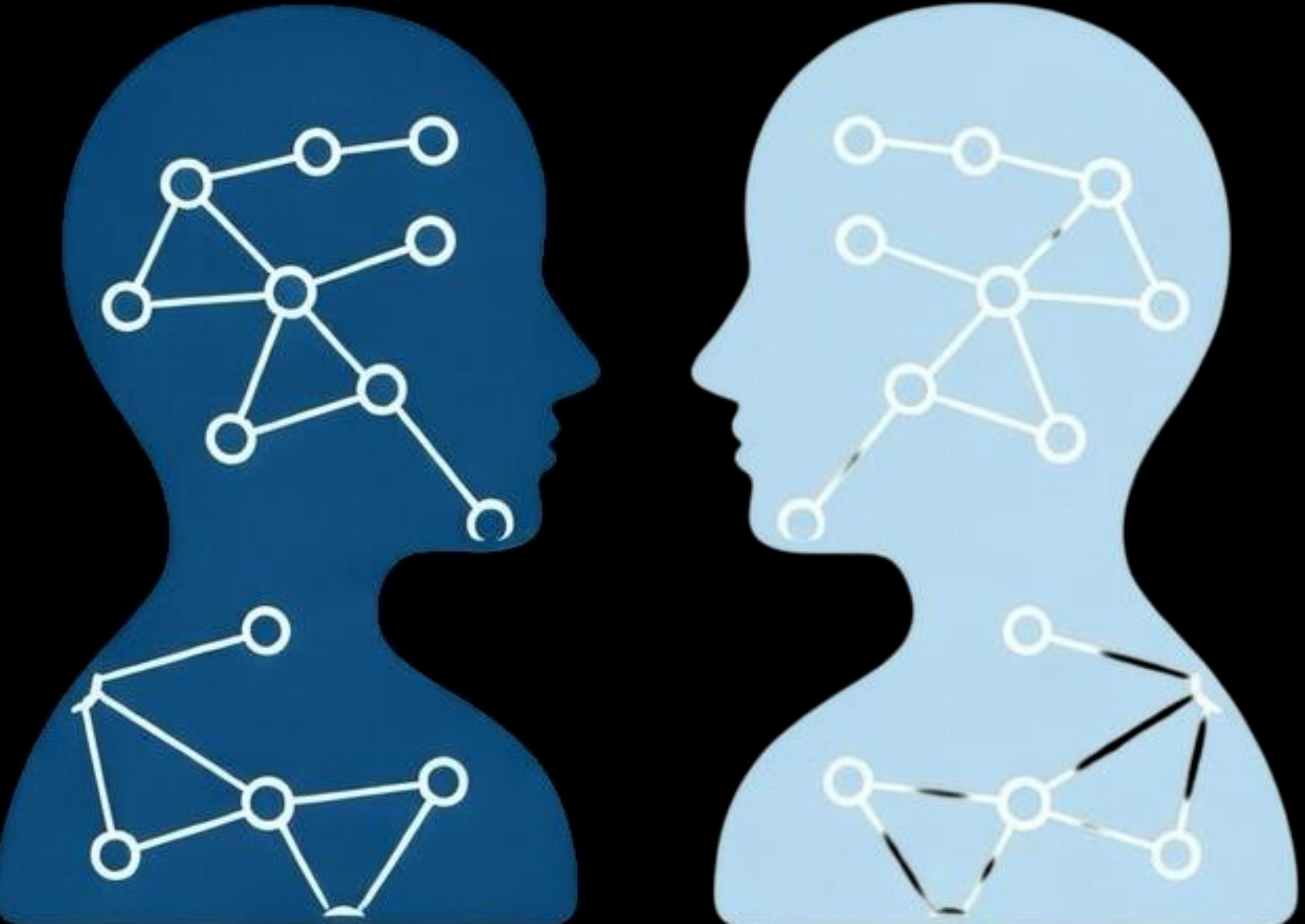


Digital Twin for SDN Networks

Predictive Network Management using Machine Learning

Networking Mod. 2 Project

De Marco Matthew, Lo Iacono Andrea, Revrenna Jago

Agenda

01 Project Overview & Goals

02 System Architecture

03 Key Components

04 Machine Learning Models

05 Code Walkthrough

06 Web Dashboard & Visualization

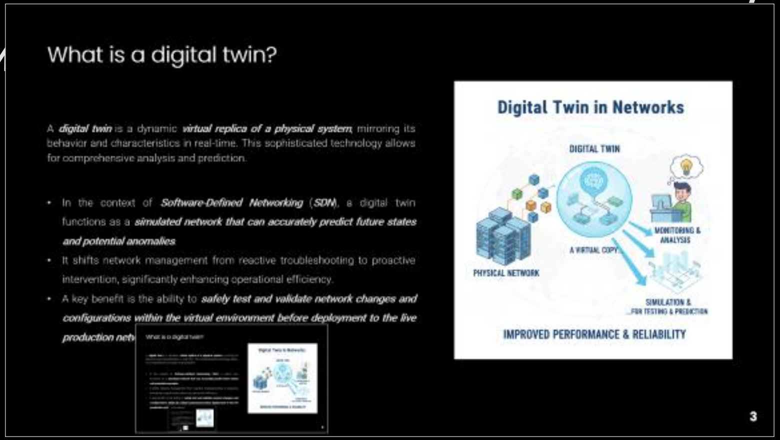
07 Demo & Results

08 Conclusions & Future Work

What is a digital twin?

A ***digital twin*** is a dynamic ***virtual replica of a physical system***, mirroring its behavior and characteristics in real-time. This sophisticated technology allows for comprehensive analysis and prediction.

