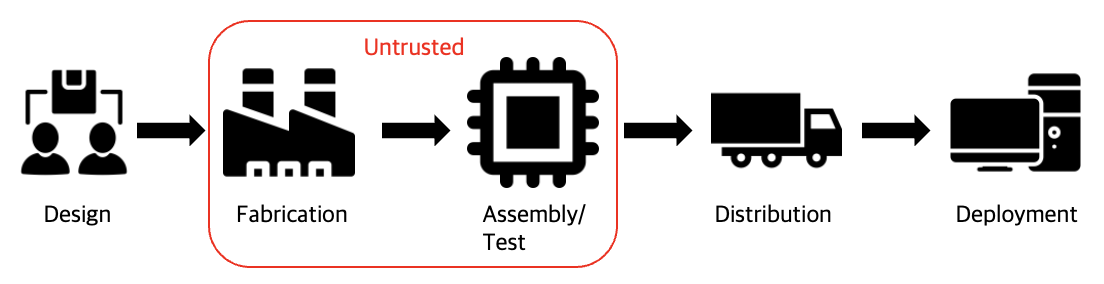
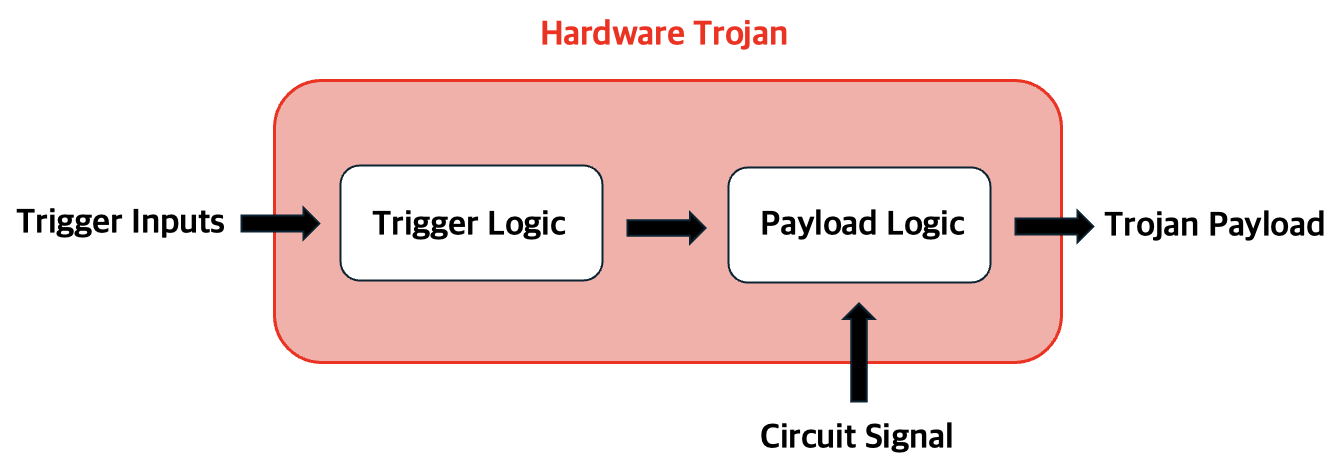
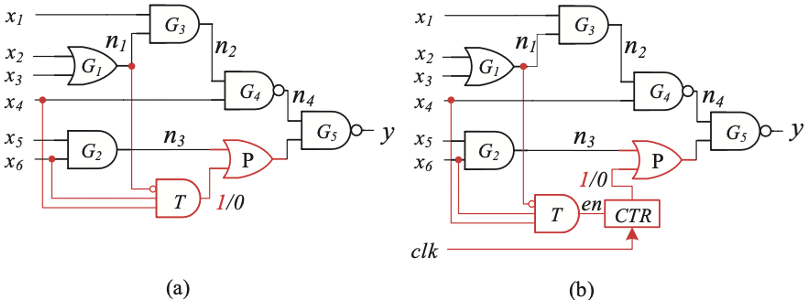
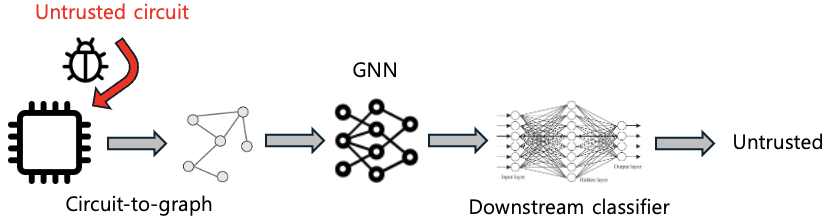
**abstruct:**

**본 논문은 현대 하드웨어 보안 위협 중 하나인 트로이 목마(Trojan)에 대한 효율적인 탐지 및 대응 방법을 제안한다. 방대한 Netlist 데이터를 효과적으로 분석하기 위해, 데이터의 표현 방식과 증강(augmentation) 기법을 탐구하며, 기존 Trojan 탐지에 주로 사용되는 GCN(Graph Convolutional Network)과 차별화된 접근을 시도한다. 특히, SAM optimizer와 attention 기법을 사용하지 않고, 단순한 residual layer를 적용하여 대규모 입력 데이터에 대응하면서도 모델의 성능을 강화하는 방법을 제안한다.**

1. **서론**  
   IT 기술의 빠른 발전으로 인해 다양한 분야에서 IT 기술이 활용되고 있으며, 다양하고 은밀한 보안 위협 또한 증가하고 있다. 현대의 보안 위협은 더욱 다양하고 복잡한 형태로 발전하였으며, 특히, 하드웨어 수준에서의 공격은 기존의 소프트웨어 중심의 보안 접근 방식으로는 충분히 대응하기 어려운 새로운 도전과제로 부상하고 있다. 하드웨어 트로이목마는 이러한 새로운 위협 중 하나로, 전자 회로 내부에 은밀하게 삽입되어 특정 조건이 충족될 때만 활성화되어 비정상적인 동작을 유발시키는 보안 위협이다. 이는 정보를 유출하거나, 시스템을 파괴하거나, 성능을 저하시키는 등, 다양한 목적으로 활용되며 신뢰성과 보안에 대한 심각한 위협으로 인식되고 있다.   
   컴퓨터가 점점 복잡한 하드웨어 구조를 가짐에 따라, 세분화되고 전문화된 생산 공정은 하드웨어 트로이목마의 잠입에 더욱 유리한 환경을 제공한다. 이러한 공정은 대부분의 경우, 전 세계 여러 국가에 걸쳐 아웃소싱되어 진행되며, 이는 다양한 공급망 단계에서의 보안 취약점을 만들어내고, 외부 공격자들에게 악의적인 행위를 할 수 있는 기회를 제공한다.  
     
   이러한 맥락에서, 하드웨어 트로이목마를 효과적으로 탐지하고 대응하는 것은 매우 중요한 연구 분야로 주목받고 있다. 기존에는 무작위로 입력값을 대입하여 시스템이 예외 상황을 처리하는 것을 관찰하는 Fuzzing을 활용한 연구도 진행되었고[[1]](#endnote-1), Regression, Reinforcement Learning과 같은 머신러닝 알고리즘을 활용하여 미세한 하드웨어 트로이목마를 감지하는 연구도 진행되었다.[[2]](#endnote-2) 최근에는 그래프 신경망(Graph Neural Network, GNN) 기술이 하드웨어 트로이목마 탐지 연구에 도입되기 시작했다. 그래프 신경망은 그래프로 표현되는 데이터를 처리하기 위한 신경망으로, 노드와 그 사이의 관계를 학습함으로써 복잡한 그래프 구조를 효과적으로 분석할 수 있다. 그래프 신경망은 회로의 구조적 특성을 그래프로 표현하고, 이 그래프 데이터에서 트로이목마가 삽입된 미묘한 변화를 감지하여 복잡한 회로 구조 속에서도 하드웨어 트로이목마의 존재 여부를 판별한다.  
     
