실시간 추론 시스템 아키텍처

개요

Violence Detection 시스템의 `inference.realtime` 모드의 전체 아키텍처와 로직 흐름을 상세히 설명한다. 비디오/RTSP 입력부터 이벤트 관리 및 최종 결과 출력까지의 완전한 처리 과정을 다룬다.

목차

시스템 개요

목적

실시간 비디오 스트림에서 폭력 행동을 탐지하고, 관련 이벤트를 관리하여 즉각적인 알림과 로깅을 제공하는 시스템이다.

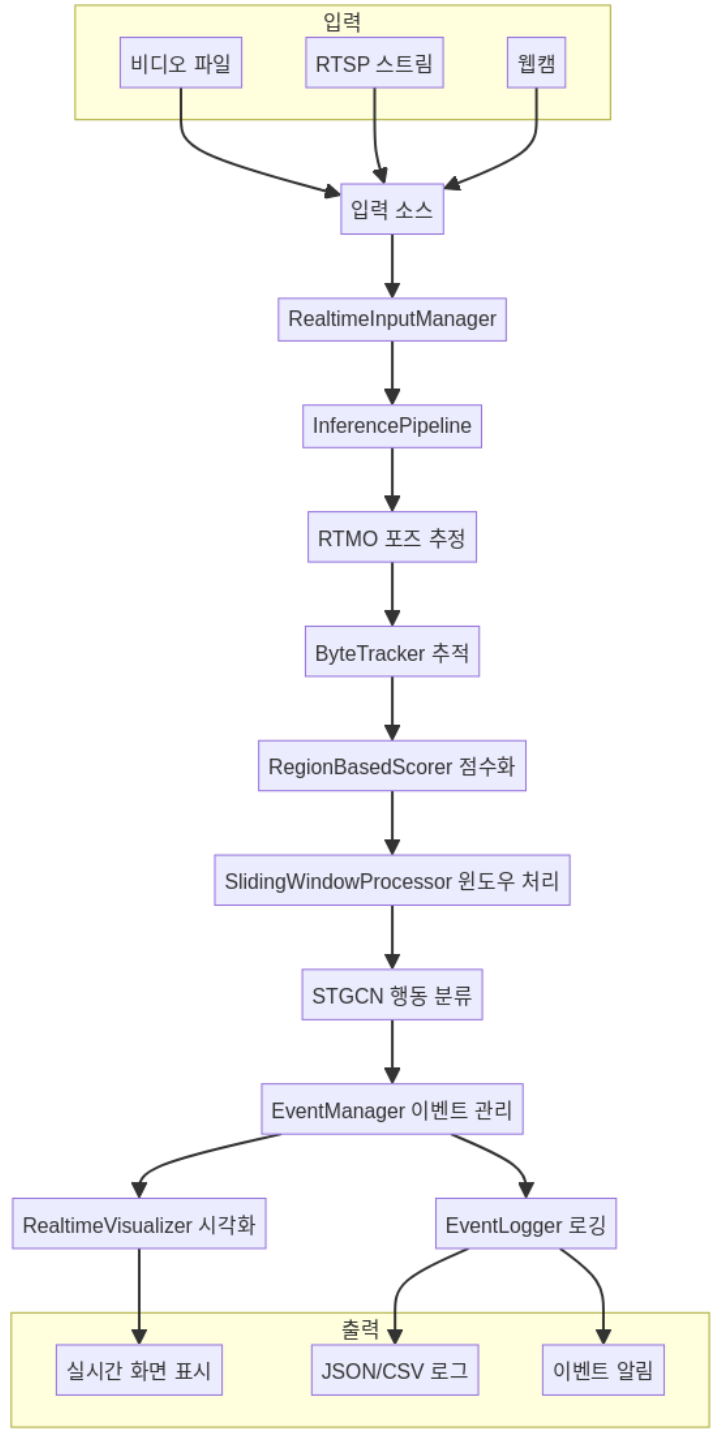
☒ 기술 스택

- **컴퓨터 비전**: OpenCV, RTMO (포즈 추정)
- **딥러닝**: PyTorch, ONNX, TensorRT
- **추론 엔진**: ST-GCN++ (행동 분류)
- **추적**: ByteTracker
- **이벤트 관리**: 커스텀 EventManager
- **시각화**: OpenCV GUI

성능 지표

- **실시간 처리**: 30-75 FPS (GPU 환경)
- **지연 시간**: 100-200ms (분류 포함)
- **정확도**: 90%+ (RWF-2000+ 데이터셋 기준)