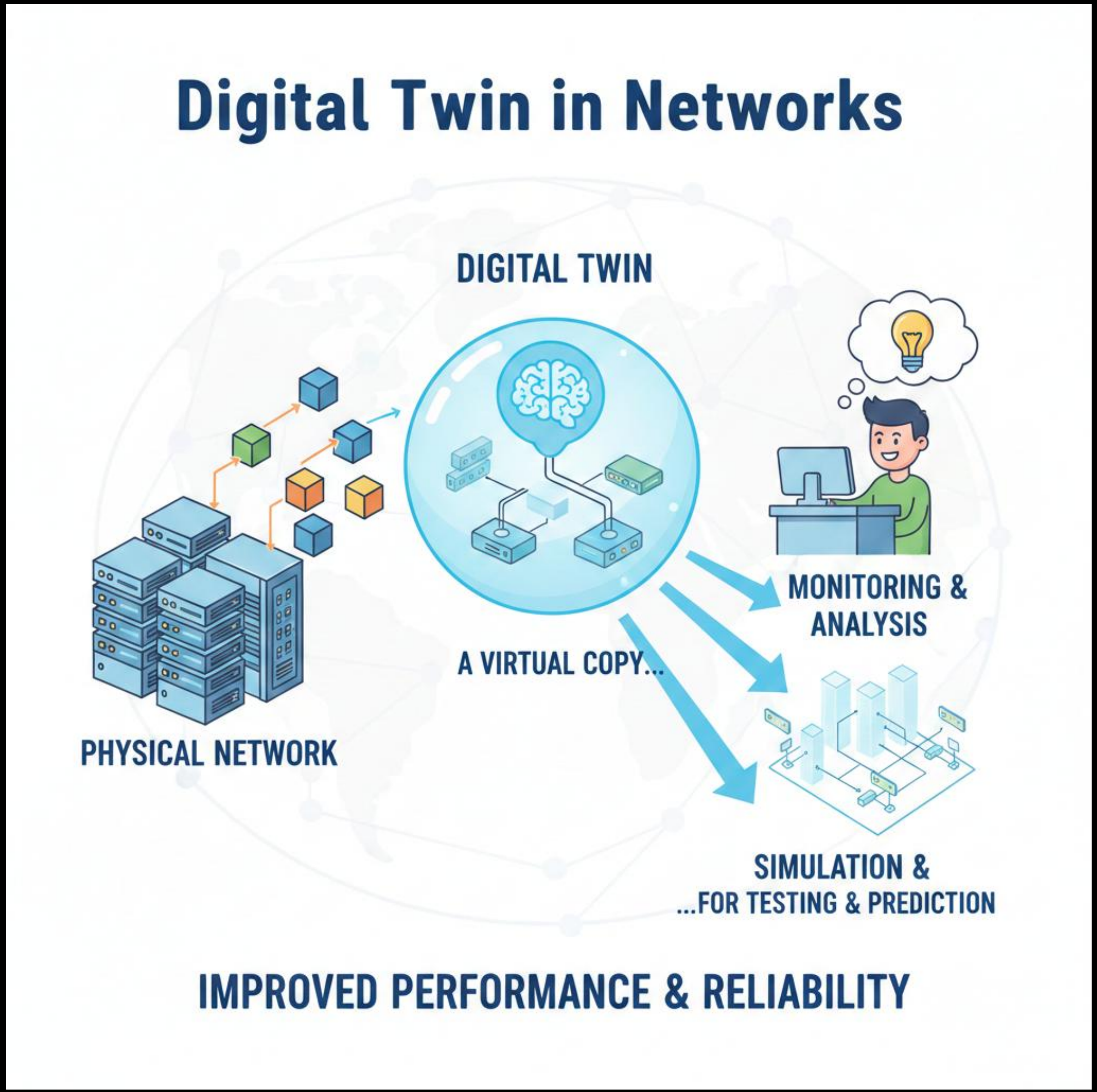
- In the context of ***Software-Defined Networking*** (***SDN***), a digital twin functions as a ***simulated network that can accurately predict future states and potential anomalies***.
- It shifts network management from reactive troubleshooting to proactive intervention, significantly enhancing operational efficiency.
- A key benefit is the ability to ***safely test and validate network changes and configurations within the virtual environment before deployment to the live production netw***


What is a digital twin?

A ***digital twin*** is a dynamic ***virtual replica of a physical system***, mirroring its behavior and characteristics in real-time. This sophisticated technology allows for comprehensive analysis and prediction.

- In the context of ***Software-Defined Networking*** (***SDN***), a digital twin functions as a ***simulated network that can accurately predict future states and potential anomalies***.
- It shifts network management from reactive troubleshooting to proactive intervention, significantly enhancing operational efficiency.
- A key benefit is the ability to ***safely test and validate network changes and configurations within the virtual environment before deployment to the live production netw***

3



Project Goals

Intelligent Digital Twin for SDN

Develop a sophisticated digital twin specifically designed for Software-Defined Networks, capable of dynamic replication and analysis.

Predictive Traffic Analysis

Implement advanced models to predict network traffic patterns 30-60 seconds into the future, enabling timely interventions.

Traffic State Classification

Categorize network traffic into distinct states: *NORMAL*, *ELEVATED, HIGH*, and *CRITICAL*, to provide clear operational insights.

Dual Model Approach

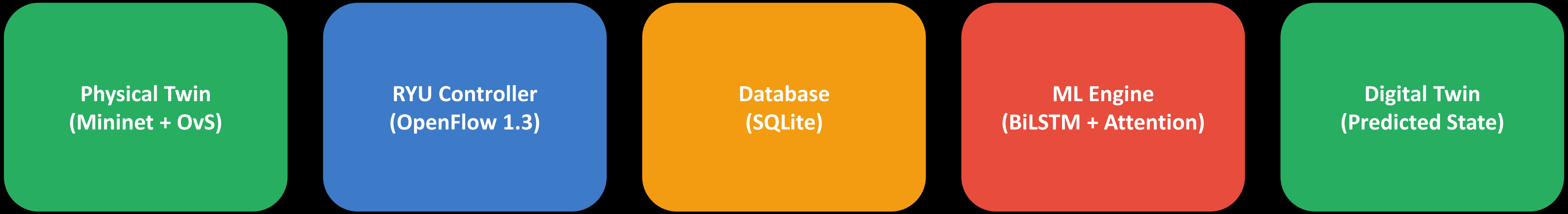
Utilize a dual model strategy, combining a Classifier and a Seq2Seq predictor for each traffic scenario, ensuring comprehensive prediction.

Real-time Visualization

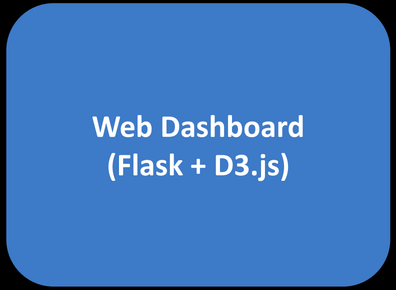
Create a real-time visualization platform to display both current and predicted network states, offering immediate operational awareness.

System Architecture

The system operates by collecting real-time traffic data from the Physical Twin, storing it efficiently in the SQLite database. The ML Engine then processes this data using BiLSTM models to predict future network states 30-60 seconds ahead. These predictions are then used to update the Digital Twin and visualized through the Web Dashboard.


Physical Twin (Mininet + OvS)
RYU Controller (OpenFlow 1.3)
Database (SQLite)
ML Engine (BiLSTM + Attention)
Digital Twin (Predicted State)

Data Flow: Physical Twin → Controller → Database → ML Engine → Predictions → Digital Twin



Machine Learning Engine

Dual Model Approach

1

Specialized Models

Each of the five distinct traffic scenarios (*Normal, Burst, DDoS, Congestion, Mixed*) is addressed by two specialized models, allowing for highly accurate predictions.