   그러나 그래프 신경망을 이용한 하드웨어 트로이목마 탐지 기술은, 회로의 크기가 커질수록 탐지 과정의 복잡도가 급격히 증가하는 한계점을 드러냈다. 따라서, 본 연구는 기존의 그래프 신경망 기반 하드웨어 트로이목마 탐지 연구의 한계를 극복하고자 방대한 large scale Netlist 데이터에 대해 GCN을 통한 효율적인 하드웨어 트로이목마 탐지의 새로운 접근 방식과 분석을 제안하고 기존의 연구들과 하드웨어 트로이목마 탐지 성능을 비교한다. 본 논문은 다음과 같이 구성된다. 2장에서는 하드웨어 트로이목마와 그래프 신경망에 대해서 서술한다. 3장에서는 관련된 기존 연구들에 대해서 서술한다. 4장에서는 본 연구의 핵심 기술에 대해 소개하고 구현 과정 대하여 설명한다. 5장에서는 실험 결과를 기존의 연구와 비교하고 마지막으로 6장에서 결론을 맺는다.
2. **백그라운드**
3. **하드웨어 트로이목마**  
   하드웨어 트로이목마는 전자 회로 내부에 은밀하게 삽입되어, 특정 조건이 충족되거나 특정 신호가 감지될 때 활성화되어 악의적인 동작을 수행하는 정교한 보안 위협이다. 이러한 동작에는 직접적인 데이터 유출은 물론, 전력 사용 패턴의 미묘한 변화를 통해 정보를 유출하는 사이드 채널 공격, 시스템의 정상적인 기능 방해, 성능을 저하시키는 동작까지 포함한다. 이러한 하드웨어 트로이목마는 소프트웨어 기반의 공격과 달리 은밀성과 회로 내부에 삽입되는 이유로 외부에서 감지하고 제거하는 것이 매우 어렵다는 특징을 가진다.  
   집적 회로(ICs)의 설계 및 생산 과정은, 생산 공장을 건설하고 유지하기 데 드는 막대한 비용으로 인해 아웃소싱을 통해 생산이 이루어지고 있다. 이런 글로벌화된 반도체 공급망에는 신뢰할 수 없는 다양한 엔티티가 포함되어 있기 때문에 신뢰할 수 없는 IC 생산 현장에서 하드웨어 트로이 목마가 삽입되는 위험이 존재한다. 신뢰할 수 없는 아웃소싱에서, 공격자는 하드웨어 트로이 목마를 활성화하여 보안 기능을 우회하여, 설계를 변경하거나 악의적인 행동을 하는 회로를 추가하여 시스템을 제어하거나 중요한 정보를 유출할 수 있다.[[3]](#endnote-3)  
     
     
   하드웨어 트로이목마는 전형적으로 크게 두 부분으로 구성된다. 트리거는 특정 조건에서 트로이목마를 작동시키는 회로이며[[4]](#endnote-4), 이런 트리거가 작동하는 이벤트는 디자인 시뮬레이션 또는 테스트 중에 발생할 가능성이 낮지만, 장시간의 실제 작동중에서 발생할 수 있다. 트리거 조건에 기반하여, 하드웨어 트로이목마는 아날로그 및 디지털 트로이목마로 분류될 수 있다. 아날로그 하드웨어 트로이목마는 온도, 지연 또는 장치 노화 효과와 같은 아날로그 조건에 의해 활성화되는 반면, 디지털 하드웨어 트로이목마는 일부 Bool 논리 함수에 의해 트리거된다.[[5]](#endnote-5)  
   페이로드는 트로이목마의 악성 기능을 수행하는 회로로[[6]](#endnote-6), 전송된 무선 신호 또는 직렬 데이터 포트 인터페이스를 통해 신뢰 위반을 일으킬 수 있다. 또한, 전력 트레이스 또는 열 방사선을 통해 정보가 유출되는 사이드 채널 공격이나 denial-of-service(DoS) 공격과 같은 회로 기능의 비인가된 변경일 수 있다.[[7]](#endnote-7)  
     
   이 논문의 실험에서 사용된 하드웨어 트로이목마는 Combinational 하드웨어 트로이목마와 Sequential 하드웨어 트로이목마로 두 가지이다.   
   Combinational 하드웨어 트로이목마는 회로의 기본 입력 또는 내부 노드로부터 가져온 트리거와 일단 트리거가 실행되면 활성화될 수 있는 페이로드로 구성된다. (a)는 페이로드(P) OR 게이트와 함께 Combinational 하드웨어 트로이목마가 주입된 회로의 예시를 나타낸다. 활성화 시 트리거(AND 게이트)의 출력은 로직 1이 되며, 그렇지 않으면 항상 0이다. 이러한 Combinational 하드웨어 트로이목마는 트리거가 조건을 만족하여 활성화되어 1을 전달하면 페이로드가 작동하게 된다.  
   Sequential 하드웨어 트로이목마는 특정한 입력 패턴이 발생하거나 트리거될 때, 일정 기간 후에 페이로드를 전달한다. (b)는 트리거가 AND 게이트와 카운터(CTR)로 구성된 Sequential 트로이 목마를 나타낸다. 여기서 트리거 매커니즘은 두 가지 종류로 나뉜다. (1) 트리거 조건이 충족될 때마다(en = 1) 카운터가 증가하거나, (2) 트리거 조건이 나타나면 카운터가 활성화되고, 이후 클럭마다 증가하는 방식이다. (1)의 경우, 카운터가 최대 카운트에 도달할 때만 페이로드가 전달되어, IC의 테스트 또는 정상 동작 중에 특정 테스트 패턴 또는 입력이 연속적으로 여러 번 발생할 가능성이 매우 낮기 때문에 감지를 더욱 어렵게 만든다.[[8]](#endnote-8)
4. **그래프 신경망**

그래프 신경망(Graph Neural Networks, GNN)은 그래프 구조화된 데이터를 분석하고 해석하기 위한 첨단 신경망 아키텍처이다. 그래프 신경망은 노드(Node)와 엣지(Edge)로 구성된 그래프 형태의 데이터를 처리함으로써, 복잡한 관계와 상호작용이 내재된 데이터의 깊이 있는 분석이 가능하다. 

하드웨어 트로이목마 탐지에 그래프 신경망을 적용하는 것은 하드웨어 디자인에서 추출된 Data Flow Graphs(DFG)를 기반으로 한다. 이러한 DFG는 회로의 동작을 그래프로 표현한 것으로, 그래프 신경망은 이러한 그래프를 입력으로 받아 하드웨어 트로이목마의 특징을 학습한다.

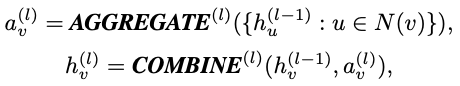
이를 통해 RTL(Register Transfer Level)과 gate-level netlist에 대한 하드웨어 트로이목마를 효과적으로 탐지할 수 있다.

DFG를 기반으로 하는 하드웨어 트로이목마 탐지는 크게 RTL code를 기반으로 하는 방법과 netlist를 기반으로 하는 방법이 존재한다. 둘은 구조, 추상화 수준 및 코드 크기(신호 및 연산 수) 측면에서 매우 다르다.   
RTL code는 하드웨어 디자인의 high level 표현이며, RTL에서의 하드웨어 트로이목마는 보통 설계된 하드웨어의 기능적인 변조를 의미한다. 비정상적인 동작을 하도록 설계된 코드 또는 회로를 삽입되어 시스템 보안을 침해한다. RTL에서의 DFG를 이용한 탐지는 정상적인 동작을 진행하는 모델과 비정상적인 동작을 진행하는 모델을 생성하여 각각의 DFG를 비교하여 차이를 분석해 탐지하는 방법이다.   
Gate-level netlist에서의 DFG는 논리 데이트와 레지스터 등의 하드웨어 구성 요소의 데이터 흐름을 나타낸다. Netlist에서 하드웨어 트로이목마를 찾기 위한 그래프 생성 파이프라인과 GNN 모델을 생성하여 하드웨어 트로이목마의 여부를 판별한다.

Rozhin Yasaei등은[[9]](#endnote-9) 하드웨어 디자인이 하드웨어 트로이목마에 감염되었는지를 판단하기 위한 함수를 근사화하기 위하여 다음의 세가지 단계로 구성된 방법론을 제시하였다. 가장 먼저, 자동화된 DFG 생성 파이프라인을 통해 하드웨어 디자인에서 DFG를 추출한다. 다음으로, 추출된 DFG가 그래프 학습 및 특징 추출을 위한 graph convolution layer 등을 포함하는 GNN 프레임워크를 통과하고, 이 과정에서 그래프의 중요 특징을 기반으로 벡터화된 그래프 임베딩을 생성한다. 마지막으로, Multi-Layer Perceptron(MLP)을 사용하여 그래프 임베딩에 대한 분류를 수행하고 최종적으로 트로이목마 감염 여부를 판단한다. 각 단계의 세부사항은 다음과 같다.  
  
