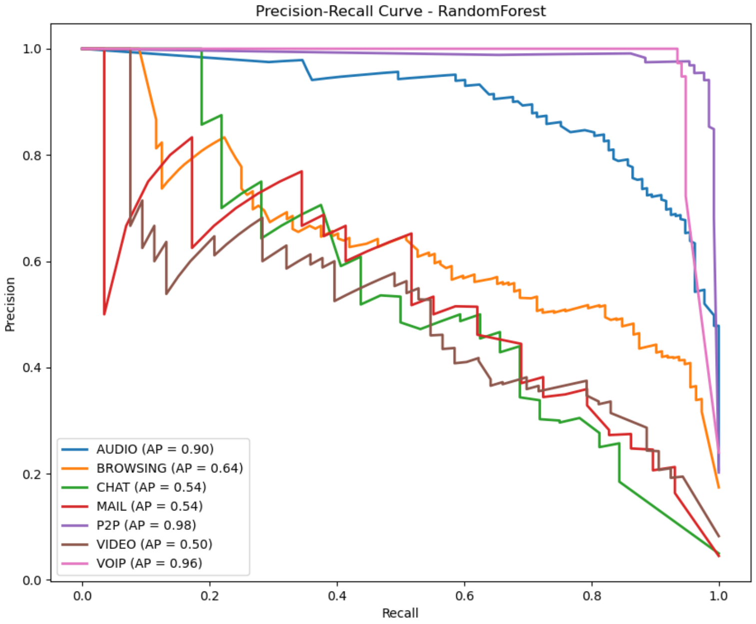

Model Loss
Loss
Epoch
Train
Validation

CNN_GRU

Precision Recall



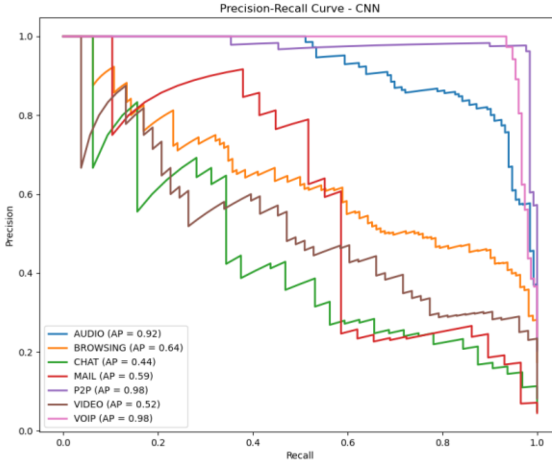
KNN



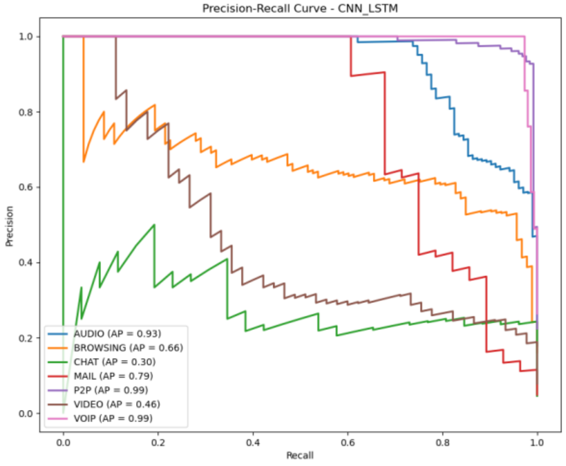
RF

LEHIGH UNIVERSITY

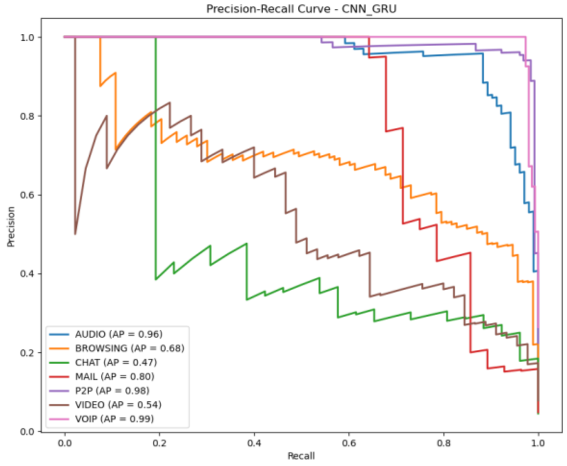
Precision Recall



CNN



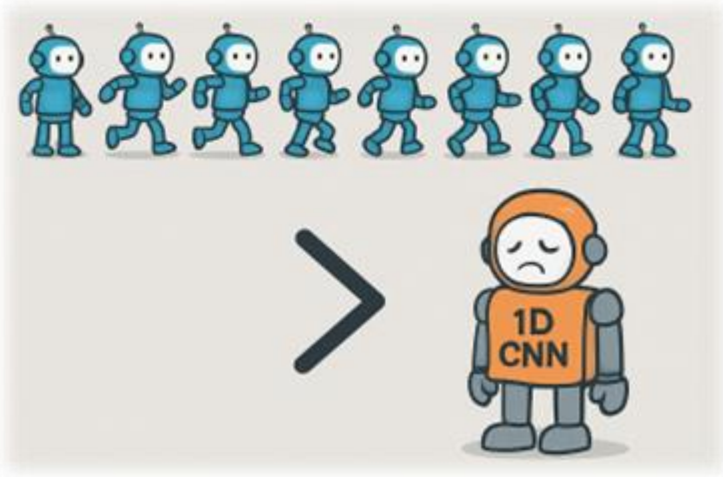
CNN_LSTM



CNN_GRU

Summary and Future Work

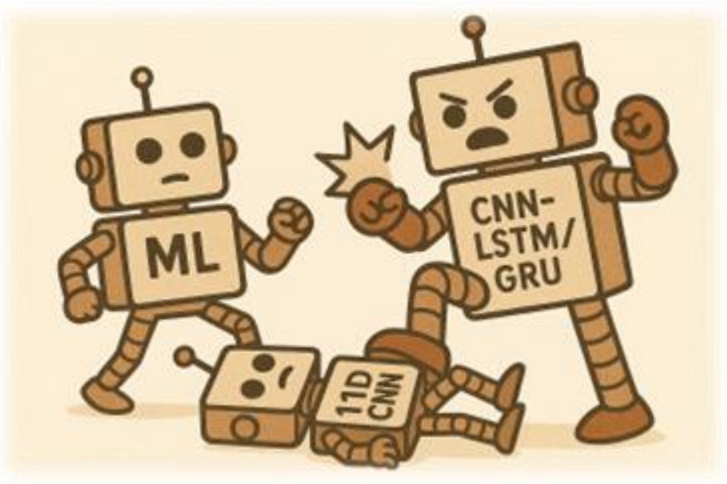
Value of Temporal Features



- **Accuracy uplift**
  - CNN + LSTM: +3.7% vs. static 1D-CNN
  - CNN + GRU: +6.8% vs. static 1D-CNN
- **Flows most helped** ：Audio、browsing
- **Flows less helped:** Chat
- **Costs & limitations**
  - more training time
  - Increased model complexity & memory footprint

Conclusions

- **Key findings**
- Temporal models spent more resources, but it does work
- **Model pros & cons**
- **K-NN**: no training, slow inference
- **RF**: robust, static
- **1D-CNN**: high cost, complex structure, low performance
- **CNN-LSTM/GRU**: higher cost, higher performance



Limitations and future directions

- **Limitations**
- Dataset size is relatively small
- Fixed windowing scheme may lose fine-grained temporal variations
- **Future Directions**
- Develop adaptive windowing strategies to better capture dynamic timing patterns
- Extend to multi-class application classification (e.g., YouTube, Facebook, Skype)

Thank you

Liu, Yucheng

LEHIGH UNIVERSITY