2

BiLSTM Classifier

This model categorizes the current network traffic state into one of four critical levels: ***NORMAL, ELEVATED, HIGH, or CRITICAL***, providing immediate situational awareness.

3

Seq2Seq Predictor

Designed to generate a ***detailed 60-second traffic forecast***, this model provides data for graphical visualizations, enabling proactive capacity planning.

4

Comprehensive Training

A total of 10 models are trained (5 scenarios × 2 model types), ensuring robust performance across a wide range of network conditions.

5

Dynamic Switching

The system dynamically switches between models based on the detected traffic conditions, optimizing prediction accuracy and relevance in real-time.

Real-time Network Traffic Classification with BiLSTM

This presentation outlines a robust BiLSTM-based classifier for real-time network traffic analysis, providing critical insights into network health and potential anomalies.

BiLSTM Classifier Architecture

- **Input: 90 timesteps**
- **Processing: BiLSTM & Attention**
- **Output: 4 Classes**

Traffic Classification States

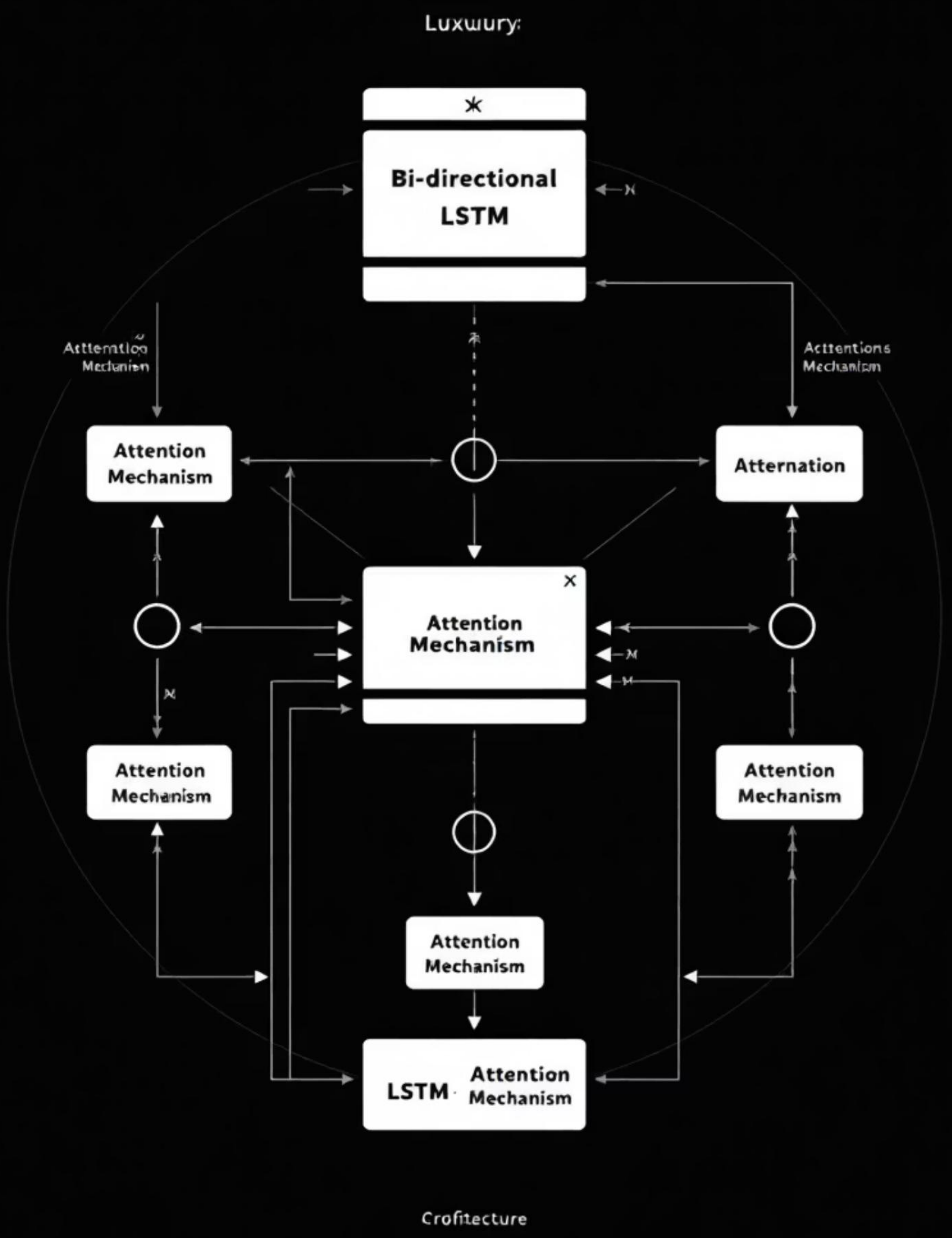
- **NORMAL: < 25%**
- **ELEVATED: > 25 % & < 50 %**
- **HIGH: > 50 % & < 75 %**
- **CRITICAL: > 75%**

Our BiLSTM model captures temporal dependencies from both past and future contexts within the 90-timestep input. The system predicts network states with a ***60 second horizon***, critical for preemptive action.

BiLSTM Classifier: Architectural Deep Dive

- **Bidirectional LSTM**: provides a comprehensive understanding of traffic flow dynamics.
- **Attention Mechanism**: Dynamically weights the importance of different timesteps, allowing the model to focus on the most relevant historical traffic data for accurate prediction.
- **LayerNorm + GELU Activation**: Ensures stable and efficient training of the deep neural network, improving convergence and generalization performance.

```
class BiLSTMClassifier(nn.Module):
    def __init__(self, input_size, lstm_hidden, lstm_layers, num_classes,
dropout=0.3):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=lstm_hidden,
            num_layers=lstm_layers,
            batch_first=True,
            bidirectional=True
        )
        self.attention = nn.Sequential(
            nn.Linear(lstm_hidden * 2, lstm_hidden),
            nn.Tanh(),
            nn.Linear(lstm_hidden, 1)
        )
        self.classifier = nn.Sequential(
            nn.LayerNorm(lstm_hidden * 2),
            nn.Linear(lstm_hidden * 2, lstm_hidden),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(lstm_hidden, num_classes)
        )
```