**a.** **Graph Convolution:**

GNN에 대한 입력은 그래프 로 나타내며, 여기서 는 방향성이 있는 간선의 집합이고, 는 정점의 집합이다. 간선은 adjacency matrix 로 표현되며, 각 노드는 노드 속성을 특정하는 특성 벡터 를 구현한다.

Graph Convolution은 메시지 전파 단계와 읽기 단계로 두 단계로 나뉜다. 메시지 전파 단계는 다음과 같은 두 가지 하위 함수를 포함한다.



여기서 는 노드 에 연결된 노드 집합이며, AGGREGATE는 이웃 노드의 특징 벡터를 수집하여 레이어 에 대한 집계된 특징 벡터를 생성한다. COMBINE은 이전 노드 특성 를 와 결합하여 다음 특징 벡터 를 생성한다. 메시지 전파는 미리 결정된 수의 반복에 대해 수행된다. AGGREGATE 및 COMBINE 함수는 모든 ∈ V 에 대한 그래프 컨볼루션 벡터 에 의해 수행되며, 이를 노드 임베딩 행렬이라고 한다. 메시지 전파의 각 반복 에 대한 노드 임베딩은 다음과 같이 업데이트 된다.

여기서 은 GCN(Graph Convolution Network) 레이어에서 사용되는 학습 가능한 가중치이다. 는 이웃 노드의 특징 벡터를 집계하는 데 사용되는 의 adjacency matrix 이며, 는 이전에 계산된 특성이 현재 반복에서도 고려되도록 하기 위해 자체 루프 연결을 추가하는 identity matrix이다. 이 자체 루프는 누적된 메시지가 이전 노드 특징 벡터와 결합되는 COMBINE 함수와 유사하게 작용한다. 은 을 정규화하는 데 사용되는 diagonal degree matrix이고, σ(.)는 Rectified Linear Unit(ReLU)과 같은 활성화 함수이다.

**b. Graph Embedding Generation:**   
Read-out phase에서 graph pooling layer인 에서 획득한 노드 임베딩을 집계하고 DFG 의 그래프 임베딩 라고 부르는 벡터인 graph-level 특성을 추출한다.

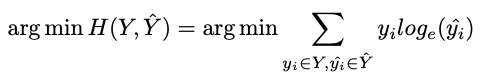
여기서, READOUT 연산은 각각 sum-pooling, mean-pooling 또는 max-pooling으로 표시되는 모든 노드 임베딩에 걸쳐 각 특성 차원의 합, 평균 또는 최대값일 수 있다.

**c. Multi-Layer Perceptron Classifier:**

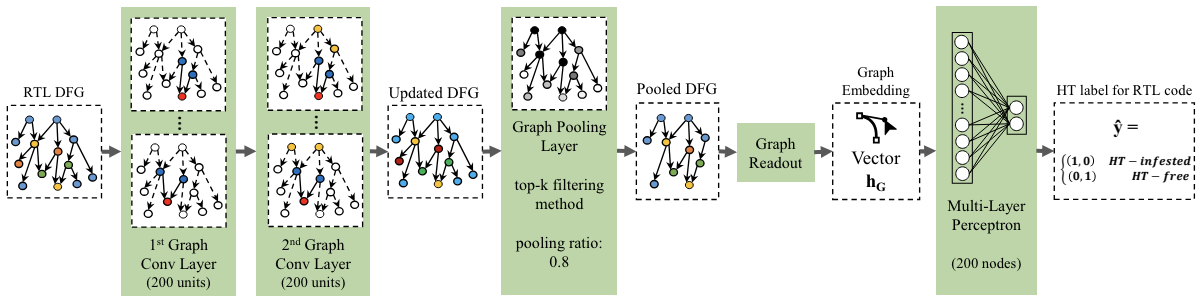
임베딩 벡터 를 MLP 계층과 Softmax 활성화 함수로 추가 처리하여 다음과 같은 최종 예측을 생성한다.



이 계층은 에 사용되는 숨겨진 유닛의 수를 줄이고 두 클래스(HT-infested 또는 HT-free)의 확률을 나타내는 2차원 출력을 생성한다. 마지막으로 의 예측값을 Softmax 함수를 사용하여 정규화하고 예측값이 더 높은 클래스를 검출 결과로 선택한다.

모델을 훈련하기 위해 다음과 같이 설명되는 실측 레이블 와 예측 레이블 사이의 로 표시되는 cross-entropy 손실 함수를 계산한다.

모델은 cross-entropy 손실 함수를 최소화하기 위해 경사 하강법을 사용하여 반복적인 프로세스를 통해 훈련된다.

[[10]](#endnote-10)

그래프 신경망을 사용한 하드웨어 트로이목마 탐지의 주요 장점 중 하나는 새로운 하드웨어 트로이목마가 발견되더라도 이를 쉽게 감지할 수 있다는 것이다. 이전 작업과 같이 새로운 속성을 정의하거나 추가 기능 엔지니어링을 도입할 필요 없이 모델을 쉽게 업데이트하여 새로운 위협에 대응할 수 있다.

1. **related work**

기존 연구들 중 그래프 신경망을 제외한 방법의 하드웨어 트로이목마 검출은 크게 고전적 방식의 퍼징(Fuzzing) 기반 검출 방법과 기계 학습을 통한 검출 방법 두 가지로 나뉜다. 퍼징은 무작위로 생성되는 테스트 패턴(Test Pattern)을 통해 하드웨어 디자인에서 설계 결함, 하드웨어 트로이목마 등의 취약점을 찾는 검출 방법으로, 충족 가능성 문제(Satisfiability Problem, SAT Problem)로 귀결되는 알고리즘 구축을 통해 하드웨어 트로이목마를 검출한다. 퍼징 방법은 충족성 문제 특성상 시간이 오래 소모되는 단점이 존재하나 탐지 정확도가 높다는 장점이 존재한다. 기존 연구들은 휴리스틱 방법을 통해 정적인 환경에서 특정 횟수 이내 하드웨어 트로이목마 검출을 수행할 수 있도록 진행되었으며, 최근 연구에서는 기존 방향의 퍼징 검출기에 기계 학습 및 심층 학습을 통해 검출기의 성능을 향상시키는 모델을 제안한다. 기계 학습을 통한 검출 방식은 하드웨어 트로이목마가 포함된 하드웨어 디자인에서 추출한 피처를 대상으로 결정 트리, 다층 퍼셉트론 등 사전에 선정된 하드웨어 트로이목마와 연관된 피처를 기계 학습을 통해 학습, 추론하는 구조를 지닌다. 이 방식은 퍼징에 비해 빠른 속도를 보이나 탐지 정확성이 상대적으로 떨어지는 모습을 보인다.

R.S. Chakraborty 등[[11]](#endnote-11)이 제안한 MERO 검출 방법은 퍼징 기반의 넷리스트 하드웨어 트로이목마 검출 방법이다. MERO는 희귀 노드(Rare Node) 개념을 도입하여 퍼징의 성능을 향상시킨다. 무작위 테스트 패턴 생성과 동일한 시작을 갖는다. MERO는 생성된 테스트 패턴이 희귀 노드를 트리거 할 때 이를 감소 패턴 집합에 더해 테스트 패턴의 길이를 단축하고 하드웨어 트로이목마를 검출할 가능성이 높은 테스트 패턴을 우선 획득하게 되는 구조를 지니고 있다. MERO는 기존의 무작위 테스트 패턴 생성에 비해 실행 시간을 단축하고 퍼징 정확성을 향상시킬 수 있는 장점을 지녔으나, 시작 테스트 벡터 크기에 영향을 받기 때문에 무작위성을 지니며 대상 하드웨어 디자인의 크기가 커질 경우 실행 시간이 크게 증가하는 확장성(Scalability) 문제를 지닌다.

