Pusan National University
Computer Science and Engineering
Technical Report 2025-09

시계열 특성을 활용한 다양한 이상 거래 탐지 딥러닝 모델 개발과 성능 분석

저자 1 202055551 배근호

저자 2 201924601 추민

저자 3 202155650 윤소현

지도교수 송길태

목 차

1. 서론

1.1. 연구 배경

금융 산업에서 사기 거래 탐지는 카드 해킹, 무단 도용, 보이스피싱 등 다양한 금융 범죄를 조기에 발견하고 예방하는 데 필수적인 과제이다. 인터넷 뱅킹, 모바일 결제, 온라인 커머스 등 디지털 금융 서비스가 확산되면서 거래 데이터의 양은 폭발적으로 증가하고 있으며, 이에 따라 금융 사기의 발생 빈도와 규모도 꾸준히 증가하고 있다.

이러한 상황에서 대규모 데이터를 처리하기 위해 머신러닝이 사기 탐지에 도입되었으며 XGBoost, 랜덤 포레스트 등 결정트리 기반 모델은 다양한 전처리 기법과 결합되어 높은 성능을 보여왔다. 그러나 이러한 접근법은 거래 데이터가 본질적으로 지닌 시계열적 특성과 장기 의존성을 충분히 반영하지 못한다는 한계가 있다. 거래는 시간의 흐름 속에서 연속적으로 발생하며, 특정 시점의 단일 특징만으로는 미묘한 패턴이나 점진적인 변화를 탐지하기 어렵다.

최근에는 딥러닝 기반 접근이 이러한 한계를 극복하기 위한 대안으로 주목받고 있다. 특히 트랜스포머(Transformer) 모델은 어텐션(attention) 메커니즘을 활용해 시계열 데이터 내 모든 시점 간 관계를 병렬적으로 학습할 수 있어 복잡하고 장기적인 시간 패턴을 효과적으로 포착한다. 이러한 특성 덕분에 트랜스포머는 금융 거래 시계열 분석에 적합한 모델로 고려된다.

본 연구는 트랜스포머 기반 임베딩을 통해 거래 데이터의 시계열적 특성과 장기 의존성을 학습하고, 나아가 고객 및 카드 정보와 같은 다양한 추가 데이터를 융합함으로써 이상 거래 탐지 성능을 향상시킨 딥러닝 모델을 개발하고자 한다.

1.2. 기존 문제점

기존의 사기 거래 탐지 연구는 규칙 기반 방식과 머신러닝 기법을 통해 일정 수준의 성과를 보여왔다. 특히 XGBoost, 랜덤 포레스트와 같은 결정트리 기반 모델은 다양한 전처리 기법과 결합하여 높은 정확도로 대규모 데이터를 분석할 수 있다[1]. 그러나 이러한 전통적 머신러닝 접근법은 거래 데이터가 가지는 시계열적 특성과 장기 의존성을 충분히 반영하지 못한다는 한계가 있다. 거래는 시간의 흐름 속에서 연속적으로 발생하며, 단일 시점의 정적 특징만으로는 사기 행위의 복잡한 양상을 포착하기 어렵다.

또한 거래 데이터만을 분석 대상으로 삼는 경우가 많아, 고객 특성이나 카드 속성과 같

은 부가 정보를 충분히 활용하지 못하는 문제가 있다. 실제 금융 사기는 단순히 거래 금액이나 빈도에 의해서만 발생하지 않고, 고객의 행동 패턴, 카드 발급 이력, 지역적 요인 등 다양한 변수가 복합적으로 작용한다. 따라서 기존 연구 방식은 데이터의 다차원적 속성을 통합적으로 반영하는 데 한계가 있으며, 이는 탐지 정확도와 일반화 성능을 저해하는 요인이 된다.

1.3. 연구 목표

본 연구의 목표는 거래 데이터의 시계열적 특성과 다양한 부가 정보를 종합적으로 활용하여 이상 거래 탐지 성능을 향상시키는 것이다. 이를 위해 다음과 같은 구체적인 연구 방향을 설정한다.

첫째, 거래 데이터의 시계열 정보와 장기 의존성을 효과적으로 학습하기 위해 트랜스포머 기반 임베딩을 적용한다. 트랜스포머의 어텐션 메커니즘은 모든 시점 간의 상호 관계를 병렬적으로 학습할 수 있어, 전통적 머신러닝 기법에서 간과되기 쉬운 복잡한 거래 패턴을 포착하는 데 강점을 가진다.

둘째, 트랜스포머를 활용하여 얻은 임베딩을 바탕으로, 고객·카드 등 다양한 속성 데이터를 융합하여 이상 거래 탐지 모델을 개발한다.

- MLP (Multi-Layer Perceptron): 거래 및 고객·카드 정보를 결합한 단순 신경망 기반 탐지 모델
- TGN (Temporal Graph Network): 시간 정보를 반영한 그래프 신경망 기반 탐지 모델
- HTGN (Heterogeneous Temporal Graph Network): 고객, 카드, 거래 등 이종 데이터를 반영한 그래프 기반 탐지 모델

위 세 가지 모델을 각각 구현하여 성능을 비교한다.

셋째, 개발된 모델들의 성능을 정량적으로 비교·분석한다. 단일 트랜스포머 모델의 테스트 성능을 검증하는 동시에, 트랜스포머 임베딩을 입력으로 활용한 MLP, TGN, HTGN 모델의 이상 거래 탐지 성능을 측정하여 평가한다. 평가 지표로는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1-score, PR-AUC를 사용한다.

이와 같은 과정을 통해 본 연구는 거래 데이터의 시계열적 특성과 고객·카드 속성을 종

합적으로 반영한 다양한 딥러닝 기반 이상 거래 탐지 모델을 제안하고, 모델 간 성능을 비교함으로써 실제 금융 보안 시스템에 활용 가능한 최적의 탐지 전략을 모색하고자 한다.

2. 연구 배경

2.1. 트랜스포머

트랜스포머는 2017년 논문 "Attention Is All You Need"[2]에서 처음 소개된 딥러닝 모델로, 기존의 순환 신경망(RNN)이나 컨볼루션 신경망(CNN)의 장기 의존성 문제, 병렬 연산 문제 등의 한계를 극복하며 자연어 처리(NLP) 분야에서 혁신을 일으켰다. 트랜스포머의 핵심은 셀프 어텐션(Self-Attention) 메커니즘이다. 이는 입력 시퀀스의 각 요소가 다른 모든 요소와의 관계를 파악하여 가중치를 부여하는 방식으로 문장 내 단어 간의 의존성을 효과적으로 학습한다. 이러한 특징은 금융 거래와 같은 시계열 데이터에도 매우 유용하다. 거래 시퀀스에서 특정 거래가 이전 또는 이후의 여러 거래들과 어떤 관계를 가지는지 학습함으로써 시퀀스의 장기적인 의존성을 성공적으로 포착할 수 있다.

2.2. 다층 퍼셉트론 (MLP)

다층 퍼셉트론(MLP)은 입력층, 은닉층, 출력층으로 구성된 완전 연결 신경망으로, 비선형 변환을 통해 복잡한 패턴을 학습할 수 있는 기본적이면서도 강력한 모델이다. 금융 거래 이상 탐지에서 MLP는 주로 특징 추출과 분류 단계에서 활용된다. 거래의 금액, 시간, 위치, 상점 유형 등 다양한 수치형 특징들을 입력으로 받아 은닉층에서 이들 간의 복잡한 상호작용을 학습하고, 최종적으로 정상 거래와 이상 거래를 구분하는 확률값을 출력한다. 특히 MLP의 장점은 해석 가능성과 계산 효율성에 있다. 각 특징의 가중치를 분석하여 어떤 요소가 이상 거래 판단에 중요한 역할을 하는지 파악할 수 있으며, 실시간 거래 승인 시스템에서 요구되는 빠른 응답 시간을 만족시킬 수 있다. 또한 다른 복잡한 모델들과 결합하여 앙상블 학습의 기반 모델로도 효과적으로 활용된다.