Seq2Seq Predictor: Graph Forecasting for Network Visualization

- **CNN + BiLSTM Architecture**: A hybrid model combining Convolutional Neural Networks for feature extraction and Bidirectional LSTMs for sequence learning, optimized for time-series forecasting.
- **60-Second Traffic Predictions**: Generates high-fidelity predictions of future network traffic behavior, essential for preemptive visualization and operational planning.
- **Dashboard Integration**: These predictions are directly utilized for plotting dynamic traffic graphs on the dashboard.

```
class A100Seq2Seq(nn.Module):
    """CNN + BiLSTM Seq2Seq for graph visualization"""
    def __init__(self, config):
        super().__init__()
        # CNN feature extractor
        self.conv = nn.Sequential(
            nn.Conv1d(1, 64, kernel_size=5, padding=2),
            nn.ReLU(),
            nn.Conv1d(64, 128, kernel_size=3, padding=1)
        )
        # BiLSTM encoder
        self.lstm = nn.LSTM(input_size=128, hidden_size=256,
                            num_layers=2, bidirectional=True)
        # Output: 60-second prediction horizon
        self.fc = nn.Linear(hidden * 2, prediction_horizon)
```

State Predictor: Real-time Inference for Network State Assessment

- **Historical Data Input**: The model processes 90 seconds of historical traffic data, leveraging a comprehensive temporal context for robust predictions.
- **Classified Output with Confidence**: Provides a precise classification of the network state (Normal, Elevated, High, Critical) coupled with a confidence score, indicating prediction reliability.
- **Estimated Future Bandwidth**: Based on the predicted state, the system estimates future bandwidth requirements.

```python
def predict(self, historical_data: np.ndarray) -> Dict:
    # Normalize input data
    if self.scaler is not None:
        data_scaled = self.scaler.transform(historical_data)

    # Convert to tensor and predict
    x = torch.FloatTensor(data_scaled).unsqueeze(0).to(self.device)
    with torch.no_grad():
        logits = self.model(x)
        probs = torch.softmax(logits, dim=1)
        state_id = torch.argmax(probs, dim=1).item()
        confidence = probs[0, state_id].item()

    return {
        'state': self.class_names[state_id],  # NORMAL, ELEVATED, HIGH, CRITICAL
        'confidence': confidence,
        'estimated_bandwidth': self._estimate_bandwidth(state_id)
    }
```

Empirical Validation: Training Results Across Scenarios

Our models underwent rigorous training and validation across various network scenarios to ensure high performance and reliability. The following sections detail the training results for both the Classifier (state prediction) and the Seq2Seq Predictor (graph forecasting) under distinct traffic conditions.

1

Normal Scenario

Baseline performance for healthy network conditions.

2

Burst Scenario

Evaluating response to sudden traffic surges.

3

Congestion Scenario

Performance under sustained high-load conditions.

4

DDoS Scenario

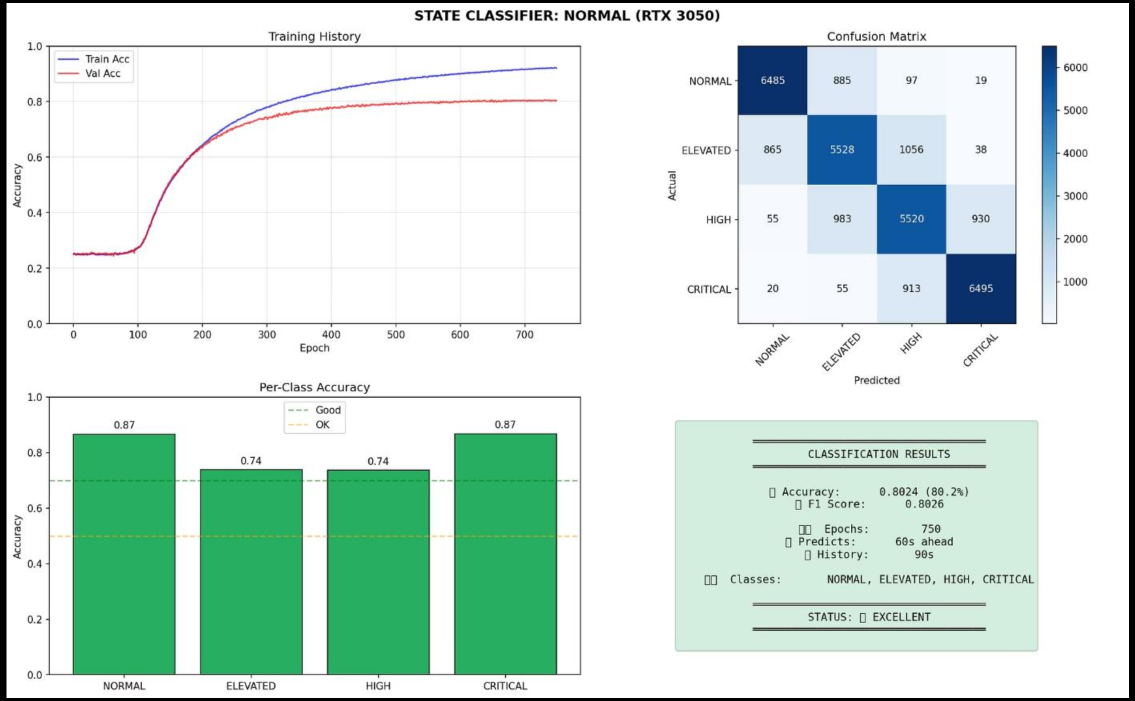
Detection capabilities during malicious attacks.

5

Mixed Scenario

Comprehensive evaluation with diverse traffic patterns.

Normal Scenario: Optimized Performance Under Baseline Traffic



Actual \ Predicted	NORMAL	ELEVATED	HIGH	CRITICAL
NORMAL	6485	885	97	19
ELEVATED	865	5528	1056	38
HIGH	55	983	5520	930
CRITICAL	20	55	913	6495