Y. Lyu 등[[12]](#endnote-12)은 핵심 경로(critical path)를 통해 실행되는 하드웨어 트로이목마가 부채널 분석의 패스 딜레이(path delay)에서 유의미한 차이를 보임을 주장했다. 해당 연구는 MERO와 마찬가지로 하드웨어 트로이목마의 특성에 기반한 휴리스틱 방법으로, 하드웨어 트로이목마의 특성 중 MERO에서 제안한 희귀 노드 뿐만 아니라 작동 과정에서 사용되는 제어 흐름(control flow)를 고려하여 테스트 패턴을 생성한다. 해당 검출 모델은 MERO와 동일한 방식으로 테스트 패턴을 생성하나, 희귀 노드의 활성 여부를 나타내는 피처 벡터(feature vector)의 해밍 거리와 테스트 벡터의 해밍 거리의 합을 통해 패스 딜레이를 계산하여 정렬을 수행하여 최적 테스트 패턴을 생성한다. 해당 연구는 MERO와 비교하여 정확도를 향상시키고 테스트 패턴 생성을 더욱 단축시켰으나 마찬가지로 무작위 테스트 패턴 생성 시작에 의존한다는 문제점을 공유한다.

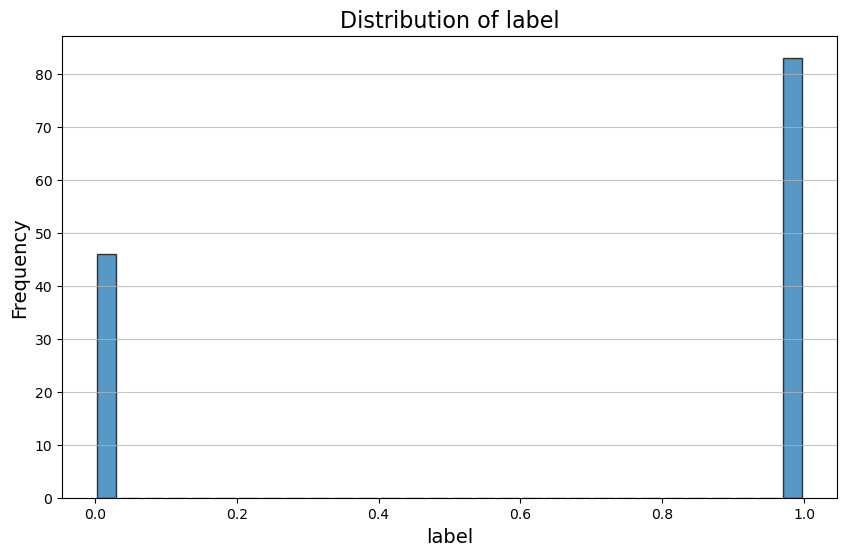
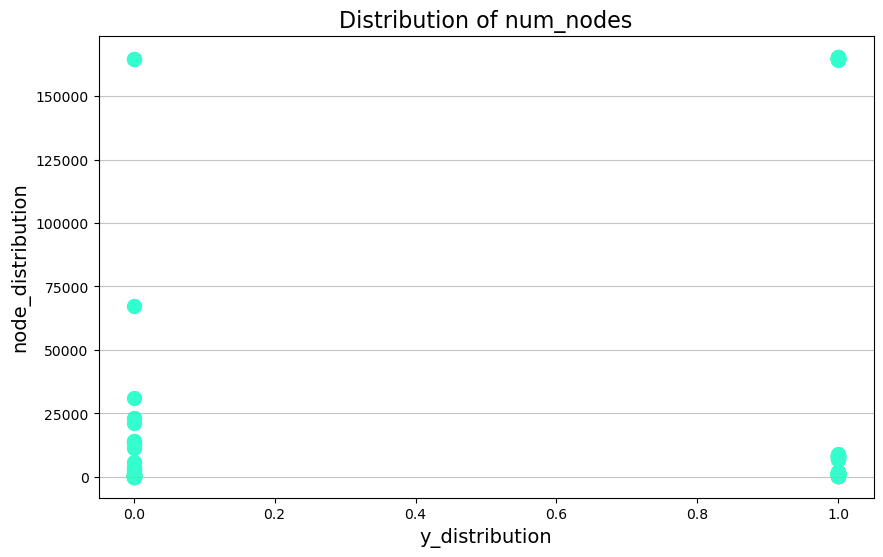
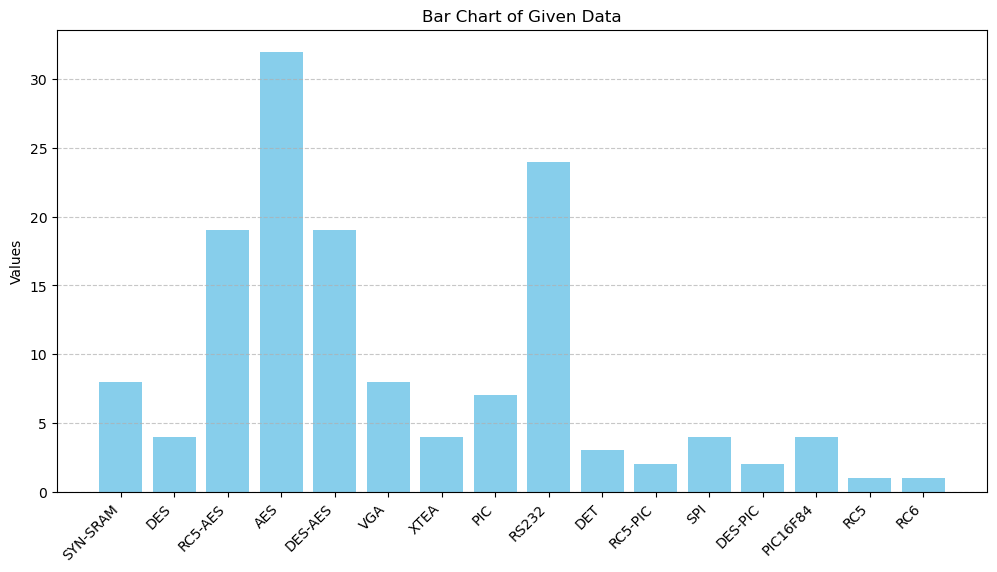
Z. Pan 등[[13]](#endnote-13)은 강화 학습을 통해 하드웨어 트로이목마 검출 용도의 부채널 분석에 사용되는 통한 테스트 패턴 생성을 최적화 하는 방식을 제안했다. 해당 연구는 앞서 제시한 하드웨어 트로이목마의 특성인 희귀 노드와 핵심 경로를 적용하여 검출 모델이 하드웨어 디자인을 학습하여 테스트 패턴을 생성하는 구조를 제시한다. 해당 모델은 MERO와 동일하게 최초 테스트 패턴의 생성 시 희귀 노드를 트리거하는 테스트 패턴을 생성하는 생성 알고리즘과, 희귀 노드의 활성화 정도와 각 벡터의 해밍 거리를 피처로 사용하여 강화 학습을 수행하는 학습 알고리즘으로 구성되어 있다. 이 연구는 생성 알고리즘을 통해 유효한 테스트 패턴을 생성하고, 학습 알고리즘을 통해 강화 학습으로 훈련된 모델을 사용하여 최적의 테스트 패턴을 생성한다. 이 모델은 이전 연구들에서 제안한 핵심 개념을 통해 테스트 벡터의 생성, 최적 테스트 패턴 생성 시간 등 기존 퍼징 기반 검출 방법과 비교하여 높은 정확성과 짧은 상대적으로 실행 시간이 소모되는 장점을 보였으나, MERO의 확장성 문제와 같은 퍼징 모델의 단점을 공유한다.

Hoang M. Le 등[[14]](#endnote-14)은 고수준 합성(High-Level Synthesis, HLS) 대상 퍼징 모델을 제안했다. 해당 연구는 기존 퍼징 기반 검출 모델이 고수준 합성을 통해 생성된 하드웨어 트로이목마(Synthesizable Hardware Trojan, SHT) 검출에 적합하지 않다는 배경에서 진행되어 SHT가 지닌 변이(Mutation)을 적용한 범위 지정 퍼징(Coverage-guided Fuzzing) 구조를 제안한다. 해당 모델은 초기화(Initialization), 피드백 반복(Feedback Loop), 종료(Stop Fuzzing) 세 단계로 이루어져 있으며 각 단계는 위한 제어 흐름과 테스트벤치를 사용한 SystemC 시뮬레이션을 통해 퍼징 환경 구현, 시뮬레이션에서 발생하는 정보를 통해 범위(Coverage)를 수정하는 피드백 수행, 퍼징 종료 후 대상 하드웨어 디자인의 정상 및 비정상 판별을 수행한다. 이 연구는 기존 하드웨어 기술 언어(Hardware Description Language, HDL) 기반 검출기 뿐만 아니라 고수준 합성을 통해 생성되는 하드웨어 트로이목마에 대한 분석과 이를 적용한 검출 구조에 대해 제안한다.