2.3. 시간 엣지 그래프 (TGN)

시간 엣지 그래프 네트워크(TGN)는 기존 그래프 신경망(GNN)이 시간에 따른 feature

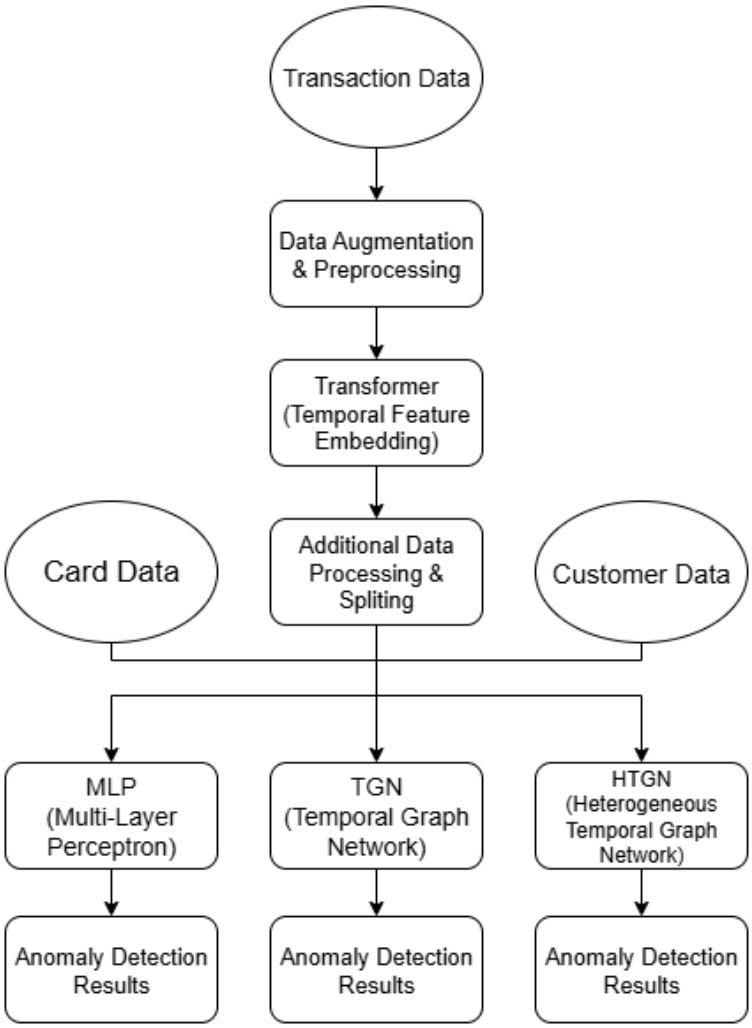
변화 또는 연결성을 표현할 수 없다는 한계를 극복하기 위해 제안된 모델이다. TGN은 거래 이벤트를 시간 순으로 처리하기 위한 메모리 모듈과 그래프 기반 연산자를 결합한 딥러닝 프레임워크를 제공한다[3]. 금융 거래에서는 현재 발생한 거래와 과거에 발생한 거래들 간의 패턴을 분석했을 때 이상치가 발견되는 경우, 이것이 사기 거래로 이어지는 경우가 많다. 따라서 TGN을 활용하여 엣지에 거래 발생 시각 정보를 부여함으로써 시간 흐름에 따른 거래 패턴을 효과적으로 학습할 수 있게 하고, 해당 패턴 속에 존재하는 이상치를 탐지할 수 있게 한다.

2.4. 이종 그래프 (HTGN)

이종 그래프 네트워크(HTGN)는 TGN을 확장한 개념으로, 그래프 내에 여러 종류의 노드(예: 고객, 카드, 상점)와 여러 종류의 엣지(예: 거래 간 연결)가 존재하는 복잡한 구조를 처리할 수 있다. 실제 금융 거래 데이터는 다양한 종류의 엔티티들로 이루어져 있으므로, HTGN은 이러한 이종성을 효과적으로 모델링하는 데 유리하다. 예를 들어, 한 고객의 거래 패턴을 분석할 때, 어떤 카드를 사용했는지, 어떤 상점에서 주로 결제했는지 등 다양한 정보를 종합적으로 고려하여 탐지 성능을 더욱 향상시킬 수 있다.

3. 연구 내용

3.1. 데이터 흐름도



[그림 3-1. 데이터 흐름도]

3.2. 데이터셋 구축 및 전처리

3.2.1. 데이터셋 소개

본 연구에서는 금융 거래 데이터를 기반으로 이상 거래 탐지 실험을 수행하기 위하여 Kaggle에서 제공되는 공개 금융 데이터셋을 활용하였다. 이 데이터셋은 2010년대 금융 거래 기록, 고객 정보, 카드 데이터를 포함하고 있으며, 이상 거래 탐지, 고객 분석, 신용 평가 등 다양한 분석 목적으로 설계되었다. 크게 거래 데이터(transactions), 고객 데이터

(users), 카드 데이터(cards), 사기 거래 라벨(fraud), 상점 업종 코드에 따른 카테고리(mcc codes) 로 구성되어 있으며, 각 데이터 파일의 주요 특징은 다음과 같다.

거래 데이터(transactions_date.csv)

거래 데이터 수는 13,305,915개이고 속성은 아래와 같다.

속성명	설명	값 예시
id	거래 고유 식별자	7475327
date	거래 발생 시각 (YYYY-MM-DD HH:MM:SS)	2010-01-01 00:01:00
client_id	거래를 발생시킨 고객 ID	1556
card_id	사용된 카드 ID	2972
amount	거래 금액 (신용 거래는 양수, 직불 거래는 음수)	$-77.00
use_chip	결제 방식 (Chip / Swipe 등)	Swipe Transaction
merchant_id	상점 고유 ID	59935
merchant_city	상점 도시 (온라인 거래는 ONLINE)	Beulah
merchant_state	상점 주 (미국 외 지역의 경우 국가명)	ND
zip	상점 우편번호 (미국 내 지역인 경우) (미국 외 지역이나 온라인 거래는 결측)	58523
mcc	상점 업종 코드	5499
errors	거래 시 발생한 오류 종류 거래 시 발생한 오류가 없으면 결측 (사기 라벨과는 다른 속성)	(결측)

[표 3-1. 거래 데이터 속성 정리]

고객 데이터(users_data.csv)

고객 데이터 수는 2,000개이고 속성은 아래와 같다.

속성명	설명	값 예시
id	고객 고유 ID	825
current_age	현재 나이	53
retirement_age	은퇴 예상 나이	66
birth_year	출생 연도	1966
birth_month	출생 월	11
gender	성별	Female
address	주소	462 Rose Lane
latitude	거주지 위도	34.15
longitude	거주지 경도	-117.76
per_capita_income	거주 지역 1인당 평균 소득	$29,278
yearly_income	고객의 연간 소득	$59,696
total_debt	고객의 총 부채	$127,613
credit_score	신용 점수	787
num_credit_cards	보유 카드 수	5

[표 3-2. 고객 데이터 속성 정리]

카드 데이터(cards_data.csv)

카드 데이터 수는 6,146개이고 속성은 아래와 같다.

속성명	설명	값 예시
id	카드 고유 ID	4524

client_id	소유 고객 ID	825
card_brand	카드 브랜드	Visa
card_type	카드 유형 (Credit / Debit)	Debit
card_number	카드 번호	4344676511950444
expires	카드 만료일 (MM/YYYY)	12/2022
cvv	카드 보안코드	623
has_chip	카드 보안칩 여부	YES
num_cards_issued	발행된 카드 수	2
credit_limit	신용 한도	$24,295
acct_open_date	계좌 개설일 (MM/YYYY)	09/2002
year_pin_last_changed	마지막 비밀번호 변경 연도	2008
card_on_dark_web	다크웹 유출 여부	No

[표 3-3. 카드 데이터 속성 정리]

사기 거래 라벨 데이터(train_fraud_labels.csv)

사기 거래 라벨 데이터 수는 8,914,963개이고 속성은 아래와 같다.

속성명	설명	값 예시
id	거래 고유 식별자	7475327
Status	거래 정상 여부 (Yes=사기, No=정상)	No

[표 3-4. 사기 라벨 데이터 속성 정리]

상점 업종 코드 데이터(mcc_codes.json)

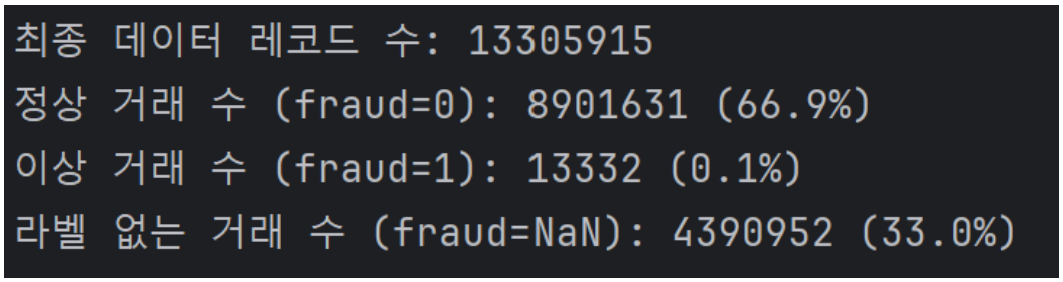
상점 업종 코드 데이터 수는 109개이고 속성은 아래와 같다.

속성명	설명	값 예시
mcc	업종 코드 (Merchant Category Code)	5411
description	업종 설명	Grocery Stores, Supermarkets

[표 3-5. 상점 업종 코드 데이터 속성 정리]

3.2.2. 데이터 전처리 과정

거래 데이터는 원시 상태에서 각 거래에 대한 사기 여부가 별도로 제공되지 않고, 별도의 라벨 데이터셋(fraud.csv)에 저장되어 있다. 이에 따라 본 연구에서는 거래 데이터(transactions_date.csv)와 사기 라벨 데이터(fraud.csv)를 거래 고유 식별자(id) 기준으로 병합하여 각 거래에 대해 정상(0)과 사기(1) 라벨을 부여하였다. 또한, 거래 데이터에는 상점 업종 코드(mcc)만 존재하므로, 상점 업종 코드 매핑 파일(mcc_codes.json)을 활용하여 업종 유형(mcc_type) 속성을 새롭게 생성하였다. 거래 금액(amount) 속성에서는 달러 기호($) 및 구분자(,)를 제거한 뒤 실수형(float)으로 변환하고 절대값을 적용하여 모든 결제 금액을 양수로 정규화하였다.