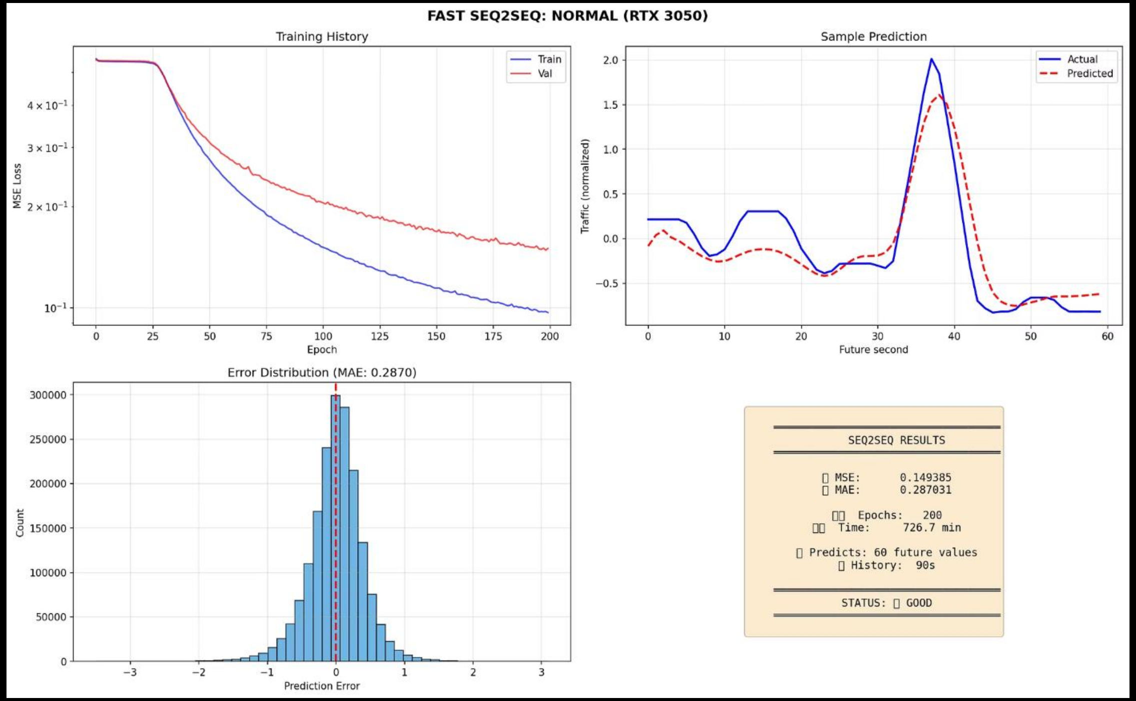
```
        CLASSIFICATION RESULTS

  □ Accuracy:      0.8024 (80.2%)
    □ F1 Score:      0.8026

    □□  Epochs:        750
   □ Predicts:      60s ahead
     □ History:       90s

□□  Classes:       NORMAL, ELEVATED, HIGH, CRITICAL

        STATUS: □ EXCELLENT
```



```
      SEQ2SEQ RESULTS

   □ MSE:      0.149385
   □ MAE:      0.287031

    □□  Epochs:   200
  □□  Time:     726.7 min

 □ Predicts: 60 future values
     □ History:  90s

      STATUS: □ GOOD
```

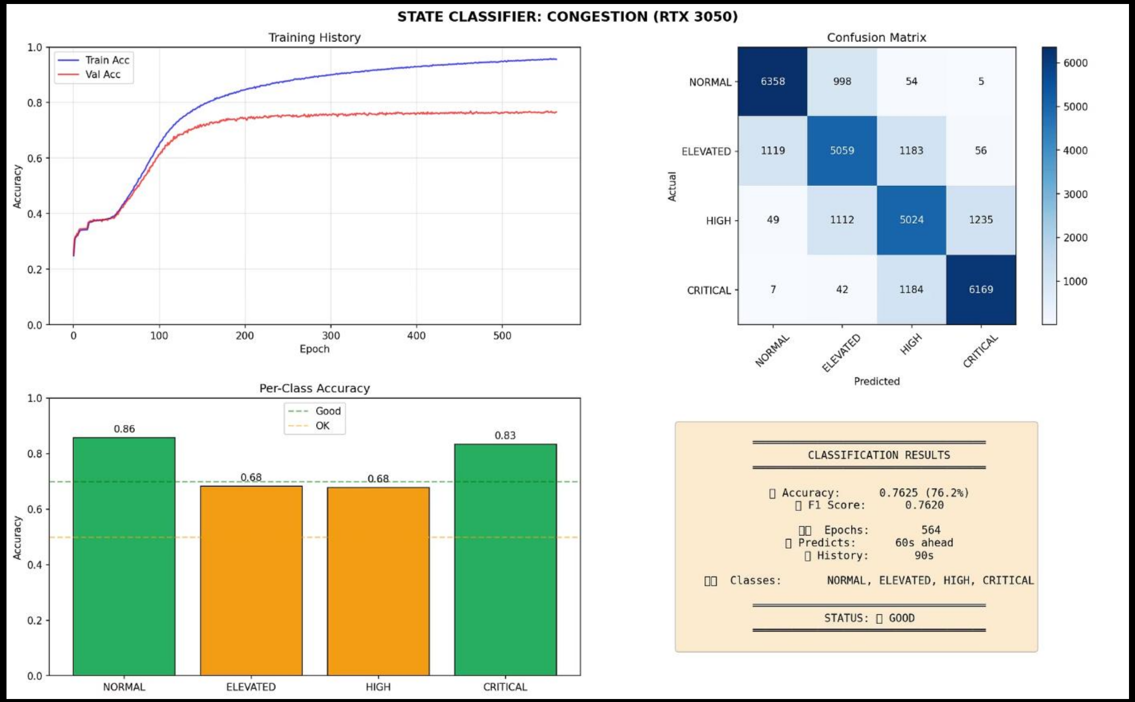
Classifier (State Prediction)

The classifier demonstrates high accuracy in identifying 'NORMAL' network conditions, with minimal false positives. The model reliably maintains a stable prediction confidence, ensuring network managers receive consistent and trustworthy assessments during typical operations.

Seq2Seq (Graph Forecasting)

The Seq2Seq model accurately forecasts normal traffic patterns, producing smooth and stable prediction graphs. This enables operators to confirm baseline network health and plan resources effectively without anticipating unusual fluctuations.

Congestion Scenario: Detecting and Predicting Overload Conditions



Actual \ Predicted	NORMAL	ELEVATED	HIGH	CRITICAL
NORMAL	6358	998	54	5
ELEVATED	1119	5059	1183	56
HIGH	49	1112	5024	1235
CRITICAL	7	42	1184	6169