전체 아키텍처


입력
비디오 파일
RTSP 스트림
웹캠
입력 소스
RealtimeInputManager
InferencePipeline
RTMO 포즈 추정
ByteTracker 추적
RegionBasedScorer 점수화
SlidingWindowProcessor 윈도우 처리
STGCN 행동 분류
EventManager 이벤트 관리
RealtimeVisualizer 시각화
EventLogger 로깅
출력
실시간 화면 표시
JSON/CSV 로그
이벤트 알림

핵심 컴포넌트

1. 입력 처리 계층

RealtimeInputManager

- **역할**: 다양한 입력 소스 통합 관리
- **지원 형식**: 비디오 파일, RTSP/RTMP 스트림, 웹캠
- **특징**: 비동기 프레임 캡처, 버퍼링, FPS 제어

2. 추론 처리 계층

InferencePipeline

- **역할**: 전체 추론 파이프라인 오케스트레이션
- **특징**: 모듈식 구조, 성능 추적, 비동기 처리

RTMO (Real-Time Multi-Person Pose Estimation)

- **역할**: 실시간 다중 인체 포즈 추정
- **지원 모드**: PyTorch, ONNX, TensorRT
- **출력**: 17개 키포인트 좌표 + 신뢰도

ByteTracker

- **역할**: 다중 객체 추적
- **특징**: ID 일관성 유지, 누락/재등장 처리

ST-GCN++ (Spatial-Temporal Graph Convolutional Networks)

- **역할**: 골격 기반 행동 분류
- **입력**: 시계열 포즈 시퀀스 (100 프레임)
- **출력**: Fight/NonFight 확률

3. 이벤트 관리 계층

EventManager

- **역할**: 폭력 이벤트 생명주기 관리
- **기능**: 발생/지속/해제 로직, 임계값 기반 판단

EventLogger

- **역할**: 이벤트 로깅 및 저장
- **지원 형식**: JSON, CSV

4. 시각화 계층

RealtimeVisualizer

- **역할**: 실시간 결과 시각화
- **기능**: 포즈 표시, 분류 결과, 이벤트 상태, 성능 지표

데이터 플로우

메인 처리 루프

```
while video_playing:
    # 1.
    frame = input_manager.get_frame()

    # 2.
    poses = pose_estimator.estimate(frame)

    # 3.
    tracked_poses = tracker.track(poses)

    # 4.
    scored_poses = scorer.score(tracked_poses)

    # 5.
    window_processor.add_frame(scored_poses)

    # 6.            ( )
    if window_processor.is_ready():
        window_data = window_processor.get_window()
        classification_queue.put(window_data)

    # 7.
    if classification_result_available:
        result = get_classification_result()
        event_data = event_manager.process_result(result)

        # 8.
        visualizer.update(frame, poses, result, event_data)

    # 9.
    visualizer.show()
```

데이터 구조 변환

```
(H×W×3)
```

```
↓
 (N×17×3) [N , 17   , (x,y,conf)]
↓
 (N×17×3 + track_id)
↓
  (M×17×3 + score) [M N,    ]
↓
  (100×M×17×2) [100  , M , 17   , (x,y)]
↓
(Fight  , NonFight  )
↓
  (event_type, confidence, timestamp)
```

주요 클래스 상세

InferencePipeline

```python
class InferencePipeline(BasePipeline):
    """             """

    def __init__(self, config: Dict[str, Any]):
        """


        Args:
            config:
        """

    def initialize_pipeline(self) -> bool:
        """           """

    def run_realtime_mode(self, input_source: str) -> bool:
        """       """

    def process_frame(self, frame: np.ndarray, frame_idx: int) -> Tuple[FramePoses, Dict]:
        """       """

    def _classification_worker(self):
        """         """

    def get_performance_stats(self) -> Dict[str, Any]:
        """      """
```

주요 메서드 시그니처

initialize_pipeline()

```
def initialize_pipeline(self) -> bool:
    """


    Returns:
        bool:

       :
    1.      (RTMO)
    2.    (ByteTracker)
    3.     (RegionBasedScorer)
    4.    (STGCN)
    5.
    6.
    """
```

process_frame()

```
def process_frame(self, frame: np.ndarray, frame_idx: int) -> Tuple[FramePoses, Dict]:
    """


    Args:
        frame:      (H×W×3)
        frame_idx:

    Returns:
        Tuple[FramePoses, Dict]: (      ,     )

       :
    1.     : frame → poses
    2.   : poses → tracked_poses
    3.    : tracked_poses → scored_poses
    4.       : scored_poses → window_buffer
    5.
    """
```