[그림 3-2. 라벨링 결과 분포]

라벨링 및 금액 전처리 이후, 거래 데이터 내 주요 속성의 결측치를 다음과 같은 규칙에 따라 처리하였다.

- 온라인 거래 처리
    - merchant_city 값이 "ONLINE"인 경우, 해당 거래는 오프라인 매장의 실제 지역 정보가 존재하지 않으므로 merchant_state는 "ONLINE", zip은 00000로 통일하여 채워 넣었다.

- 미국 외 지역 거래의 ZIP 코드 처리
    - 데이터에서 미국 거래는 merchant_state에 주(State) 약어(예: "CA", "NY")로 표기되지만, 미국 외 거래의 경우 국가명이 들어가며 zip 값이 비어 있었다.
    - 이러한 경우, 해당 국가를 식별할 수 있도록 국가명을 대표하는 알파벳 3자리 + "00" 형식(예: CAN00, FRA00)으로 ZIP 코드를 생성하여 채워 넣었다.
- Errors 속성 처리
    - errors 속성은 거래 처리 중 발생한 오류를 나타내며, 오류가 없을 경우 결측값으로 표기되어 있었다.
    - 따라서 결측값은 "No Error"로 치환하여 거래 상태를 명확히 하였다.
- Fraud 라벨 없는 거래 제거
    - 지도학습을 위해서는 사기 여부(fraud) 라벨이 반드시 필요하다.
    - 데이터가 충분히 많으므로 fraud 값이 결측인 행은 제거하여, 학습 및 분석에 활용 가능한 데이터만 남겼다.

이를 통해 거래 데이터의 지역 정보와 오류 상태가 명확히 정의되고 지도학습을 위한 라벨링이 있는 데이터셋을 확보하였다.

.

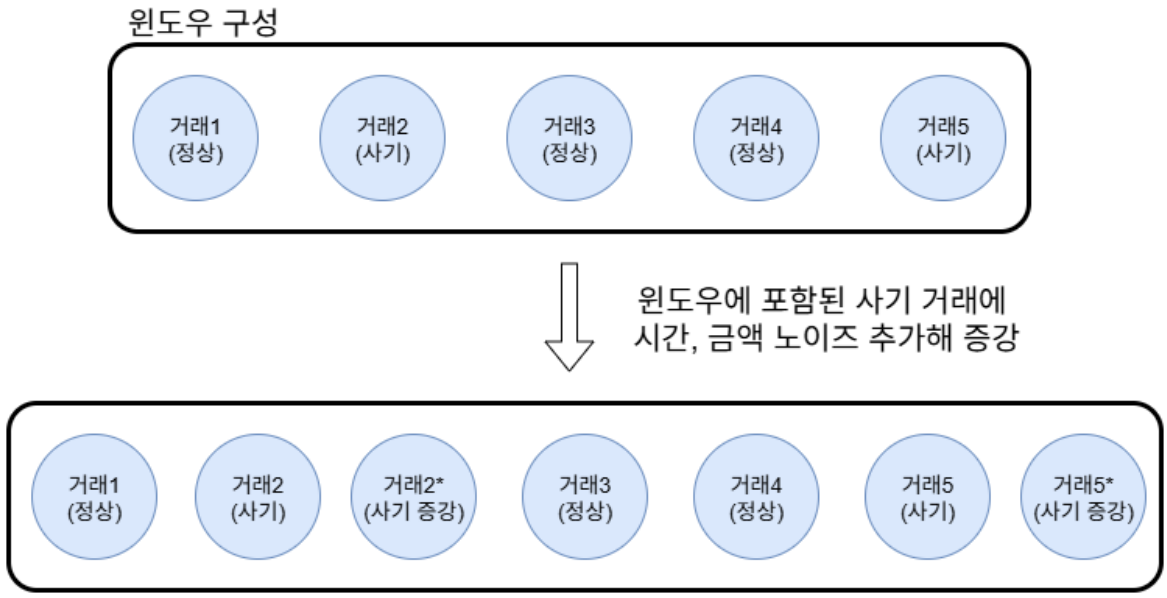
3.2.3. 데이터 불균형 문제 완화

금융 거래 데이터에서는 정상 거래가 대부분을 차지하며 사기 거래는 극히 일부에 불과하다. 본 연구의 거래 데이터도 정상 거래가 압도적으로 많고 사기 거래는 극히 일부에 불과하여 학습 시 모델이 정상 거래 패턴에 과적합되고 사기 거래를 탐지하지 못하는 문제가 발생할 수 있다. 이를 해결하기 위해, 본 연구에서는 사기 거래 증강과 정상 거래 언더샘플링을 병행하였다.

사기 거래는 단일 거래가 아닌 일련의 패턴으로 발생하는 경우가 많다. 이 점을 반영하기 위해 고객과 카드를 기준으로 거래 데이터를 그룹화하고, 거래 데이터로 윈도우를 구성하고, 각 거래의 금액(amount)과 시간(timestamp)에 랜덤으로 노이즈를 더하여 새로운 사기 거래 데이터를 생성하였다. 단순히 새로운 데이터를 무작위로 생성하는 대신 이러

한 방식은 새로운 거래 샘플을 만들어내면서도 등록되지 않은 고객, 카드나 존재하지 않는 우편번호-지역 조합과 같은 컬럼 간 불일치가 발생하지 않도록 보장한다. 즉, 원본 시퀀스를 기반으로 증강을 수행함으로써 고객–카드–상점, 우편번호-지역 간 관계의 일관성을 유지하였다.

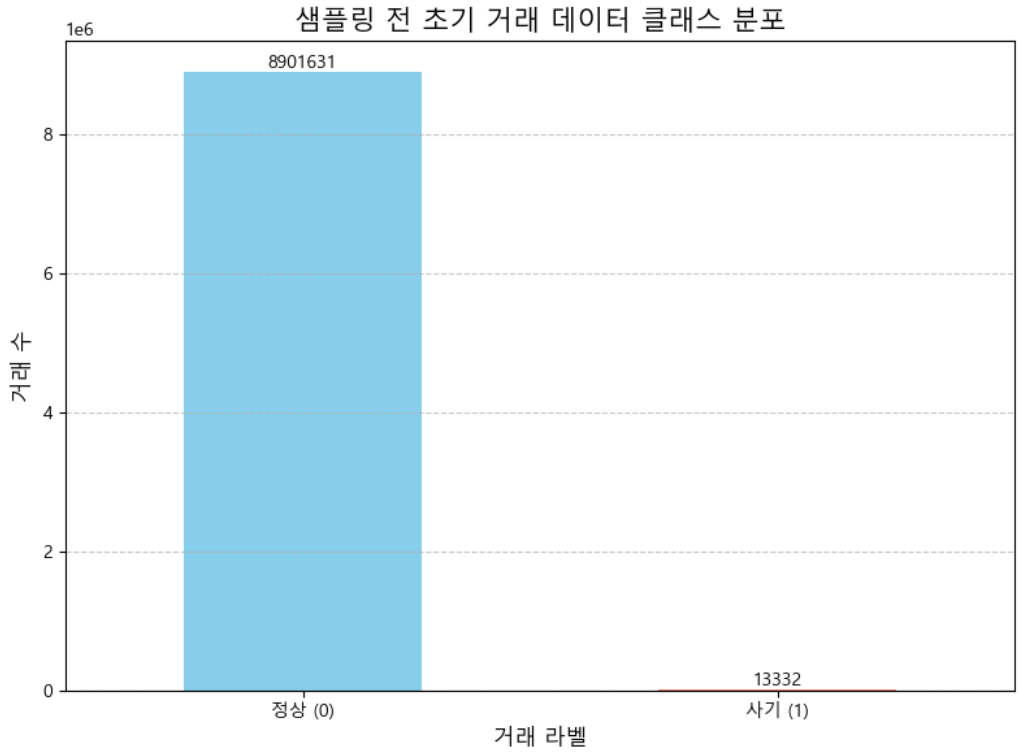
증강 과정의 핵심 로직은 다음과 같다. 먼저, client_id와 card_id를 기준으로 고객–카드별로 거래 데이터를 길이 5의(WINDOW_SIZE=5) 윈도우를 순차적으로(stride=1) 추출한다. 이후 윈도우 내 사기 거래가 포함된 경우, 각 거래를 복사하고 거래 금액에는 ±10% 범위의 랜덤 변동을 적용하였고, 시간에는 ±30분 범위의 랜덤 노이즈를 부여하여 새로운 사기 거래 데이터를 생성한다. 이 과정에서 각 윈도우 내 사기 거래만 선택하여 증강을 적용하였고 WINDOW_SIZE와 stride를 조절하여 사기 거래를 원본 데이터의 약 6배 수준으로 증강하였다.