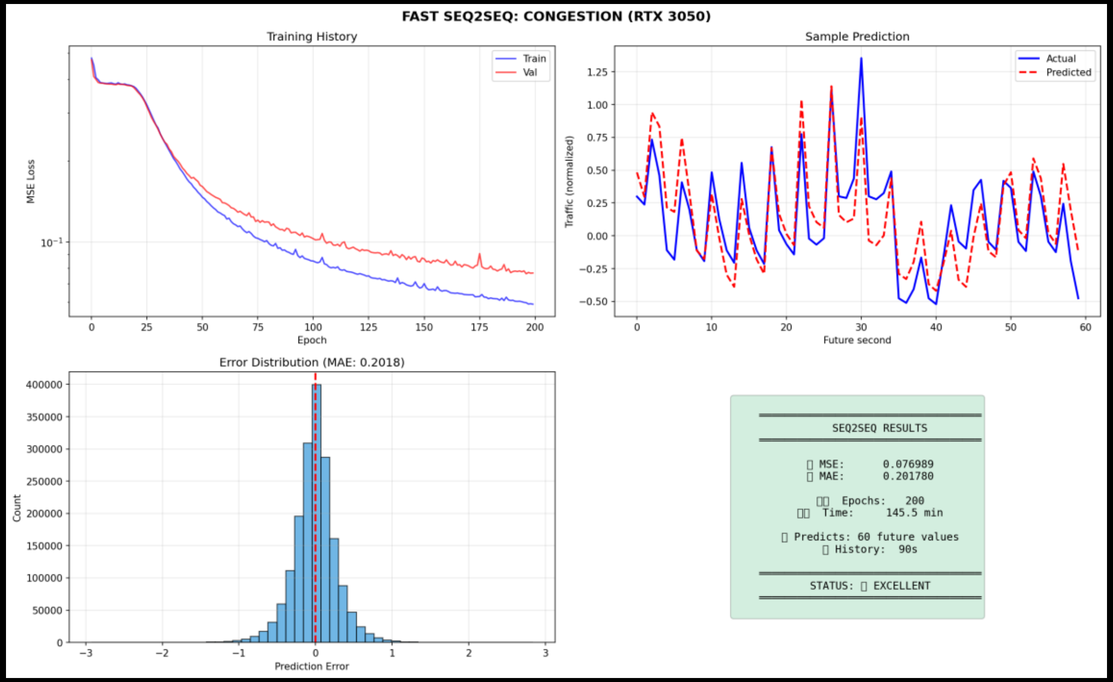
```
CLASSIFICATION RESULTS

□ Accuracy:      0.7625 (76.2%)
  □ F1 Score:      0.7620

  □□  Epochs:        564
 □ Predicts:      60s ahead
   □ History:       90s

□□  Classes:       NORMAL, ELEVATED, HIGH, CRITICAL

STATUS: □ GOOD
```



```
SEQ2SEQ RESULTS

□ MSE:      0.076989
□ MAE:      0.201780

 □□  Epochs:   200
□□  Time:     145.5 min

□ Predicts: 60 future values
   □ History:  90s

STATUS: □ EXCELLENT
```

Classifier (State Prediction)

The classifier accurately identifies and maintains its classification of 'HIGH' or 'CRITICAL' states during sustained congestion. High confidence scores during these events ensure operators are aware of prolonged network strain, facilitating effective intervention.

Seq2Seq (Graph Forecasting)

The Seq2Seq model provides reliable forecasts of persistent high traffic volumes and potential degradation during congestion. This foresight allows for proactive traffic shaping and rerouting strategies to alleviate pressure points.

The Digital Twin Model Dispatcher

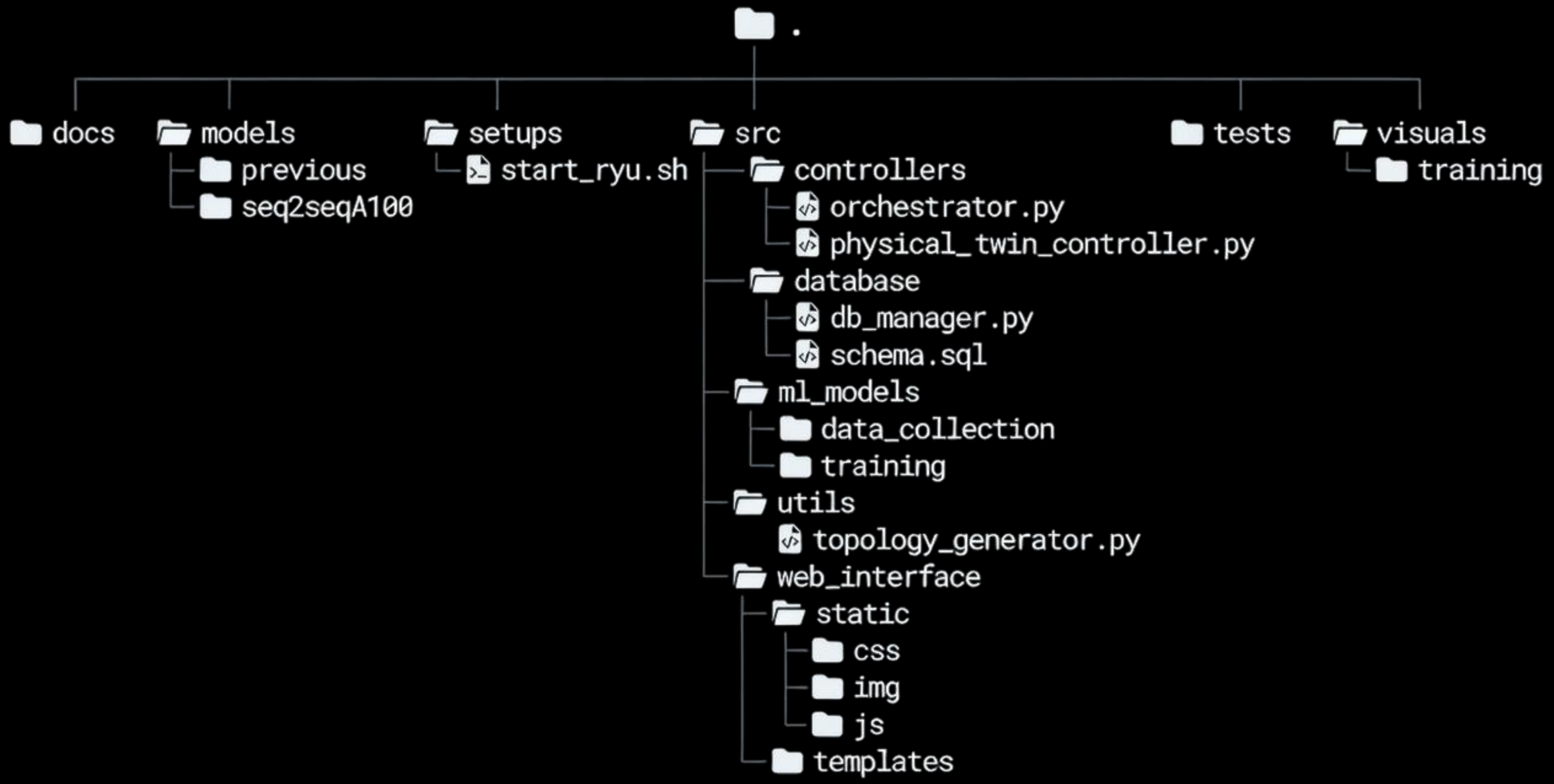
Model Dispatcher Architecture

The Model Dispatcher acts as the central coordination layer between the Digital Twin and the ML models. It provides a unified interface for:

- **State Classification**: Determines the current network state (Normal, Congested, Failure) using a trained classifier.
- **Traffic Prediction**: Leverages a Seq2Seq model to forecast future traffic patterns.
- **Model Lifecycle Management**: Handles loading, caching, and inference for all ML models.
- **Abstraction Layer**: Decouples the SDN controller logic from specific model implementations.