EventManager

```
class EventManager:
    """      """

    def __init__(self, config: EventConfig):
        """      """
```

```
    def process_classification_result(self, result: Dict[str, Any]) -> Optional[EventData]:
        """            """

    def add_event_callback(self, event_type: EventType, callback: Callable):
        """      """

    def get_current_status(self) -> Dict[str, Any]:
        """        """
```

이벤트 처리 로직

process_classification_result()

```
def process_classification_result(self, result: Dict[str, Any]) -> Optional[EventData]:
    """


    Args:
        result: {
            'window_id': int,
            'prediction': str,  # 'violence' or 'normal'
            'confidence': float,
            'timestamp': float,
            'probabilities': List[float]
        }

    Returns:
        Optional[EventData]:       (   None)

       :
    1.      (alert_threshold: 0.7)
    2.
    3.
    4.
    5.
    """
```

RealtimeInputManager

```
class RealtimeInputManager:
    """       """

    def __init__(self, input_source: Union[str, int], buffer_size: int = 10,
                 target_fps: Optional[int] = None, frame_skip: int = 0):
        """       """
```

```
    def start(self) -> bool:
        """      """

    def get_frame(self) -> Optional[Tuple[np.ndarray, int]]:
        """       """

    def stop(self):
        """      """

    def get_video_info(self) -> Dict[str, Any]:
        """      """
```

지원하는 입력 형식

```
#
input_source = "/path/to/video.mp4"

# RTSP
input_source = "rtsp://camera_ip:554/stream"

#
input_source = 0  #   "/dev/video0"

# RTMP
input_source = "rtmp://server/live/stream"
```

RealtimeVisualizer

```
class RealtimeVisualizer:
    """       """

    def __init__(self, window_name: str = "Violence Detection",
                 display_width: int = 1280, display_height: int = 720,
                 fps_limit: int = 30, confidence_threshold: float = 0.4):
        """     """

    def show_frame(self, frame: np.ndarray, poses: Optional[FramePoses] = None,
                   classification: Optional[Dict[str, Any]] = None,
                   additional_info: Optional[Dict[str, Any]] = None,
                   overlay_data: Optional[Dict[str, Any]] = None) -> bool:
        """    """

    def update_event_history(self, event_data: Dict[str, Any]):
        """        """

    def update_classification_history(self, classification: Dict[str, Any]):
```

```
    """          """
```

API 참조

설정 구조

메인 설정 (config.yaml)

```yaml
mode: inference.realtime

models:
  pose_estimation:
    inference_mode: onnx  # pth, onnx, tensorrt
    onnx:
      model_path: /path/to/rtmo.onnx
      device: cuda:0
      score_threshold: 0.3
      input_size: [640, 640]

  action_classification:
    model_name: stgcn
    checkpoint_path: /path/to/stgcn.pth
    confidence_threshold: 0.4  #
    window_size: 100
    device: cuda:0

  tracking:
    tracker_name: bytetrack
    track_thresh: 0.4
    track_buffer: 50
    match_thresh: 0.8

events:
  alert_threshold: 0.7        #
  normal_threshold: 0.5       #
  min_consecutive_detections: 3
  min_consecutive_normal: 5
  min_event_duration: 2.0
  max_event_duration: 300.0
  cooldown_duration: 10.0
  save_event_log: true
  event_log_format: json
  event_log_path: output/event_logs

files:
```

```
  video:
    input_source: /path/to/video.mp4  #   RTSP URL
    output_path: output/result_video.mp4
    save_output: false

performance:
  target_fps: 30
  max_queue_size: 200
  processing_mode: realtime
```

EventConfig 클래스

```
@dataclass
class EventConfig:
    """      """
    alert_threshold: float = 0.7          #
    min_consecutive_detections: int = 3   #
    normal_threshold: float = 0.5         #
    min_consecutive_normal: int = 5       #
    min_event_duration: float = 2.0       #          ( )
    max_event_duration: float = 300.0     #          ( )
    cooldown_duration: float = 10.0       #        ( )
    enable_ongoing_alerts: bool = True    #
    ongoing_alert_interval: float = 30.0  #         ( )
    save_event_log: bool = True          #
    event_log_format: str = "json"       #     (json/csv)
    event_log_path: str = "output/event_logs"  #
```

데이터 구조

FramePoses

```
@dataclass
class FramePoses:
    """          """
    persons: List[PersonPose]
    frame_idx: int
    timestamp: float
    video_info: Dict[str, Any]
```

PersonPose

```
@dataclass
class PersonPose:
    """        """
    keypoints: np.ndarray    # (17, 3) [x, y, confidence]
    bbox: Optional[np.ndarray]  # [x1, y1, x2, y2]
```

```
    track_id: Optional[int]
    score: float
    detection_confidence: float
```