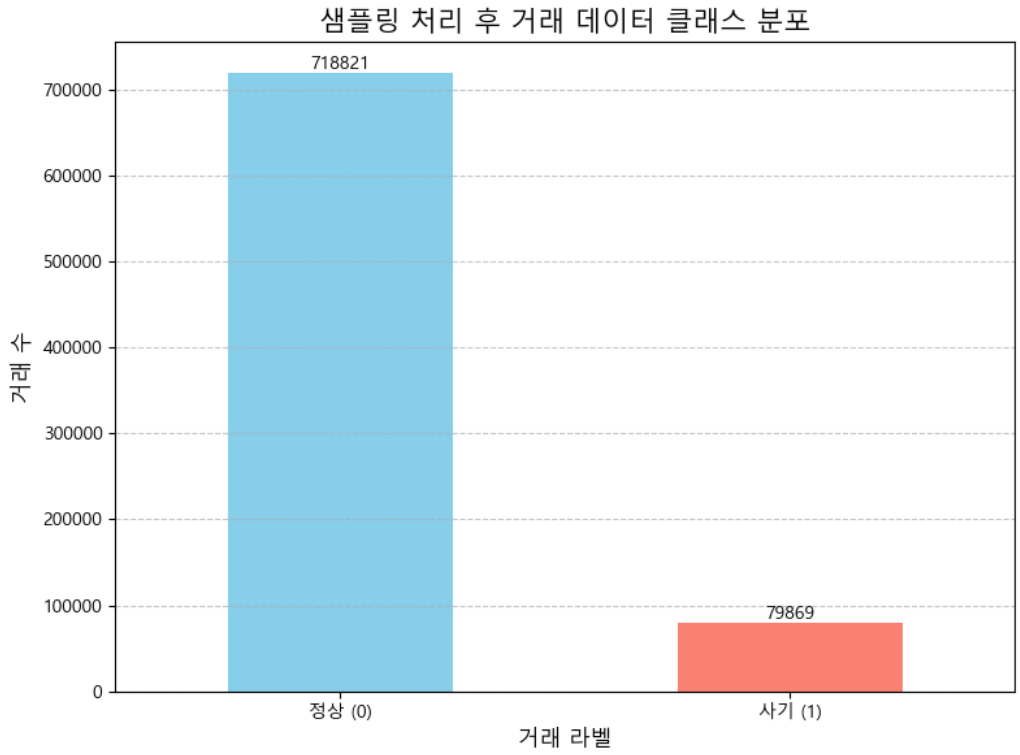
[그림 3-3. 슬라이딩 윈도우 + 노이즈 증강]

정상 거래 언더샘플링은 단순히 데이터를 줄이는 것이 아니라 사기 거래와 유사한 특징을 가진 정상 거래를 포함하기 위한 전략을 사용하였다. 거래 금액, 우편변호(상점 위치 간접), 상점 업종 코드 특성을 기반으로 각 사기 거래와 가장 가까운 정상 거래를 KNN을 이용해 사기 거래 수 만큼 선별하고 목표 정상 거래 수(사기 거래 대비 약 9배 수준)가 될 때까지 랜덤 선별을 통해 보충하였다.

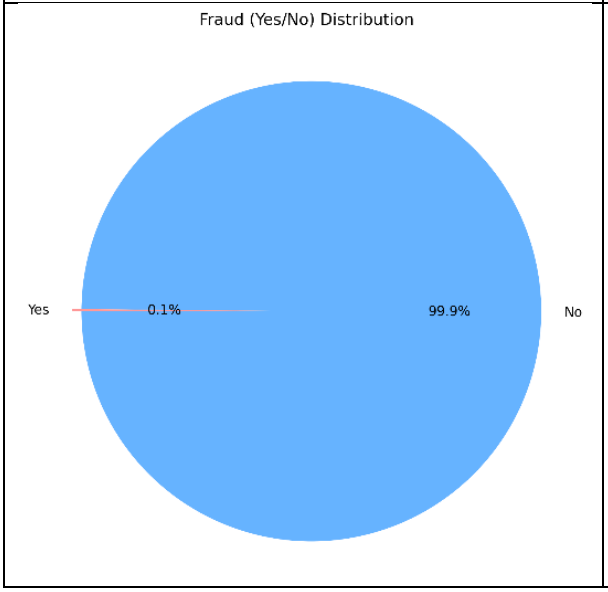
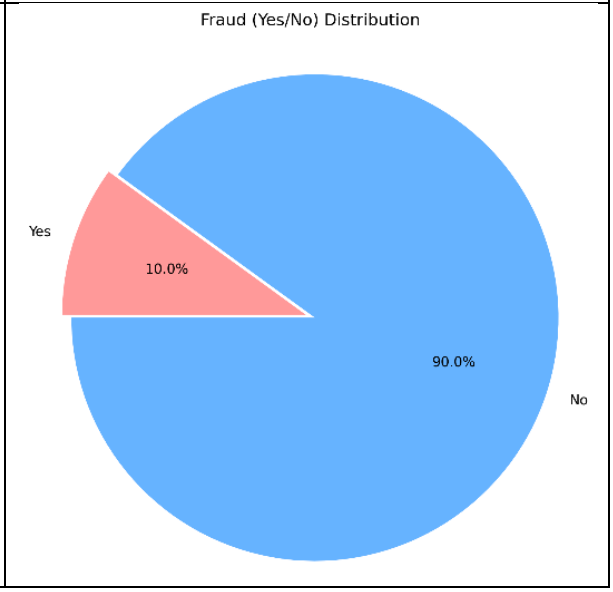
이와 같이 사기 거래 증강과 정상 거래 언더샘플링을 병행함으로써 학습 데이터는 사기 거래를 충분히 포함하고 정상 거래와 사기 거래의 극심한 불균형을 완화하여 모델이 과적합 없이 사기 거래를 효과적으로 학습할 수 있는 구조로 구성된다. 사기 거래는 증강 후 원본 대비 약 6배 수준으로 증가하였고 학습 데이터의 최종 비율은 정상 거래 : 사기 거래 = 약 9:1 수준이다. 이 과정을 통해 모델은 시간적, 금액적, 공간적 패턴을 고려하여 이상 거래 탐지 능력을 향상시킬 수 있다.



[그림 3-4. 데이터 증강 및 샘플링 전 거래 데이터 분포]



[그림 3-5. 데이터 증강 및 샘플링 후 거래 데이터 분포]

증강 및 언더샘플링 전 정상 및 사기 거래 비율	증강 및 언더샘플링 후 정상 및 사기 거래 비율
	

[그림 3-6. 데이터 증강 및 샘플링 전후 거래 데이터 비율 비교]

3.2.4. 데이터 증강 이후 추가 처리

데이터 증강 이후 모델에 공간적 특성 활용과 학습 시 데이터 누수를 방지하기 위해 추가적인 전처리를 수행하였다.

거래 데이터에 포함된 상점의 위치 정보(우편번호, 도시, 주)를 활용해 위도(latitude)와 경도(longitude) 좌표로 변환하여 거래의 공간적 특성을 모델에 반영할 수 있도록 하였다. 위도/경도 추출은 geopy를 사용했으며 비용 절감을 위해 중복된 상점 위치를 제거한 후 처리하였다.

위치 정보의 정확성을 높이기 위해 다음과 같은 우선순위로 좌표를 추출하고 추후 활용하기 위해 별도의 위치 목록 파일로 저장하였다.

1. 우편번호(zip): 가장 정확한 위치를 제공하는 미국 우편번호를 우선적으로 사용하여 좌표를 조회하였다.
2. 도시와 주(city, state): 우편번호로 좌표를 얻지 못했거나 미국 외 국가의 상점 위치인 경우 도시와 주 정보를 조합하여 좌표를 조회하였다.

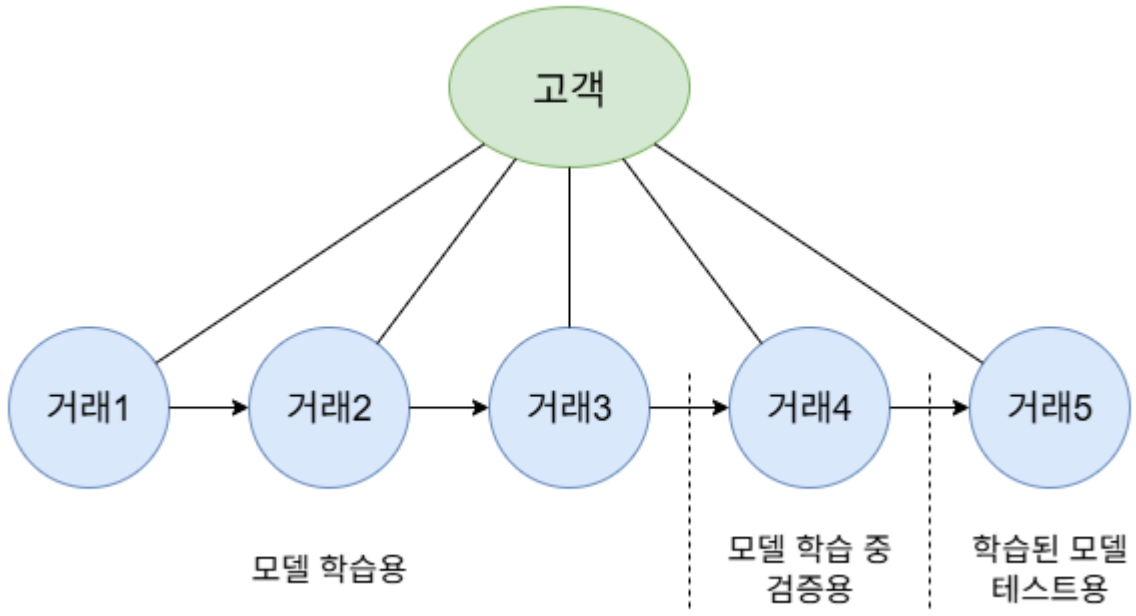
zip	merchant_	merchant_	latitude	longitude
45887	OH	Spencervill	40.70521	-84.366
45891	OH	Van Wert	40.8771	-84.5901
45895	OH	Wapakone	40.57513	-84.1789
45898	OH	Willshire	40.70448	-84.7425
46001	IN	Alexandria	40.25257	-85.6684
46011	IN	Anderson	40.11173	-85.7312
46012	IN	Anderson	40.12888	-85.6494
46013	IN	Anderson	40.06287	-85.6775
46016	IN	Anderson	40.09918	-85.681