This design follows the Strategy Pattern, allowing easy swapping or updating of models without modifying the controller code.

Project Architecture



```
.
docs
models
    previous
    seq2seqA100
setups
    start_ryu.sh
src
    controllers
        orchestrator.py
        physical_twin_controller.py
    database
        db_manager.py
        schema.sql
    ml_models
        data_collection
        training
    utils
        topology_generator.py
    web_interface
        static
            css
            img
            js
        templates
tests
visuals
    training
```

Orchestrator – Scenario Detection

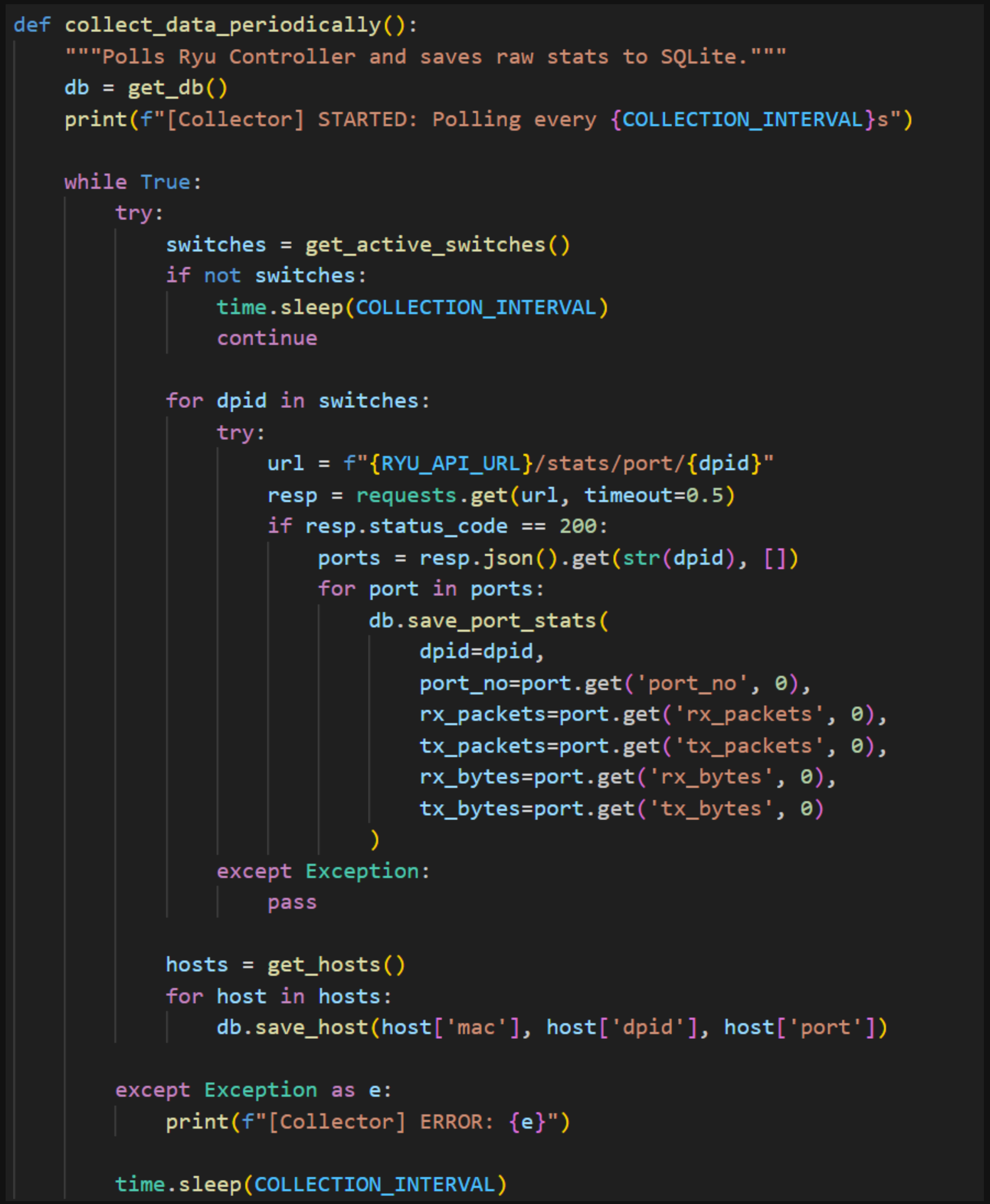
- Continuously monitors traffic from the Physical Twin.
- Dynamically selects appropriate model pairs based on traffic patterns.
- Runs both Classifier (state) and Seq2Seq (graph) for each prediction.

Scenario Dispatcher Logic

```
# 2. Thresholds (30 Mbps Link)
    CAPACITY = 30 * 1e6 / 8
    CRITICAL_LOAD_THRESHOLD = CAPACITY * 0.80
    HIGH_LOAD_THRESHOLD = CAPACITY * 0.50
    SATURATION_THRESHOLD = CAPACITY * 0.90
    # --- DECISION LOGIC ---
    # Rule 1: Saturation -> DDoS
    if recent_avg > SATURATION_THRESHOLD:
        return 'DDOS'
    # Rule 2: Critical Load -> DDoS vs Burst
    if recent_avg > CRITICAL_LOAD_THRESHOLD:
        # Volatility Check
        if recent_std < (recent_avg * 0.3):
            return 'DDOS'
        # Sustained High Check (25s)
        check_duration = 25
        if len(traffic_window) > check_duration:
            sustained_window = traffic_window[-check_duration:]
            if np.min(sustained_window) > CRITICAL_LOAD_THRESHOLD:
                return 'DDOS'
        return 'BURST'
```