K. Hasegawa 등[[15]](#endnote-15)은 게이트 수준의 넷리스트의 하드웨어 트로이목마 벤치마크인 Trust-HUB에서 추출된 11개의 Trojan net 피처를 추출하여 다층 신경망을 사용하여 정상 회로와 트로이목마 회로를 구별하는 모델을 제안했다. Trojan net 피터는 초기 51개의 피처에서 랜덤 포레스트(Random Forest)를 통해 11개가 선정되었으며, 다층 신경망은 11개의 피쳐와 2개의 출력을 가진다. 실험 과정에서 1개, 2개, 3개의 Middle layer가 사용되었다. 해당 연구는 퍼징과 다르게 테스트 패턴 생성이 아닌 하드웨어 디자인에서 피처를 추출하여 이를 다층 신경망에서 분류하는 구조를 지닌다. 따라서 이전 퍼징 모델들이 공유했던 확장성 문제 및 실행 시간에 관련된 문제점에서 자유롭다는 특징이 존재하나 Trojan net 피처만으로 수행되는 하드웨어 트로이목마 분류는 퍼징에 비해 정확도가 떨어진다는 단점이 존재한다.

1. **Design**
2. **dataset**

GCN으로 Netlist 단의 graph 데이터에 대해 HT detection 을 진행할 시 large-scale circuits가 challenges가 될 수 있다. Netlist는 RTL과 다르게 데이터가 방대하고 크기 때문에 모델에 한번에 input 하여 구성하는 것은 시간과 용량같은 방대한 리소스를 요구한다. 본 연구 실험에서 사용한 TrustHub 데이터 셋은 129개의 데이터가 존재하며 각각46개와 83개의 기본 구조와 trojan 구조를 가지고 있고 고르지 않은 구조 분포를 가지고 있다. 하나의 데이터셋이 가질 수 있는 최대 크기는 165493의 크기이며 최소 크기는 62이다.

이러한 데이터에 대해 max input 크기를 줄이며 데이터의 다양성을 늘리는 효율적인 방법이 필요하다. 이에 대해 3가지 측면에서 방법을 탐구한다.

1. **데이터 표현 방식**

그래프 데이터를 표현하는 방식에는 direct 와 undirect 가 있다. direct 는 방향성을 지니며 모델의 입장에서는 input 과 output 의 추가적인 feature 를 고려하며 학습을 할 수 있다. 반면 undirect는 방향성이 없기에 오로지 connection 을 위주로 모델이 학습해야 한다. 이는 I/O feature에 대해 selection 의 효과를 야기하며 direct 보다 더 빠른 train 효과를 기대한다.

feature matrix 는 cell 들의 feature embedding 으로 나타난다. 이 때 유사한 feature 들은 묶어서 학습하는 group 방법은 undirect 와 마찬가지로 비슷한 cell 에 대한 사소한 feature 를 무시하는 selection 효과를 야기할 수 있으며 모들 cell 에 대해 one hot encoding 으로 feature 를 표현한 original 방법보다 빠른 train 효과를 기대할 수 있다.

1. **샘플링 어그멘테이션**

Netlist 데이터의 가장 큰 문제인 방대한 크기의 input 데이터는 모델이 학습에 사용할 시간과 용량을 비효율적으로 늘린다. 이를 줄이기 위해 원본 구조에서 sub 구조를 sampling 해서 input 을 만드는 법을 도출하는 방법을 고안한다. 기본적으로 설정한 hyperparameter인 max node를 기준치로 이를 초과하는 input 에 대해 sampling 을 진행하여 sub graph로 학습하는 방법이다. 이는 리소스를 제한하는 방법일 뿐만아니라 sampling 의 위치에 따라 augmentation 효과로 데이터를 다양하게 뽑아 학습할 수 있도록 해준다. 효과적인 augmentation 을 하기 위해 원본 구조에서 유심히 봐야하는 trojan cell 등을 중점적으로 sampling 을 진행하며 최대한 cell 들간의 connection 이 유지될 수 있도록 중요 cell 들에서 random 하게 뽑은 init node에서 bfs 를 통해 cell 들을 연결해가며 sampling 을 진행한다. 또한 더욱 효과적인 augmentation을 위해 max node 를 넘지 않은 데이터에 대한 augmentation은 graph 데이터를 표현하는 adjacency matrix의 특징인 같은 graph라도 permute 된 matrix 가 나올 수 있다는 점에서 row와 col을 graph가 망가지지 않게 permute 하는 방법으로 augmentation을 진행한다. 이를 통해 적은 class 당 데이터를 augmentation 하여 진행하여 accuracy 상승과 robustness 한 학습을 기대할 수 있다.

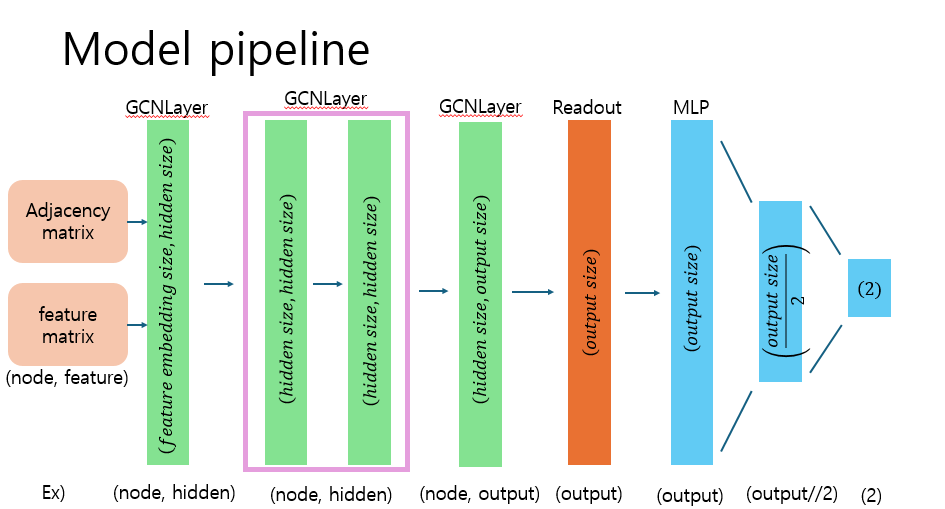
1. **컴파일러**

netlist 데이터를 tcl 파일에서 verilog file 로 변환하는데의 컴파일러를 다르게 하여 다양한 종류의 graph 데이터를 뽑아낼 수도 있다. 이에 본 연구에서는 기본적인 normal complier 와 함께 max area를 0으로 설정하여 최대한 면적을 작게하는 area complier 와 clock 속도 자체를 500MHz로 설정하여 고속의 clock 에도 동작하도록 한 fclock complier을 사용하여 단독으로 사용하거나 합쳐서 사용하는 방식으로 모델의 학습에 기여하고자 한다.

1. **model**

모델은 Rozhin Yasaei [[16]](#endnote-16)가 사용했던 GCN 을 base 로 수정하여 사용한다.