EventData

```
@dataclass
class EventData:
    """    """
    event_type: EventType
    timestamp: float
    window_id: int
    confidence: float
    duration: Optional[float] = None
    additional_info: Optional[Dict[str, Any]] = None

    def to_dict(self) -> Dict[str, Any]:
        """     """
```

EventType

```
class EventType(Enum):
    """    """
    VIOLENCE_START = "violence_start"    #
    VIOLENCE_END = "violence_end"        #
    VIOLENCE_ONGOING = "violence_ongoing" #
    NORMAL = "normal"                    #
```

성능 지표

PerformanceTracker

```
class PerformanceTracker:
    """      """

    def get_stage_fps(self) -> Dict[str, float]:
        """   FPS   """
        return {
            'pose_estimation': float,
            'tracking': float,
            'scoring': float,
            'classification': float,
            'overall': float
        }

    def get_processing_time_stats(self) -> Dict[str, float]:
        """     """
```

```
        return {
            'avg_processing_time': float,
            'max_processing_time': float,
            'min_processing_time': float
        }
```

성능 최적화

최적화 전략

1. GPU 메모리 최적화

```yaml
# ONNX/TensorRT
pose_estimation:
  inference_mode: onnx  #   tensorrt

#
batch_size: 1  #
```

2. 비동기 처리

```
#
classification_queue = Queue(maxsize=10)
classification_thread = threading.Thread(target=self._classification_worker)
```

3. 프레임 스킵핑

```yaml
#  FPS
frame_skip: 2  # 2    1
```

4. 윈도우 슬라이딩 최적화

```yaml
#
window_size: 100
window_stride: 50  # 50%
```

성능 모니터링

실시간 FPS 추적

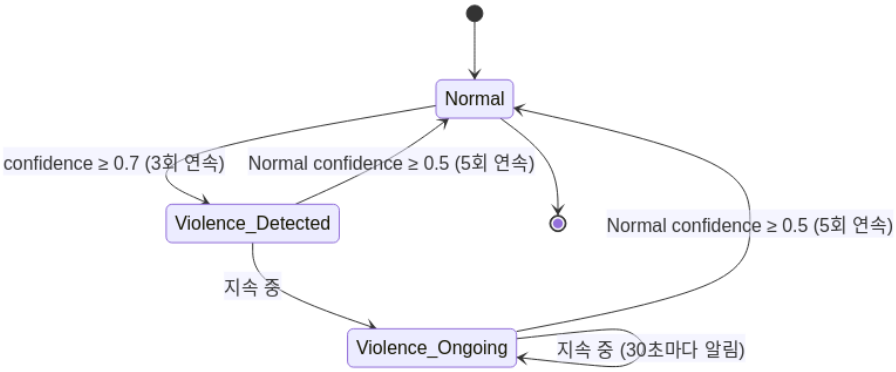
- **포즈 추정 FPS**: RTMO 추론 속도
- **추적 FPS**: ByteTracker 처리 속도
- **분류 FPS**: ST-GCN++ 추론 속도
- **전체 FPS**: 엔드투엔드 처리 속도

메모리 사용량 모니터링

- **GPU 메모리**: 모델 로딩 + 추론 버퍼
- **CPU 메모리**: 프레임 버퍼 + 윈도우 데이터
- **큐 사용률**: 분류 큐 점유율

이벤트 관리

이벤트 생명주기



임계값 설정

confidence_threshold (0.4)

- **용도**: 시각화 색상 결정
- **로직**: Fight/NonFight 확률이 0.4 이상이고 상대방보다 높으면 해당 색상 표시
- **영향**: 화면 표시 색상만 결정

alert_threshold (0.7)

- **용도**: 이벤트 발생 판단
- **로직**: Violence 예측 확률이 0.7 이상일 때 이벤트 시작
- **영향**: 실제 알림 및 로깅

normal_threshold (0.5)

- **용도**: 이벤트 해제 판단
- **로직**: Normal 예측 확률이 0.5 이상일 때 이벤트 종료
- **영향**: 이벤트 종료 및 복구

이벤트 로깅

JSON 형식

```
{
  "event_id": "evt_20241220_143022_001",
  "event_type": "violence_start",
  "timestamp": 1703058622.123,
  "window_id": 15,
  "confidence": 0.847,
  "duration": null,
  "session_id": "session_20241220_143000",
  "additional_info": {
    "probabilities": [0.153, 0.847],
    "frame_number": 1500
  }
}
```

