# CATOR: A Confidence-Adaptive Dual-Plane In-Switch Orchestrated Resilience Framework for Multi-Vector DDoS Defense in Software-Defined IoT

**Technical Manual**
Version 1.0 – August 2025

---

## 1. Introduction

### Motivation

IoT networks are highly vulnerable to large-scale DDoS attacks due to the diversity, resource constraints, and sheer number of connected devices. Traditional detection systems often fail to scale or adapt quickly to dynamic attack patterns.

**CATOR** is designed to address these challenges by integrating:

- **P4-based feature extraction** for real-time monitoring at the switch level.
- **Adaptive window-based traffic analysis** to dynamically adjust detection granularity.
- **A dynamic ensemble classifier (DYNEX)** optimized for detection accuracy and response time.
- **Adaptive cooperative mitigation (ACM)** across multi-controller SDN architectures.

---

## 2. Development Process Overview

The CATOR implementation was built in **11 structured steps**, each contributing a functional module to the final deployable system:

1. **Project Structure Setup** – Create folder layout, configs, placeholders.
2. **Configuration Files** – Define JSON configs for datasets, switches, controllers, and mitigation.
3. **P4 Programs** – Implement AWTA, FCSTD, and feature extraction in P4.
4. **DTW Implementation** – Dynamic Time-based Windowing logic for traffic windows.
5. **AWTA Python Integration** – Python controller for P4-based AWTA.
6. **DYNEX Implementation** – Dynamic ensemble classifier with meta-learner.
7. **ACM Implementation** – Local and distributed mitigation logic.
8. **Multi-Controller SDN** – Central and domain controllers + failover logic.

9. **Dataset Processing Modules** – Preprocessing and loading scripts.
10. **Model Training & Evaluation** – Train/evaluate DYNEX with standard metrics.
11. **Deployment Scripts** – Mininet topology, controller startup, P4 deployment.
12. **Test Suite** – Unit tests for DTW, AWTA, DYNEX, ACM.

---

## 3. Step-by-Step Implementation

### Step 1 – Project Structure

**Goal:** Create a modular codebase that mirrors SDN + ML pipelines.
**Key Files:** requirements.txt, .gitignore, empty folders with __init__.py where needed.
**Integration:** Ensures clean separation of concerns between P4 logic, ML, controllers, and deployment.

---

### Step 2 – Configuration Files

- **cator_config.json** → global thresholds, parameters.
- **datasets_config.json** → dataset paths, splits.
- **p4_switch_config.json** → device IDs, ports, feature extraction mapping.
- **controller_config.json** → multi-controller topology details.
- **mitigation_policies.json** → drop rules, rate limits.

---

### Step 3 – P4 Programs

- **awta.p4** – Implements adaptive window counters, feature extraction triggers.
- **fcstd.p4** – State table for flow classification.
- **features_extraction.p4** – Extracts 21 predefined statistical features.

---

### Step 4 – DTW Implementation

**File:** src/dtw.py
**Purpose:** Dynamically adjusts traffic window sizes based on entropy ($\theta$min, $\theta$max).
**Integration:** Runs before classification to adapt feature windowing to traffic volatility.

---

**Step 5 – AWTA Python Integration**

**File:** src/awta.py
**Purpose:** Communicates with P4Runtime API, fetches extracted features, passes them to DYNEX for classification.
**Integration:** Real-time bridge between data plane and control plane ML.

---

**Step 6 – DYNEX Implementation**

**File:** src/dynex.py
**Purpose:** Ensemble of Random Forest, XGBoost, and ANN meta-learner.
**Integration:** Classifies traffic as normal/attack; integrated into AWTA loop.

---

**Step 7 – ACM Implementation**

**File:** src/acm.py
**Purpose:** Applies mitigation either locally or via central controller.
**Integration:** Links DYNEX alerts to SDN mitigation flows.

---

**Step 8 – Multi-Controller SDN**

**Files:**

- controllers/central_controller.py → orchestrates multi-domain mitigation.
- controllers/domain_controller.py → domain-specific mitigation.
- controllers/failover_handler.py → promotes backup controllers if failures occur.

---

**Step 9 – Dataset Processing Modules**

**Files:** datasets/preprocessing.py, datasets/loaders.py
**Purpose:** Preprocess, normalize, and split datasets for training/testing.
**Integration:** Also applied in real-time to incoming feature vectors.

---

**Step 10 – Model Training & Evaluation**

**Files:**

- training/train_dynex.py – trains DYNEX.
- training/eval_dynex.py – evaluates with DA, F1, MCC, ATRT.

---

**Step 11 – Deployment Scripts**

**Files:**

- deployment/mininet_topology.py – sets up P4 Mininet testbed.
- deployment/start_controllers.sh – launches controllers.
- deployment/deploy_p4_switches.sh – compiles/loads P4.

---

**Step 12 – Test Suite**

**Files:** tests/test_*.py
**Purpose:** Unit tests for core modules before integration testing.

---

## 4. CATOR Integration Flow

**Workflow:**

1. AWTA (P4) extracts features from traffic flows.
2. DTW adjusts analysis window size.
3. DYNEX classifies flow as benign or malicious.
4. ACM decides mitigation strategy.
5. Controllers push mitigation flows to P4 switches.

---

## 5. Emulated Deployment Instructions

**Environment:** Ubuntu 20.04, Mininet, BMv2, P4C, Ryu, Flask

**Steps:**

1. Install dependencies: *pip install -r requirements.txt*
2. Start controllers: *bash deployment/start_controllers.sh*
3. Deploy P4 programs: *bash deployment/deploy_p4_switches.sh*
4. Run Mininet topology: *sudo python deployment/mininet_topology.py*

**6. Real-World Deployment Instructions**

**Hardware Example:** Intel Tofino

**Steps:**

1. Compile P4 for your ASIC target.
2. Update p4_switch_config.json with real device IDs & ports.
3. Connect controllers to production network.
4. Start ACM and AWTA Python processes.
5. Verify gRPC sessions between controllers and switches.

---

**7. Example Outputs**

**Detection Example:**

```
[AWTA] Features extracted from s1

[DYNEX] Classified as attack (confidence=0.94)

[ACM] Triggering distributed mitigation for DDoS

[CentralController] Sending mitigation request to C1

[DomainController-C1] Dropping flow 10.0.0.1 -> 10.0.0.3
```

**8. Troubleshooting**

- **Switch not connecting:** Check gRPC port mapping in p4_switch_config.json.
- **Dataset errors:** Ensure correct CSV delimiter and no header mismatches.
- **Low accuracy:** Retrain DYNEX with larger dataset split.