총 4개의 GCN layer 를 가지며 후에 readout 을 지나 MLP 를 지나는 구조로 되어있다. input 은 adjacency matrix와 feature matrix 이며 이는 첫 GCN layer 를 지나고 2개의 hidden layer 후 마지막 GCN layer 를 지난다. 그 후 ReadOut을 지나는데 Readout 이 야기하는 효과는 graph 데이터 특성상 같은 graph라도 permute 된 다른 matrix 로 표현 가능한데 이것을 해결해주며 각기 다른 input size 에 대해서도 일관적인 feature output vector를 뽑을 수 있게 해준다. 마지막으로 feature 에 대한 classification을 제공해주는 MLP 를 끝으로 구조가 마무리 된다. 아웃풋은 2개의 vector 로 [ Ht 일 확률, ht 가 아닐 확률 ] 과 같이 나오게 된다. 이를 통해 인풋 데이터에 대해 HT 를 classification 할 수 있다. 기존 연구에서 사용했던 Attention Layer는 연산량이 크고 입력 크기를 더욱 제한하는 단점이 있어 본 연구에서는 제외하였다. 대신, Residual Connection의 hidden layer 만을 추가하여 정보 흐름을 원활하게 하고 모델의 학습 효율성을 높였다. 이러한 구조적 개선을 통해 모델이 방대한 input 에 대해 보다 효과적으로 Trojan을 탐지할 수 있도록 설계되었다



1. **experiment**

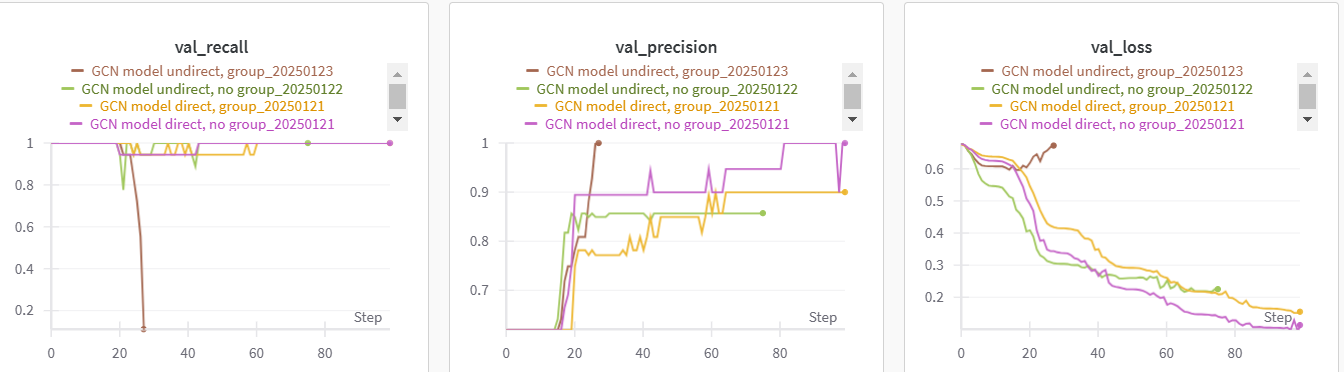
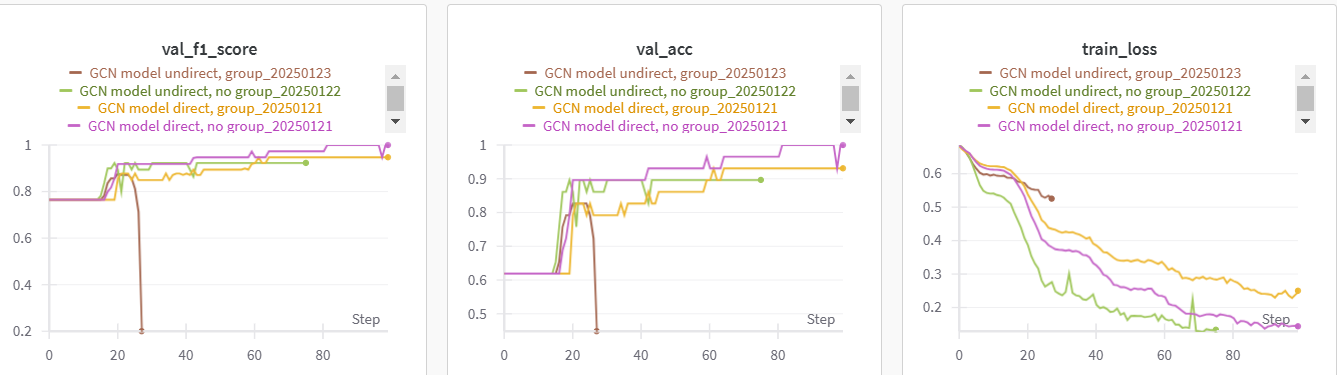
실험은 앞서 소개한 데이터의 표현 방식, 샘플링 여부, 컴파일러 에 대한 결과 분석으로 진행한다. 기본적인 config 는 criterion으로는 BCE 를 사용하고 optimizer로 AdamW 를 사용한다. 이 때 더욱 효과적인 optimization을 위해 SAM optimizer 를 사용하여 학습한다.[[17]](#endnote-17) 이는 gradient 를 완만하게 해주어 효과적인 학습을 야기한다. learning rate는 1.e-4 를 사용하고 scheduler 는 CosineAnnealingLR을 사용한다. (T\_max는 10, eta\_min은 1.e-6) 총 epoch는 100 이고 patient 는 10이다.

이번 실험에서는 batch는 1로 고정된다. 왜냐하면 input 별 사이즈가 모두 달라 하나의 tensor 로 합칠 수 없기 때문이고 다른 input 사이즈에 대해 model이 잘 대응할 수 있는가를 확인해보고자 하기 때문이다. 만약 빠른 속도의 실험을 위해 batch 를 늘린다면 max node hyperparameter 를 기준으로 padding 또는 sampling 하여 맞추어 사용한다. 실험에서의 max node는 30000을 기준으로 잡았다.

평가지표는 loss , f1 score , precision, recall, accuracy 를 사용하며 train dataset 은 원본의 0.8, validation dataset 은 원본의 0.2만 사용한다. 이 때 sparify는 최대한 class 별로 validation dataset이 고루 분포되게 설정하여 사용한다.

1. **데이터 표현 방식 : index 11, 12, 13, 14**

데이터 표현 방식에는 앞서 말한 direct, undirect 와 group, original 의 방법이 있다. 이를 2 x 2 로 총 4개의 동일한 환경에서 비교 실험을 진행해보았다. 이 때 node 가 많은 input 은 실험 환경의 리소스를 초과하는 용량을 요구하기 때문에 이를 해결하기 위해 train pipeline 안에서 max node 를 넘어서는 sample 에 대해서는 샘플링의 방식으로 subgraph를 뽑아내어 학습하였다. 총 train dataset 은 113개이고 validation dataset은 29개이다.



|  |  |  |
| --- | --- | --- |
| accuracy, loss | direct | undirect |
| original | 100%, 0.1134 | 89%, 0.2253 |
| group | 93%, 0.1549 | 44%, 0.6724 |

|  |  |  |
| --- | --- | --- |
| f1 score,  precision,  recall | direct | undirect |
| original | 100%, 100%, 100% | 92%, 85%, 100% |
| group | 94.7%, 90%, 100% | 20%, 100%, 11.1% |

위 결과를 통해 알 수 있는 것은 undirect 는 direct 보다 loss가 더욱 빠르게 떨어진다는 것을 확인할 수 있지만 최종적으로 도달 가능한 성능이 낮은 편이다. 이것으로 Trojan을 detection 하는 것에는 input, output 의 고려와 세부적인 cell 의 차이같은 feature 또한 중요하다는 것을 알 수 있다.

feature 의 detail이 가장 낮은 undirect, group은 학습이 실패하는 모습도 보인다. 이는 Trojan 을 detection 할 때 방향 또는 세부 cell 의 특징 중 하나는 고려해야 Trojan 을 detection 할 수 있다고 해석 가능하다.

feature detail이 가장 큰 direct, original 은 100% 에 가까운 정확도를 보였다. 하지만 이는 아무런 augmentation 을 하지 않은 작은 평가 데이터에 의해 어느정도 과장된 결과로도 해석 가능하다. 따라서 데이터를 augmentation 후 평가한 지표와 비슷한 결과가 나온다면 신뢰 가능한 결과라고 인식할 수 있다.