[그림 3-7. 변환한 위도/경도 좌표]

거래 데이터에 geopy로 구한 좌표 속성을 추가한 다음 모델의 현실적인 예측 능력을 평가하기 위해 전체 거래 데이터를 학습용(Train)과 테스트용(Test) 데이터셋으로 분리하였다. 이 과정은 특정 고객의 거래가 학습이나 테스트에 몰리는 것을 예방하고 시계열 속성을 위해 시간 흐름을 유지하며 사기 거래의 특성을 고려한 분할 전략을 적용하였다.

- 고객별 시계열 분리: 전체 데이터를 무작위로 분할하는 대신, 고객(client_id)별로 거래 시퀀스를 그룹화한 후 시간 순서(date)에 따라 정렬하였다. 이는 과거 거래를 기반으로 미래 거래를 예측하는 실제 금융 환경을 모사하고 일부 고객에 대한 거래 데이터가 학습용이나 테스트용에 몰리도록 하지 않게 하기 위함이다.
- 사기 연속 구간(Fraud Block) 처리: 사기 거래가 연달아 발생하는 사기 블록의 경우 분할 경계(train_ratio 기준)가 해당 블록의 중간에 위치하면 블록 전체를 한쪽 데이터셋(학습용 또는 테스트용)에 포함시켰다.
    - 사기 탐지 모델은 사기 블록의 전체적인 패턴을 학습하는 것이 중요하다. 사기 블록을 분리하지 않고 한쪽으로 몰아줌으로써 모델이 온전한 사기 패턴을 학습할 수 있게 하여 사기 패턴에 대한 예측 능력을 보다 정확하게 평가할 수 있도록 하였다.

전체 거래 데이터 분할 후 학습용 데이터에 대해 다시 한번 분할을 적용하여 모델 학습용, 학습 과정 중 성능 검증용(얼리스탑핑 조건 검사), 학습 완료된 모델 테스트용으로 데이터를 분할하였다.



[그림 3-8. 고객 별 거래 데이터 분할]

3.3. 트랜스포머 기반 시계열 특징 처리

3.3.1. 트랜스포머 모델 개요

본 연구에서는 고객 및 카드 기반 거래 데이터를 대상으로 시계열적 특성을 효과적으로 학습하여 이상 거래를 탐지하기 위해 트랜스포머 기반 모델을 적용하였다. 트랜스포머는 자연어 처리를 위해 만들어진 모델이지만 시계열 정보 처리에도 강점을 보여 본 연구에서는 거래 패턴 간 관계를 학습하기 위해 멀티헤드 자기 주의(Multi-head Self-Attention) 메커니즘을 활용하였다. 이를 통해 시계열 속성을 가진 거래 간의 복합적 관계를 포착하고 이상 거래 판별 성능을 향상시키고자 하였다. 이 과정은 데이터 전처리, 모델 설계, 학습 및 임베딩 생성의 세부 단계로 구성된다. 학습된 모델의 출력인 임베딩(embedding)은 후속 모델의 입력으로 사용한다.

3.3.2. 트랜스포머 입력 피처 구성 및 전처리

모델 학습에 앞서, 트랜스포머 모델에 적합하도록 원본 데이터를 전처리하고 다양한 특징을 추출하는 과정을 거쳤다. 이 과정은 모델이 거래의 시계열적, 공간적, 범주형 특성을 모두 학습할 수 있도록 하였다.

피처 유형	피처명	설명
수치형	delta_time	직전 거래와의 시간 차(초) Min-Max 정규화
수치형	is_delta_time_missing	delta_time 결측 지시자
수치형	amount	거래 금액, log1p 변환 후 Min-Max 정규화
수치형	is_online	온라인 거래 여부(0/1)
수치형	delta_distance	직전 거래와의 거리(km, 하버사인 공식 적용) Min-Max 정규화
수치형	is_delta_distance_missing	delta_distance 결측 지시자

수치형	months_to_expiry	카드 만료까지 남은 개월 수 Min-Max 정규화
수치형	account_age_days	계정 개설 후 경과일 Min-Max 정규화
수치형	hour, dayofweek, month	거래 발생 시각, 요일, 월
수치형	is_weekend, is_night	주말 및 야간 거래 여부
범주형	mcc_type_encoded	업종 코드
범주형	zip_encoded	우편번호
범주형	merchant_region_encoded	상점 지역 정보
범주형	use_chip_encoded	칩 사용 여부

[표 3-6. 트랜스포머에서 사용한 피처 정리]

생성한 파생 피처 중 delta_time과 delta_distance는 연속된 거래 간의 시간 및 공간적 관계를 포착하는 핵심 피처이다. delta_time은 이전 거래와 현재 거래 간의 시간 차이(초)를 계산하며, delta_distance는 이전 거래와 현재 거래 위치 간의 거리를 하버사인(Haversine) 공식을 사용하여 계산한다. 이 두 피처는 비정상적인 거래 속도나 물리적으로 불가능한 이동을 감지하는 데 중요한 역할을 한다. 이와 함께 is_delta_time_missing과 is_delta_distance_missing 같은 결측 지시자 피처를 추가하여, 첫 거래처럼 이전 거래가 없거나 온라인 거래 등으로 인해 발생하는 결측치(NaN) 정보를 모델이 인식할 수 있도록 하였다.

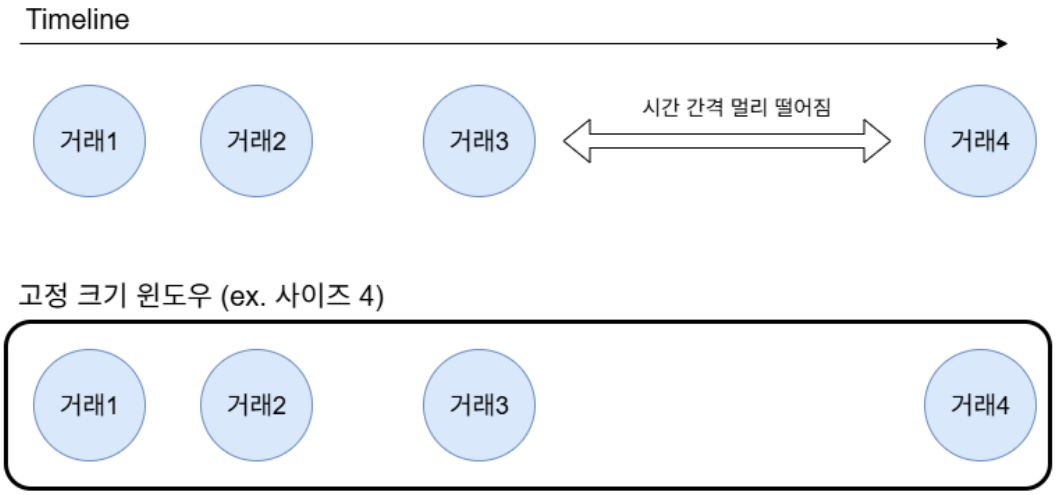
수치형 피처 스케일링에서 거래 금액(amount)은 극단적인 값으로 인해 분포가 한쪽으로 치우칠 수 있으므로 모델이 작은 값의 차이에도 민감하게 반응하도록 분포의 왜도를 완화하기 위해 log(1+x) 변환을 적용하였다. 또한, 수치형 피처들은 각기 다른 값 범위를 가지는데 트랜스포머 모델은 입력 피처의 값 크기에 민감하기 때문에 모델이 큰 값을 가진 피처에만 과도하게 집중하는 문제가 발생할 수 있다. 따라서 모델의 안정적인 학습을 위해 값을 [0, 1] 범위로 정규화하는 MinMax 스케일링을 적용하였다. 스케일러는 학습

데이터셋만을 이용해 학습되었으며, 이렇게 얻은 기준(최솟값, 최댓값)을 검증 및 테스트 데이터셋에 동일하게 적용함으로써 테스트 데이터의 정보가 모델 학습에 영향을 미치는 데이터 유출(Data Leakage)을 방지하였다. 학습 데이터의 범위를 벗어나는 새로운 값이 테스트 데이터에 포함되더라도, 클리핑을 적용해 [0, 1] 범위를 벗어나지 않도록 값을 조정하여 데이터 무결성을 유지하였다.

