# 5. Capstone Exercise: Multi-Input/Multi-Output (MIMO) 파이프라인

## 프로젝트 개요

NVIDIA 교육 과정의 캡스톤 프로젝트로, 두 가지 유형의 데이터 스트림(GPU 텔레메트리 / 네트워크 PCAP)을 동시에 처리하고, 데이터 유형에 따라 서로 다른 처리 브랜치로 라우팅하여 암호화폐 채굴 활동을 탐지하는 **비선형(Non-Linear) 파이프라인**을 완성했습니다.

## ⚙️ 사용 기술 및 역할

* **주요 기술:** NVIDIA Morpheus Pipeline, RouterStage, FileSourceStage (멀티 소스), 커스텀 메타데이터 태깅 스테이지.
* **핵심 역할:** Pipeline 객체를 사용하여 비선형 그래프를 정의하고, 커스텀 스테이지를 통해 메시지에 라우팅 키를 추가하여 RouterStage가 데이터 스트림을 NVSMI(GPU 텔레메트리) 브랜치와 PCAP 브랜치로 분기하도록 논리를 구축했습니다.

## 🛠️ 파이프라인 구성 (비선형 MIMO)

| **구성 요소** | **역할** | **분기 경로** |
| --- | --- | --- |
| **NVSMI Source / PCAP Source** | 2개의 독립적인 소스(GPU 로그, 네트워크 로그)에서 데이터 로드 | - |
| **Metadata Tagger** | 각 메시지에 **라우팅 키**(route\_a 또는 route\_b)를 메타데이터로 추가 | - |
| **RouterStage** | 메시지에 태그된 메타데이터를 확인하여 해당 브랜치로 데이터를 분기 | route\_a (NVSMI) 또는 route\_b (PCAP) |
| **NVSMI Processing Branch** | 전처리 -> abp-nvsmi-xgb Triton 추론 -> 분류 -> 저장 (NVSMI 전용) | - |
| **PCAP Processing Branch** | 전처리 -> abp-pcap-xgb Triton 추론 -> 분류 -> 저장 (PCAP 전용) | - |
| **MonitorStage** | 두 브랜치의 최종 출력을 하나로 모아(add\_edge로 연결) 전체 처리량을 통합 모니터링 | - |

## 💡 핵심 성과 및 역량

1. **비선형 파이프라인 구축:** LinearPipeline 대신 Pipeline 객체를 사용하여 데이터 스트림을 병렬 처리하고 분기/통합하는 복잡한 데이터 흐름을 성공적으로 구현했습니다.
2. **동적 라우팅 구현:** 메시지의 메타데이터를 기반으로 실시간으로 처리 경로를 결정하는 RouterStage의 key\_fn 논리를 구현하여, 서로 다른 포맷과 로직을 요구하는 데이터를 단일 파이프라인에서 처리했습니다.
3. **다중 추론 통합:** 서로 다른 두 개의 ML 모델(nvsmi-xgb, pcap-xgb)을 Triton Inference Server를 통해 각각의 브랜치에 성공적으로 통합하여 고성능 다중 탐지 시스템을 구축했습니다.

## 📌 주요 코드 로직 (파이프라인 연결부)

# 1. 커스텀 스테이지 정의 (ControlMessage에 'route' 메타데이터 태깅)  
# RouteMetadataTaggerStage는 ControlMessage.set\_metadata("route", self.\_branch\_key) 실행  
  
# 2. 파이프라인 정의 (비선형 그래프 사용)  
pipeline = Pipeline(config)  
branch\_keys = ["route\_a", "route\_b"]  
  
# 3. 입력 브랜치 정의 및 라우터 스테이지로 연결  
nvsmi\_tagger = pipeline.add\_stage(RouteMetadataTaggerStage(config, branch\_key="route\_a"))  
pcap\_tagger = pipeline.add\_stage(RouteMetadataTaggerStage(config, branch\_key="route\_b"))  
  
# 4. RouterStage 정의 및 키 함수 설정  
router = pipeline.add\_stage(RouterStage(config, keys=branch\_keys, key\_fn=router\_key\_fn))  
  
# 5. RouterStage에 두 입력 브랜치 연결  
pipeline.add\_edge(nvsmi\_tagger, router)  
pipeline.add\_edge(pcap\_tagger, router)  
  
# 6. 출력 브랜치 정의 및 연결 (Router Out)  
# Router의 출력 포트 인덱스를 사용하여 분기  
pipeline.add\_edge(router.output\_ports[0], nvsmi\_preprocess)  
pipeline.add\_edge(router.output\_ports[1], pcap\_preprocess)  
  
# 7. 최종 출력 통합 (Interleave outputs in a unified monitoring stage)  
# (중간 처리 생략)  
pipeline.add\_edge(nvsmi\_sink, pipeline\_monitor)  
pipeline.add\_edge(pcap\_sink, pipeline\_monitor)  
  
# pipeline.build() 실행 및 테스트