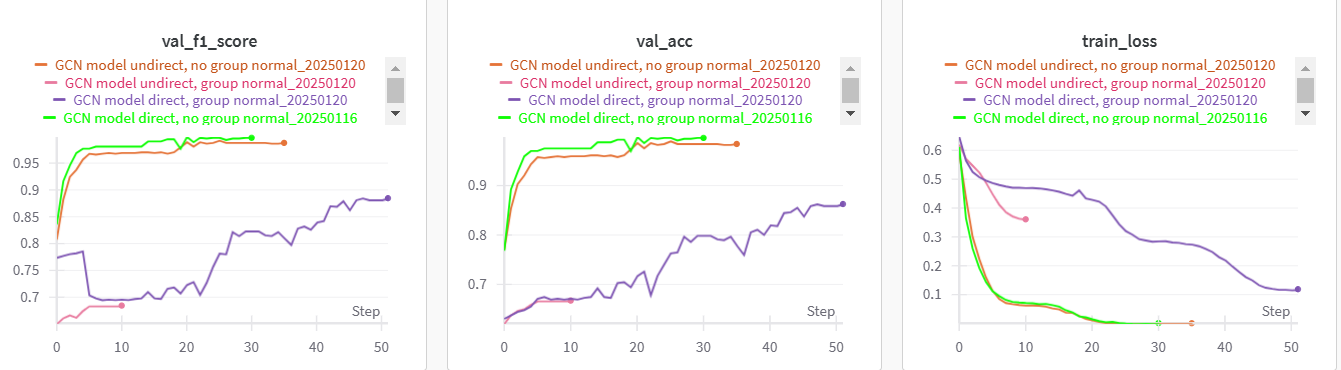
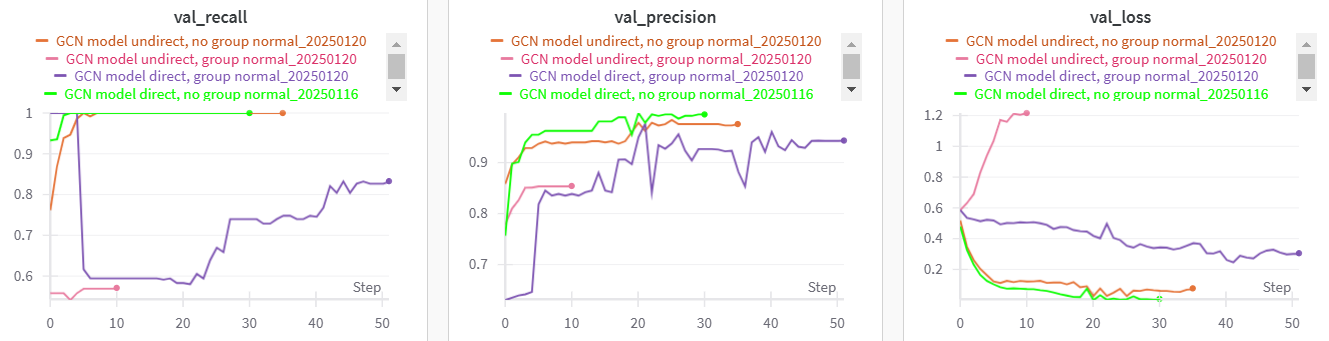
|  |  |  |
| --- | --- | --- |
| train latency (1 epoch),  epoch | direct | undirect |
| original | 2:45, 99 | 13:05, 75 |
| group | 2:43, 99 | 13:04, 27 |

latency 를 비교해보았을 때 전체적으로 1 epoch 당 높은 latency 를 가지는데 이는 train pipeline 안에서 sampling 을 진행하기 때문이다. direct 가 undirect 보다 더 빠른 latency를 가지는 이유는 undirect의 netlist code 경우 양쪽 방향의 connection을 연결하기 때문에 데이터의 길이가 약 2배까지 늘어나기 때문이다. 그럼에도 latency 가 2배가 아닌 더 길게 확인 되는 것은 BFS의 특성으로 한쪽 방향만 sampling 해야하는 direct 와 다르게 양쪽 방향으로 sampling 을 진행해야 하기 때문이었다.

따라서 성능이나 속도이나 모두 direct 가 undirect 보다 trojan detection 을 위한circuit 을 표현하기 좋은 표현 방법이며, 더욱 구체적인 original feature 정보가 성능면에서 group 정보보다 우세하다는 것을 알 수 있다.

1. **샘플링 어그멘테이션 : index 5, 8, 9, 10**

다음 실험은 샘플링을 통한 augmentation에 대한 성능 비교 실험이다. 이번 실험에서는 샘플링은 offline augmentation 으로 미리 데이터를 샘플링 어그멘테이션을 다른 random seed 를 통해 특정 횟수로 반복한다. 이번 실험에서는 x20 의 횟수로 augmentation을 진행하였고 결과적으로 train dataset의 113 개의 데이터는 2272까지, validation dataset의 29개의 데이터는 568개까지 증가할 수 있게 되었다.



|  |  |  |
| --- | --- | --- |
| accuracy, loss | augmentation X | augmentation O |
| direct,  original | 100%, 0.1134 | 99.6%, 0.008 |
| direct,  group | 93%, 0.1549 | 86%, 0.3 |
| undirect,  original | 89%, 0.2253 | 98%, 0.077 |
| undirect,  group | 44%, 0.6724 | 66%, 1.21 |

|  |  |  |
| --- | --- | --- |
| f1 score,  precision,  recall | augmentation X | augmentation O |
| direct,  original | 100%, 100%, 100% | 99.7%, 99.4%, 100% |
| direct,  group | 94.7%, 90%, 100% | 88.4%, 94.3%, 0.83% |
| undirect,  original | 92%, 85%, 100% | 98.7%, 97.5%, 100% |
| undirect,  group | 20%, 100%, 11.1% | 68.4%, 85.4%, 57.1% |

실험의 결과로 정확도와 loss 가 어느정도 튜닝되었고 보정된 사실을 알 수 있다. direct, original 에 대해서는 조금 더 보정된 정확도가 나왔고 매우 떨어진 loss를 볼 수 있었다. 이에 direct, original 데이터가 Trojan detection에 적합하다고 파악가능하며 sampling augmentation은 학습에 효과를 미쳤다고 볼 수 있다. 또한 original 데이터를 쓰는 실험들이 모두 성능이 올랐는데 이를 통해 많아진 데이터로 정교하게 학습한 model 에서 방향보다 cell feature 의 다양성을 중요시 보고 있다는 것을 알 수 있다.

|  |  |  |
| --- | --- | --- |
| train latency (1 epoch),  epoch | augmentation X | augmentation O |
| direct,  original | 2:45, 99 | 1:32, 30 |
| direct,  group | 2:43, 99 | 1:32, 51 |
| undirect,  original | 13:05, 75 | 2:14, 35 |
| undirect,  group | 13:04, 27 | 2:18, 10 |

데이터가 늘어난 만큼 model이 학습하는 epoch 가 크게 준 것을 확인 할 수 있다. 이 때 train latency 또한 줄었는데 이것은 미리 sampling 을 진행해두어 train pipeline 동안의 sampling 이 필요 없어졌기 때문이다. 하지만 줄어든 만큼 offline augmentation 의 시간이 필요하다. ( 증강 횟수에 따라서도 비례한다 ).

x20 의 offline augmentation 은 direct 에서 약 1~2시간, undirect에서 약 7~8시간이 걸렸다. 이는 train pipeline 에서 sampling 했을 때의 latency 와 어느정도 비례한다.

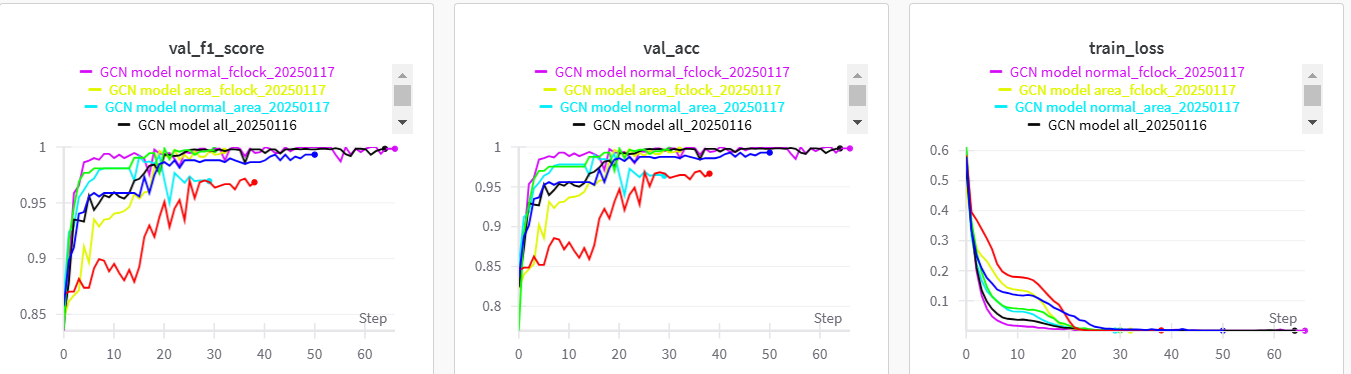
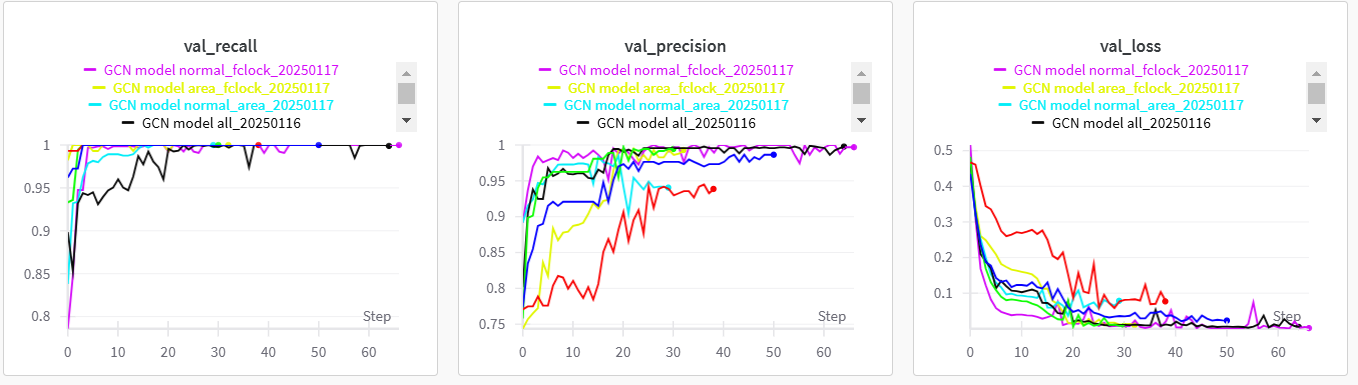
1. **컴파일러 : index 1, 2, 3, 4, 5, 6, 7**