범주형 피처 인코딩에서 mcc_type, zip과 같은 범주형 피처들은 단순한 정수 인코딩이 아닌 라벨 인코딩(Label Encoding)과 임베딩(Embedding)을 결합하여 처리하였다. 학습 데이터셋에 없던 새로운 범주가 테스트 데이터셋에 포함될 경우를 대비해, LabelEncoder의 클래스에 'UNK' (Unknown) 토큰을 추가하고 이를 0으로 인코딩한 후, 리턴 시에는 패딩 토큰과 구분을 위해 모든 인코딩 값에 1을 더하였다. 즉, UNK는 1로, 기존 범주들은 2부터 시작하는 값으로 변환되었다. 임베딩 레이어는 각 범주형 피처의 고유값 수에 UNK를 포함한 총 개수 + 1을 입력 차원으로 설정하여 패딩 토큰을 구분할 수 있도록 하였다.

3.3.3. 트랜스포머 시퀀스 구성 및 패딩 처리

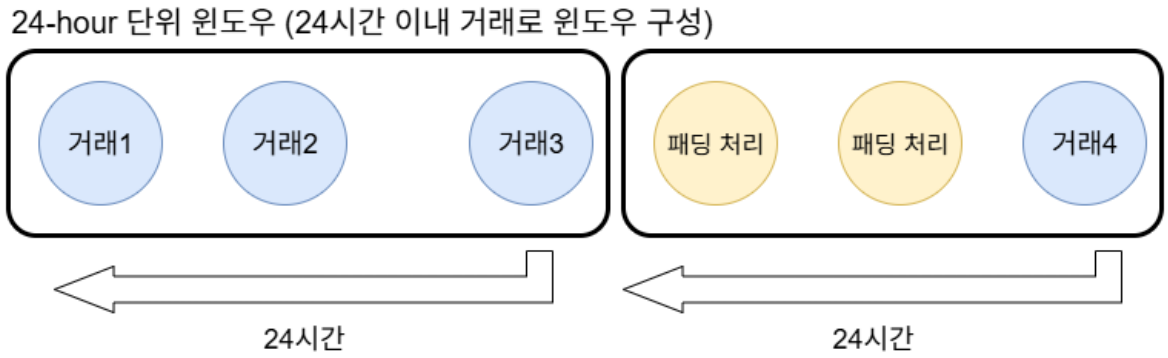
트랜스포머 모델의 입력은 고정된 길이의 시퀀스 형태여야 한다. 본 연구에서 초기에는 모델 학습을 위해 고정 크기 윈도우 방식을 사용하였다.



[그림 3-9. 트랜스포머 윈도우 구성 고정 크기]

그러나 이 방법은 단순히 최근 N건의 거래만을 포함하므로 거래 간의 시간 간격이 제대로 반영되지 않는다. 예를 들어, 한 달에 걸친 거래 12건과 하루 만에 발생한 거래 12건이 동일한 시퀀스로 처리되어 모델이 비정상적인 거래 빈도 패턴을 제대로 학습하지 못하는 문제가 발생한다. 이상 거래는 불규칙하게 발생하며 특히, 짧은 시간 간격 내 연속적인 거래와 같은 패턴에서 그 특성이 드러난다. 따라서 고정 크기 방식은 중요한 시간적 단서를 포착하지 못한다.

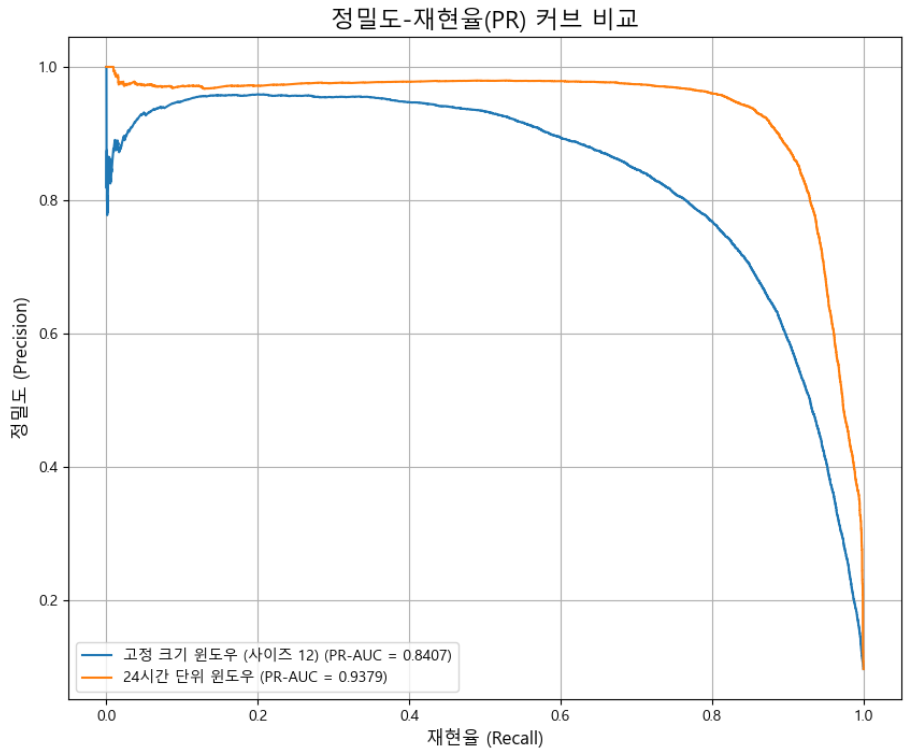
이러한 한계를 개선하기 위해 윈도우 구성 방식을 24시간 단위 윈도우로 변경하였다. 거래 기록을 client_id와 card_id를 기준으로 그룹화한 뒤, 각 거래 시점으로부터 과거 24시간 동안(time_window=timedelta(days=1)) 발생한 거래를 하나의 윈도우로 묶는다.



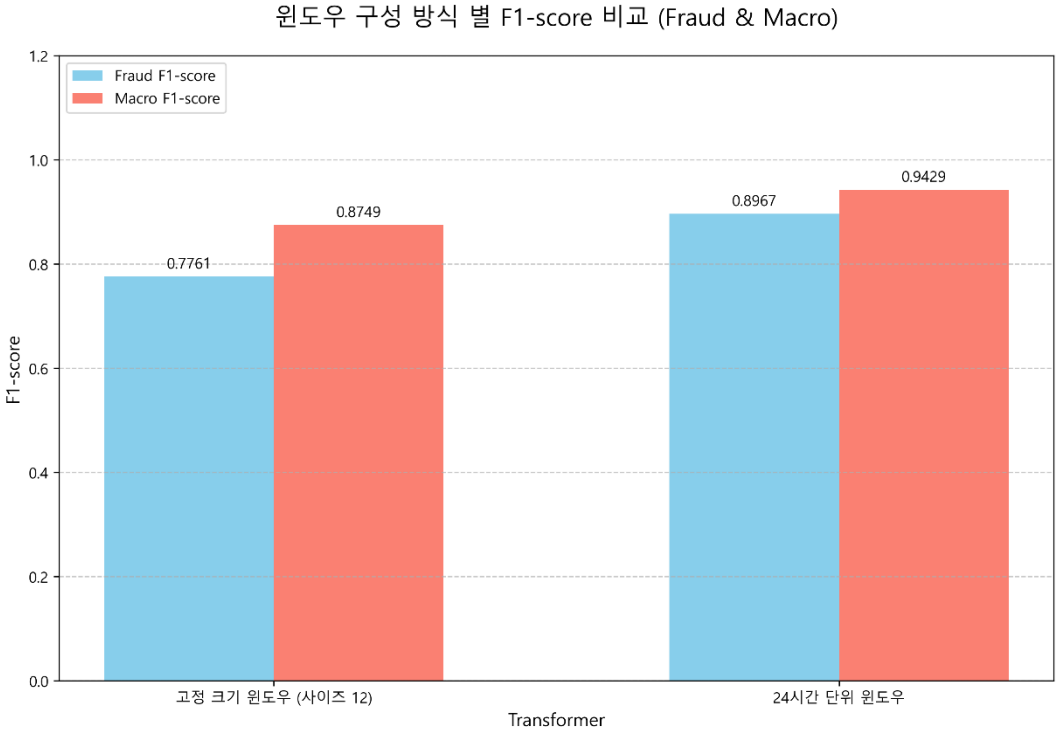
[그림 3-10. 트랜스포머 윈도우 구성 거래 기준 과거 24시간 단위]

24시간 윈도우는 모든 시퀀스가 24시간이라는 일정한 시간적 맥락을 가지므로 모델은 시퀀스 내의 거래 밀집도와 빈도를 효과적으로 학습할 수 있다. 이는 비정상적으로 짧은 시간 내에 여러 거래가 몰아서 발생하는 이상 징후를 효과적으로 탐지하는 데 기여한다. 이러한 개선을 통해 모델은 단순히 거래의 순서를 넘어, 명확한 시간적 맥락 안에서 고객의 정상적인 거래 패턴을 학습하고 비정상적인 시간적 특성을 포착하는 능력을 강화하였다.