Data Collection Pipeline

- Polls ***RYU REST API*** every 1 second for real-time insights.
- Collects comprehensive port statistics, including rx/tx bytes and packets.
- *Stores* time-series data efficiently in a SQLite database for rapid access and analysis.



```python
def collect_data_periodically():
    """Polls Ryu Controller and saves raw stats to SQLite."""
    db = get_db()
    print(f"[Collector] STARTED: Polling every {COLLECTION_INTERVAL}s")

    while True:
        try:
            switches = get_active_switches()
            if not switches:
                time.sleep(COLLECTION_INTERVAL)
                continue

            for dpid in switches:
                try:
                    url = f"{RYU_API_URL}/stats/port/{dpid}"
                    resp = requests.get(url, timeout=0.5)
                    if resp.status_code == 200:
                        ports = resp.json().get(str(dpid), [])
                        for port in ports:
                            db.save_port_stats(
                                dpid=dpid,
                                port_no=port.get('port_no', 0),
                                rx_packets=port.get('rx_packets', 0),
                                tx_packets=port.get('tx_packets', 0),
                                rx_bytes=port.get('rx_bytes', 0),
                                tx_bytes=port.get('tx_bytes', 0)
                            )
                except Exception:
                    pass

            hosts = get_hosts()
            for host in hosts:
                db.save_host(host['mac'], host['dpid'], host['port'])

        except Exception as e:
            print(f"[Collector] ERROR: {e}")

        time.sleep(COLLECTION_INTERVAL)
```

Database Layer

- Utilizes SQLite for lightweight, embedded storage, optimizing performance.
- Manages time-series traffic data with advanced indexing for fast and efficient queries.
- Incorporates prediction storage for continuous accuracy validation and model refinement.

```
class DatabaseManager:
 def save_port_stats(self, dpid, port_no, rx_bytes, tx_bytes):
 with self._get_connection() as conn:
 conn.execute(, (dpid, port_no, rx_bytes, tx_bytes))

 def get_recent_traffic(self, link_id, duration_seconds=60):
 """Get traffic history for ML prediction input."""
 # Returns DataFrame with timestamps and bytes_sent
 return pd.read_sql(query, conn)

 def store_prediction(self, dpid, port_no, predicted_bytes, timestamp):
 """Store ML predictions for validation."""
 conn.execute('INSERT INTO predictions...', ...)
```

Key Features & Innovations

1

Dual Model Architecture

Classifier + Seq2Seq models deployed per network scenario.

2

10 Trained Models

A comprehensive suite of 10 models (5 scenarios × 2 model types).

3

Dynamic Model Switching

Automatic scenario detection and agile model adaptation.

4

Real-time Visualization

Immediate traffic state classification with an intuitive web dashboard.

5

Distributed Architecture

Physical Twin VM + Digital Twin VM for robust operation.

6

60-Second Horizon

Accurate traffic graph prediction over a 60-second horizon.

Results & Performance

High Accuracy

Successfully predicts traffic states with notable precision.

Effective Classification

Classifier models reliably categorize network conditions.

Smooth Forecasts

Seq2Seq models provide smooth 60-second graph forecasts.

Sub-second Inference

Real-time predictions delivered with sub-second inference speed.

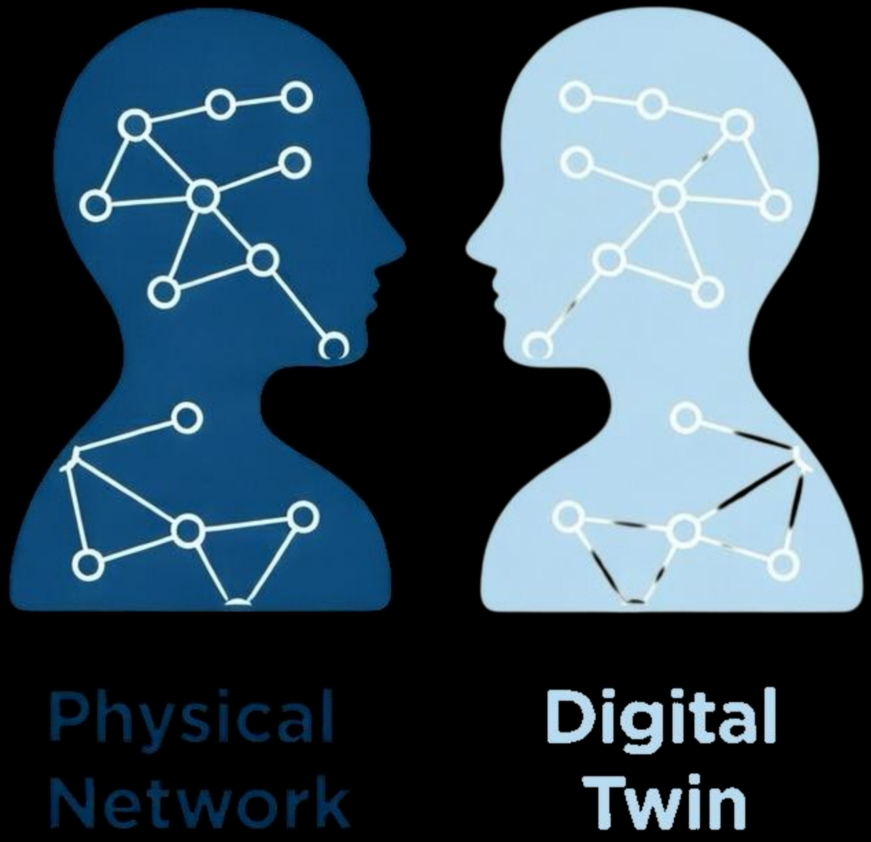
GPU-Trained Models

Leveraging NVIDIA A100/RTX 3050 for enhanced training.

Intuitive Dashboard

Web dashboard offers clear network visualization.

Summary



- Successfully built a **complete Digital Twin system** specifically for SDN networks.
- **Implemented a Dual Model Approach**: A Classifier for state detection and a Seq2Seq model for traffic forecasting per scenario.
- Developed and trained **10 models**, covering 5 distinct traffic scenarios.
- Achieved **real-time classification** and accurate 60-second traffic forecasting capabilities.
- Designed a **modular and extensible architecture**, ensuring future adaptability and scalability.

Future Work

01

Automated Rerouting

Implement automatic traffic rerouting logic based on predictive analysis.

02

Complex Topologies

Expand the system to support and manage more complex network topologies.

03

Hardware Integration

Integrate with real hardware SDN switches for practical deployment.

04

Predictive QoS Management

Add predictive Quality of Service management features.

05

Transformer Models

Explore transformer-based models for extended prediction horizons.

06

sFlow-RT Implementation

Incorporate sFlow-RT for enhanced flow monitoring and analytics.

Team & Acknowledgments

Team Members

- ***De Marco Matthew***
- ***Lo Iacono Andrea***
- ***Revrenna Jago***

Course: *Networking 2* (Softwarized and Virtualized Mobile Networks)

Professor: ***Prof. Fabrizio Granelli***

University: *University of Trento* - Academic Year 2025-2026