다음 연구로 7가지의 컴파일러 조합을 실험하였다. 각 조합은 데이터의 augmentation으로 x20 적용하여 실험을 수행하였으며, 단일 조합의 데이터 양이 가장 작고 모든 조합을 포함한 경우 데이터가 3배로 가장 많도록 구성하였다. 실험에 사용된 7가지 컴파일러 조합은 다음과 같다.

단일 조합: normal, area, fclock

이중 조합: normal + area, area + fclock, normal + fclock (데이터 2배)

삼중 조합: normal + area + fclock (데이터 3배)



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | normal | area | fclock | normal + area | area + fclock | fclock + normal | normal + area + fclock |
| accuracy | 99.6% | 99.2% | 96.6% | 96.3% | 99.6% | 99.8% | 99.8% |
| loss | 0.008 | 0.02 | 0.077 | 0.078 | 0.01 | 0.002 | 0.0052 |
| f1 score | 99.7% | 99.3% | 96.8% | 96.9% | 99.6% | 99.8% | 99.8% |
| precision | 99.4% | 98.6% | 93.8% | 94% | 99.2% | 99.2% | 99.7% |
| recall | 100% | 100% | 100% | 100% | 100% | 100% | 99.8% |

실험 결과로 baseline 보다 loss 가 더욱 떨어진 실험들은 fclock + normal 과 normal + area + fclock 이고 이들 중 가장 적은 loss를 보인 것은 fclock + normal 이다. 일반적으로 fclock을 포함한 조합이 높은 성능을 보였으며, 특히 normal을 함께 조합하였을 때 가장 효과적인 성능 개선이 나타났다. fclock의 단일 조합의 경우 다른 조합 대비 높은 loss를 기록하였으나, 다른 컴파일러와 함께 사용되었을 때 안정성이 향상됨을 확인할 수 있었다. 이를 통해 fclock으로 augmentation 효과를 볼 수 있다는 것을 알 수 있다.

1. **결론**

본 연구에서는 방대한 Netlist 데이터에 대해 효율적인 Trojan 탐지 기법을 도출하였다. Direct 여부와 Group 여부의 비교를 통해 Trojan 탐지가 선호하는 데이터 표현 방식을 분석하였으며, 입력 크기를 제한하고 증강(Augmentation) 효과를 적용하는 샘플링 기법이 유효한지 평가하였다. 또한, 다양한 컴파일러 조합이 모델의 안정성과 성능에 미치는 영향을 조사하였다. 실험 결과, Trojan 탐지는 Direct 및 Original Feature 기반의 데이터 표현을 선호하는 것으로 나타났으며, BFS 샘플링을 통해 입력 크기를 제한하는 것이 효과적이었다. 더불어, 다양한 위치에서의 샘플링과 여러 컴파일러 조합을 활용한 증강 기법이 모델의 안정성과 성능 향상에 기여함을 확인하였다. 또한, Attention Layer 없이 Residual Connection을 활용하고, 최적화 과정에서 SAM Optimizer를 도입하는 것이 Trojan 탐지 성능을 더욱 강화하는 데 효과적이었다. 본 연구의 결과는 방대한 Netlist 데이터를 대상으로 한 Trojan 탐지 문제 해결을 위한 하나의 방안을 제시하며, 향후 더욱 안정적이고 높은 성능을 갖춘 모델 개발을 위한 연구가 지속적으로 이루어질 필요가 있다.

1. Le, Hoang M., et al. "Detection of hardware trojans in SystemC HLS designs via coverage-guided fuzzing." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019. [↑](#endnote-ref-1)
2. KEVIN IMMANUEL GUBBI, BANAFSHEH SABER LATIBARI, ANIRUDH SRIKANTH, TYLER SHEAVES. “Hardware Trojan Detection Using Machine Learning: A Tutorial” Hardware Trojan Detection Using Machine Learning: A Tutorial. ACM, 2023 [↑](#endnote-ref-2)
3. Ayush Jain, Ziqi Zhou and Ujjwal Guin. “Survey of Recent Developments for Hardware Trojan Detection” 2021 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2021. [↑](#endnote-ref-3)
4. Shivam Bhasin, Francesco Regazzoni. “A Survey on Hardware Trojan Detection Techniques” 2015 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2015 [↑](#endnote-ref-4)
5. He Li, Qiang Liu, Jiliang Zhang. “A survey of hardware Trojan threat and defense” Integration, Volume 55, Pages 426-437, ISSN 0167-9260, 2016 [↑](#endnote-ref-5)
6. Shivam Bhasin, Francesco Regazzoni. “A Survey on Hardware Trojan Detection Techniques” 2015 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2015 [↑](#endnote-ref-6)
7. He Li, Qiang Liu, Jiliang Zhang. “A survey of hardware Trojan threat and defense” Integration, Volume 55, Pages 426-437, ISSN 0167-9260, 2016 [↑](#endnote-ref-7)
8. Ayush Jain, Ziqi Zhou and Ujjwal Guin. “Survey of Recent Developments for Hardware Trojan Detection” 2021 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2021. [↑](#endnote-ref-8)
9. Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, Mohammad Abdullah Al Faruque. “Hardware Trojan Detection using Graph Neural Networks” IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. IEEE, 2022. [↑](#endnote-ref-9)
10. Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, Mohammad Abdullah Al Faruque. “Hardware Trojan Detection using Graph Neural Networks” IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. IEEE, 2022. [↑](#endnote-ref-10)
11. Chakraborty, Rajat Subhra, et al. "MERO: A statistical approach for hardware Trojan detection." International Workshop on Cryptographic Hardware and Embedded Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009 [↑](#endnote-ref-11)
12. Lyu, Yangdi, and Prabhat Mishra. "Automated test generation for Trojan detection using delay-based side channel analysis." 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020. [↑](#endnote-ref-12)
13. Pan, Zhixin, Jennifer Sheldon, and Prabhat Mishra. "Test generation using reinforcement learning for delay-based side-channel analysis." Proceedings of the 39th International Conference on Computer-Aided Design. 2020. [↑](#endnote-ref-13)
14. Le, Hoang M., et al. "Detection of hardware trojans in SystemC HLS designs via coverage-guided fuzzing." 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2019. [↑](#endnote-ref-14)
15. Hasegawa, Kento, Masao Yanagisawa, and Nozomu Togawa. "Hardware Trojans classification for gate-level netlists using multi-layer neural networks." 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS). IEEE, 2017. [↑](#endnote-ref-15)
16. Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, Mohammad Abdullah Al Faruque. “Hardware Trojan Detection using Graph Neural Networks” IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. IEEE, 2022. [↑](#endnote-ref-16)
17. Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." arXiv preprint arXiv:2010.01412 (2020). [↑](#endnote-ref-17)