이러한 윈도우 구성 방식의 개선이 트랜스포머 모델 성능에 유의미한 향상을 가져왔고 개선 효과를 검증하기 위해 PR 커브와 F1-score를 비교하였다.



[그림 3-11. 트랜스포머 윈도우 구성 방식 별 PR 커브 비교]



[그림 3-12. 트랜스포머 윈도우 구성 방식 별 Fraud & Macro F1-score 비교]

이 윈도우는 이후 모델의 입력 형태인 고정된 길이의 시퀀스로 변환된다. 윈도우 길이가 모델의 시퀀스 최대 길이(max_seq_len=20)보다 짧으면 부족한 만큼 0으로 채우는 패딩(Padding)을 적용하고 윈도우 길이가 max_seq_len보다 길 경우 최신 거래 20건으로 시퀀스를 구성하였다. 이 과정에서 시퀀스 내의 실제 데이터와 패딩된 부분을 구분하기 위해 마스크(mask)를 생성하며 이 마스크는 이후 모델 학습 및 풀링 과정에서 패딩된 부분을 제외하는 데 사용된다.

또한, 윈도우를 생성하고 시퀀스를 구성할 때 후속 모델 학습에서 트랜스포머 임베딩 결과 활용을 용이하게 하기 위해 각 시퀀스에 대한 메타데이터를 함께 저장하였다. 이 메타데이터는 client_id, card_id, 시퀀스에 포함된 처음/마지막 거래일, 시퀀스 내 실제 거래 수(num_transactions_in_window) 등 시퀀스의 기본 정보와 시퀀스의 마지막 거래 시점에 해당하는 원본 피처 값(예: last_amount, last_fraud)을 가진다.

3.3.4. 트랜스포머 모델 구조 및 임베딩 차원

본 연구에서 사용한 TransactionTransformer 모델은 시계열 금융 거래 데이터의 특징을 효율적으로 학습하도록 설계하였다. 이 모델은 PyTorch의 신경망을 구성하는 기본 단위인 nn.Module을 상속받아 정의되었다.

모델 생성자에서 여러 계층과 파라미터들을 정의한다. 범주형 피처를 처리하기 위해 nn.Embedding 레이어들이 사용되는데, 이 레이어는 각 범주를 고차원 벡터로 변환하는 역할을 한다. 이는 단순한 정수 인코딩이 범주 간에 의미 없는 순서 관계를 부여할 수 있는 문제를 해결하기 위함이다. 예를 들어, mcc_type과 zip은 16차원, region은 8차원, use_chip은 4차원 임베딩으로 설정되었다. nn.Embedding의 padding_idx=0는 패딩 토큰(0)에 해당하는 임베딩 벡터를 0으로 고정하여 학습에 영향을 주지 않도록 한다.

수치형 피처의 개수와 범주형 피처 임베딩 차원을 합산한 input_dim_adjusted를 계산한 뒤, nn.Linear 레이어인 self.embed를 통해 모델의 내부 차원인 model_dim=64로 선형 투사하여 모든 입력 피처를 트랜스포머 인코더가 처리할 수 있는 동일한 차원으로 맞춘다.

트랜스포머 인코더는 nn.TransformerEncoderLayer를 기반으로 nn.TransformerEncoder를 구성한다. d_model=64, nhead=4, num_layers=2로 설정된 encoder_layer는 각 거래 간의 복잡한 관계를 학습하는 데 사용된다. num_heads=4는 모델이 4개의 다른 관점에서 어텐

션 가중치를 계산하게 하여 다양한 관계를 포착하도록 돕고, num_layers=2는 모델의 깊이를 적절히 유지하여 복잡한 패턴을 학습하게 한다.

self.embedding_head는 트랜스포머 인코더의 출력을 32차원의 최종 임베딩 벡터로 변환하는 nn.Linear 레이어이고, self.classification_head는 이 임베딩을 기반으로 2개의 클래스(정상/사기)에 대한 예측 로짓을 출력하는 역할을 한다.

TransactionTransformer 클래스의 forward 메서드는 모델의 데이터 처리 흐름을 정의한다. 먼저, 범주형 피처는 각 임베딩 레이어를 통과하여 벡터로 변환되고, 이 벡터들은 수치형 피처와 함께 하나의 입력 텐서로 결합된다. 이 결합된 입력 텐서는 self.embed 레이어를 거쳐 model_dim 차원으로 조정된 후, self.transformer 인코더에 전달된다. 이때, 패딩된 위치를 무시하도록 src_key_padding_mask 인자에 마스크를 전달한다.

인코더를 통과한 시퀀스 출력은 마스킹된 평균 풀링(Masked Average Pooling)을 거쳐 시퀀스 전체를 대표하는 단일 벡터로 압축된다. 이 풀링 과정에서 마스크를 사용하여 패딩된 위치의 값들은 계산에서 제외된다.

최종적으로, 풀링된 벡터는 self.embedding_head를 통과하여 embedding이라는 32차원 임베딩 벡터로 변환된다. 이 embedding은 후속 모델의 입력으로 활용되는 핵심적인 시계열 특징 표현이다. 이어서 embedding은 self.classification_head를 거쳐 logits를 생성한다. logits는 각 클래스에 대한 예측 점수를 나타내며, 손실 함수 계산 및 최종 분류 예측에 사용된다.

3.3.5. 트랜스포머 모델 학습 및 EarlyStopping

트랜스포머 모델 학습에서 데이터의 심각한 클래스 불균형 문제를 완화하기 위해 학습 데이터의 정상 거래와 사기 거래 개수의 역수를 가중치로 설정해 CrossEntropyLoss 함수를 적용하여 모델이 소수 클래스를 더 중요하게 학습하도록 하였다.

또한, 학습 중 기울기(gradient)가 특정 값 이상으로 커지는 현상인 기울기 폭발(exploding gradient)을 막기 위해 기울기 클리핑(gradient clipping)을 적용하여 모델 학습의 안정성을 높였다.

트랜스포머 모델 학습 전에 학습 과정 중 EarlyStopping을 위한 loss 계산에서 테스트 데이터가 사용되는 방지하기 위해 원본 거래 데이터를 증강 이후 학습용/테스트용으로 분

할했던 것과 같은 방법으로 트랜스포머 학습용 데이터를 다시 학습용/검증용으로 분할하여 EarlyStopping을 위한 loss 계산 시 검증용 데이터 시퀀스를 사용해 학습 과정에서 테스트 데이터가 사용되는 것을 방지하였다.

EarlyStopping은 매 에포크마다 검증 데이터셋에 대한 손실(val_loss)을 측정하여 best_loss와 비교하고 이 값이 min_delta=0.001 이상으로 낮아지면 해당 시점의 모델 가중치를 저장하고 EarlyStopping 카운터를 0으로 초기화한다. val_loss가 best_loss보다 min_delta 이상 낮아지지 않으면 카운터를 증가시킨다. 카운터가 patience를 초과하면 학습을 조기 종료하고, 검증 손실이 가장 낮았던 시점의 모델 가중치를 저장한다.

3.4. 이상 거래 탐지 모델 설계 및 구현

3.4.1. MLP 모델

기존의 사기 거래 탐지 연구들은 단일한 관점에 치중하여 금융 거래 데이터의 복합적 특성을 충분히 반영하지 못하는 한계를 보였다. 예컨대 RNN 및 LSTM 모델은 거래의 시계열적 순서에는 강점을 보였으나 정적 특성이나 장기적 행동 패턴은 간과하는 경향이 있었고, Isolation Forest나 One-Class SVM과 같은 이상치 탐지 기법은 통계적 이상값은 포착하더라도 시간적 맥락과 개인별 행동 변화를 고려하지 못해 오탐 가능성이 높았다. 전통적인 특징 엔지니어링 방식 역시 전문가 지식에 의존적이어서 새로운 유형의 사기 패턴이나 환경 변화에 대한 적응력이 부족했다.

본 모델은 이를 개선하고자 MLP의 유연성과 비선형 학습 능력에 주목하였다. MLP는 다양한 형태의 입력 특성을 효과적으로 통합할 수 있어, 표현 학습 · 시계열 맥락 · 주기적 행동 · 통계적 이상 탐지라는 서로 다른 관점을 반영할 수 있다. 이에 따라, 트랜스포머 기반 임베딩을 출발점으로 하여 네 가지 MLP 기반 접근법(Method 1~4)을 고안하였으며, 이후 실험을 통해 각 방법의 성능과 효율성을 비교 · 검증하였다.

3.4.1.1. 초기 4가지 모델

금융 거래 데이터의 복합적인 특성을 다각도로 반영하기 위하여 네 가지 차별화된 MLP 기반 접근법을 설계하였다. 각 방법론은 동일한 트랜스포머 임베딩을 출발점으로 하되, 서로 다른 분석 관점에 초점을 맞추어 다양한 사기 패턴을 포착하도록 설계되었다.

Method 1 트랜스포머 임베딩 + 시계열 특징 집계

트랜스포머를 통해 추출한 32차원 거래 시퀀스 임베딩에 고객·카드 수준의 집계 메타데이터(29차원)를 결합하여 총 61차원의 입력 벡터를 구성한다. 이는 시퀀스 표현 학습과 전통적인 요약 통계를 동시에 활용함으로써 개별 거래 맥락과 장기적 거래 특성을 함께 반영한다.