CSV 형식

```
timestamp,event_type,window_id,confidence,duration,session_id
1703058622.123,violence_start,15,0.847,,session_20241220_143000
1703058634.456,violence_end,23,0.523,12.333,session_20241220_143000
```

이벤트 콜백

```python
#
def on_violence_start(event_data: EventData):
    """       """
    print(f"[ALERT] Violence detected! Confidence: {event_data.confidence:.3f}")
    #    ,

def on_violence_end(event_data: EventData):
    """       """
    print(f"[INFO] Violence ended. Duration: {event_data.duration:.1f}s")
    #    ,

#
event_manager.add_event_callback(EventType.VIOLENCE_START, on_violence_start)
event_manager.add_event_callback(EventType.VIOLENCE_END, on_violence_end)
```

실행 가이드

기본 실행

```
#
python main.py --mode inference.realtime

#
```

```
python main.py --config custom_config.yaml --mode inference.realtime

#
python main.py --mode inference.realtime --log-level DEBUG
```

고급 설정

RTSP 스트림 처리

```yaml
files:
  video:
    input_source: "rtsp://admin:password@192.168.1.100:554/stream1"
    buffer_size: 10
    target_fps: 25
```

TensorRT 가속

```yaml
models:
  pose_estimation:
    inference_mode: tensorrt
    tensorrt:
      model_path: /path/to/rtmo.engine
      device: cuda:0
```

이벤트 알림 강화

```yaml
events:
  alert_threshold: 0.6        #
  min_consecutive_detections: 2  #
  enable_ongoing_alerts: true
  ongoing_alert_interval: 15.0   # 15
```

문제 해결

일반적인 문제

1. GPU 메모리 부족

```yaml
#   :       ,
models:
  pose_estimation:
    inference_mode: onnx  # PyTorch → ONNX
```

2. 실시간 처리 지연

```yaml
#   :     ,
performance:
```

```
  frame_skip: 1
  input_resolution: [640, 640]  #
```

3. 이벤트 과민 반응

```
#   :
events:
  alert_threshold: 0.8           #
  min_consecutive_detections: 5  #
```

성능 튜닝

CPU vs GPU 밸런싱

- **포즈 추정**: GPU 중심 (ONNX/TensorRT)
- **추적**: CPU 효율적
- **분류**: GPU 가속 필요
- **시각화**: CPU 처리

메모리 최적화

- **윈도우 버퍼**: 최소 크기 유지
- **프레임 큐**: 적절한 버퍼 사이즈
- **결과 캐시**: 주기적 정리

확장성

모듈 확장

새로운 포즈 추정기 추가

```
from pose_estimation.base import BasePoseEstimator

class CustomPoseEstimator(BasePoseEstimator):
    def estimate(self, frame: np.ndarray) -> FramePoses:
        #
        pass

#
ModuleFactory.register_pose_estimator(
    name='custom_pose',
    estimator_class=CustomPoseEstimator,
    default_config={'param': 'value'}
)
```

새로운 이벤트 타입 추가

```
class EventType(Enum):
    VIOLENCE_START = "violence_start"
    VIOLENCE_END = "violence_end"
    VIOLENCE_ONGOING = "violence_ongoing"
    NORMAL = "normal"
    CUSTOM_EVENT = "custom_event"  #
```

분산 처리

멀티 카메라 지원

```
#
cameras = [
    "rtsp://camera1/stream",
    "rtsp://camera2/stream",
    "rtsp://camera3/stream"
]

for camera_id, source in enumerate(cameras):
    pipeline = InferencePipeline(config)
    pipeline.run_realtime_mode(source)
```

결론

본 실시간 추론 시스템은 모듈식 아키텍처와 이벤트 중심 설계를 통해 확장 가능하고 유지보수가 용이한 폭력 탐지 솔루션을 제공한다. 각 컴포넌트는 독립적으로 최적화될 수 있으며, 다양한 입력 소스와 배포 환경을 지원한다.

주요 장점

- **실시간 성능**: 30-75 FPS 처리 능력
- **모듈식 구조**: 각 컴포넌트 독립적 교체 가능
- **이벤트 기반**: 실시간 알림 및 로깅
- **다중 백엔드**: PyTorch, ONNX, TensorRT 지원
- **확장성**: 새로운 모델 및 기능 쉽게 추가

향후 개선 방향

- **엣지 디바이스**: Jetson, 모바일 디바이스 지원
- **클라우드 연동**: AWS, Azure 클라우드 서비스 통합
- **고급 분석**: 행동 패턴 분석, 예측 모델링
- **다중 모달**: 오디오, 텍스트 등 추가 모달리티 지원