Method 2 메타데이터 활용 윈도우 평탄화

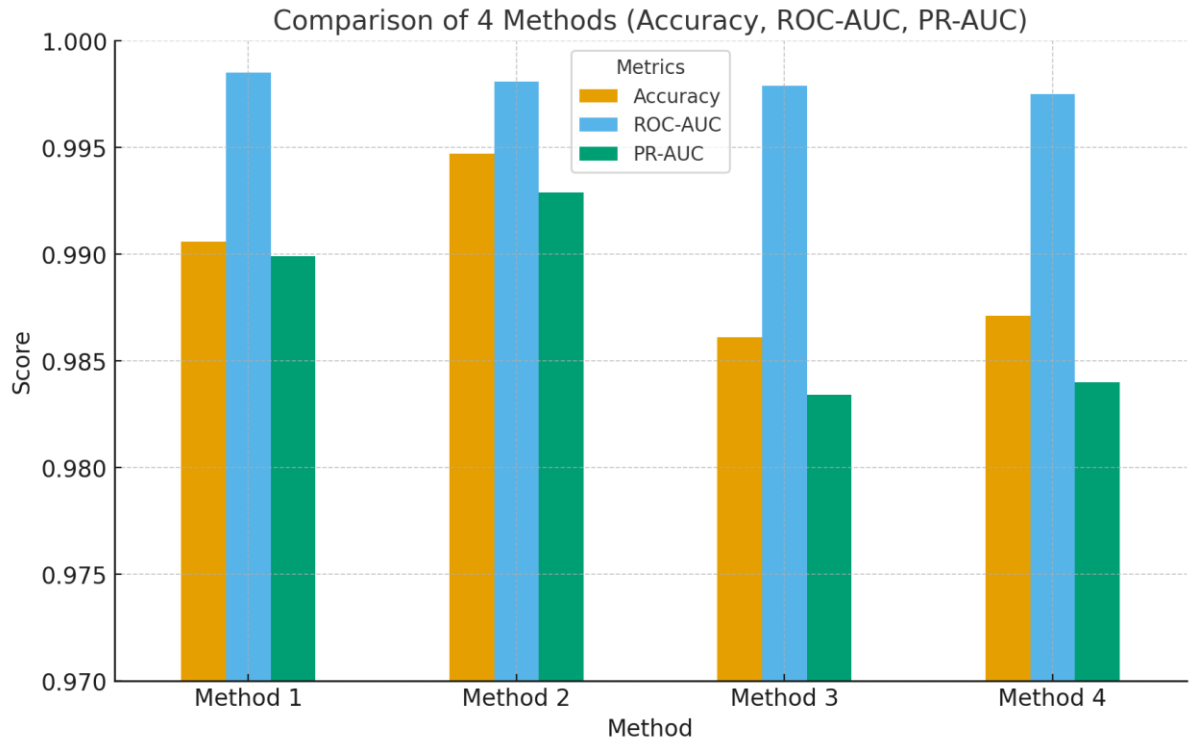
거래 데이터를 연속된 시간 윈도우 단위로 분할한 뒤, 각 윈도우의 13개 메타 특징을 추출하고 이를 5개 구간에 대해 평탄화하여 총 65차원의 입력으로 변환한다. 이 방식은 순차적 거래 패턴을 MLP가 직접 학습할 수 있도록 하여 시계열적 연속성을 모델링하는 데 중점을 두었다.

Method 3 메타데이터 활용 윈도우 평탄화

거래 데이터를 연속된 시간 윈도우 단위로 분할한 뒤, 각 윈도우의 13개 메타 특징을 추출하고 이를 5개 구간에 대해 평탄화하여 총 65차원의 입력으로 변환한다. 이 방식은 순차적 거래 패턴을 MLP가 직접 학습할 수 있도록 하여 시계열적 연속성을 모델링하는 데 중점을 두었다.

Method 4 이상 패턴 감지 특징

개인별 정상 거래 분포 대비 편차를 정량화하여 이상 정도를 측정하는 접근법이다. Z-score, IQR, 백분위수 기반의 통계 지표를 활용하여 총 35차원의 이상 패턴 특징을 구성하였으며, 이는 거래 행태의 미묘한 변화를 포착해 통계적으로는 드물지만 사기 가능성이 높은 패턴을 탐지하는 데 기여한다.



[그림 3-13. MLP 기반 4가지 모델 성능 비교]

	Accuracy	ROC-AUC	PR-AUC	Δ*시간*
Method 1	0.9906	0.9985	0.9899	~5분
Method 2	0.9947	0.9981	0.9929	~7분
Method 3	0.9861	0.9979	0.9834	~46분
Method 4	0.9871	0.9975	0.9840	~13분

[표 3-7. MLP 기반 4가지 모델 성능 비교]

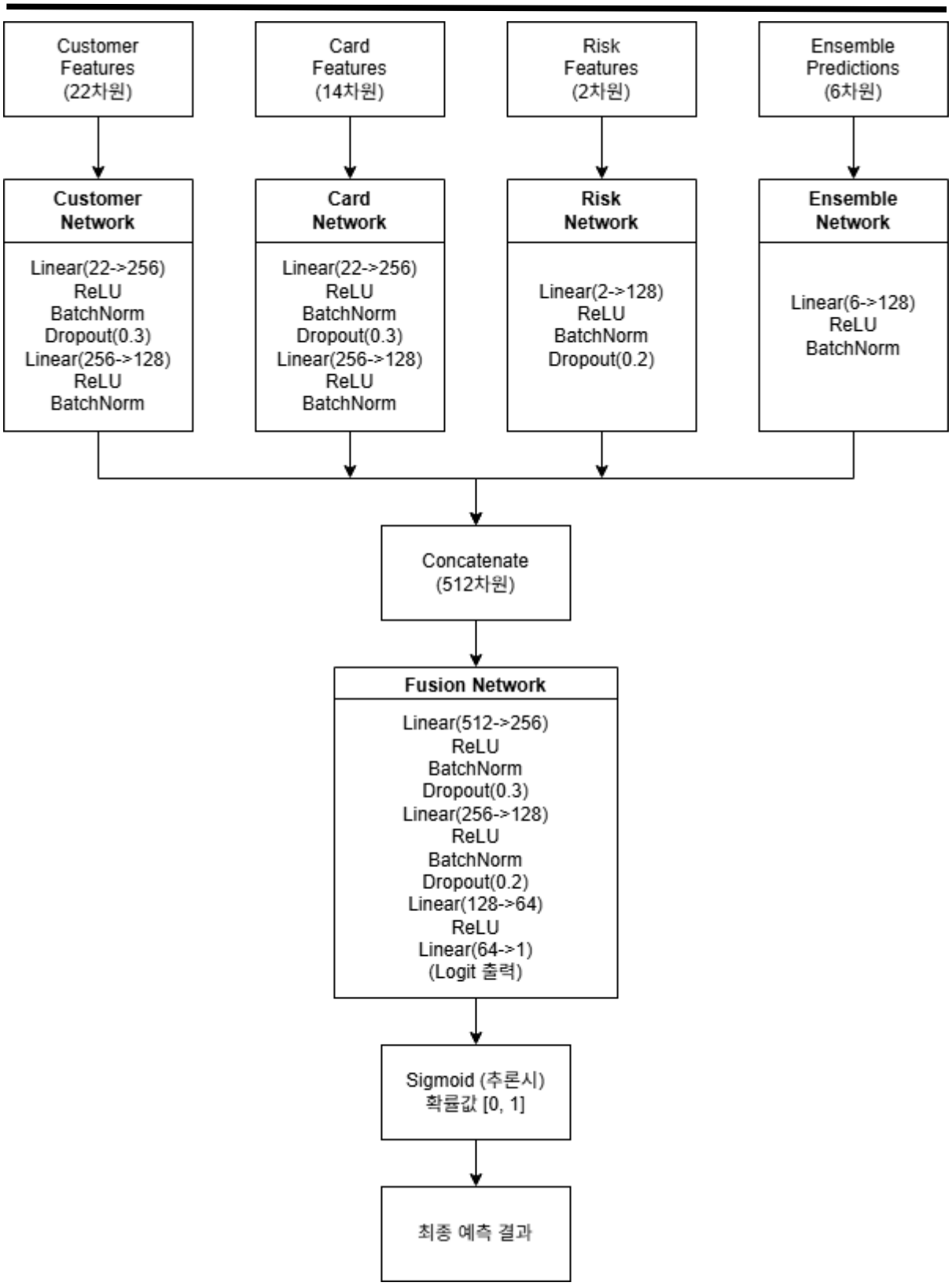
모델 성능 평가 결과, 제안된 네 가지 MLP 접근법 중 Method 1, Method 2, Method 4가 모두 안정적이고 높은 성능을 보였다. Method 1(트랜스포머 임베딩 + 집계 특성)은 ROC-AUC 0.9985, PR-AUC 0.9899로 우수한 균형 성능을 나타내며, 약 5분 내외의 처리시간으로 실용성이 뛰어났다. Method 2(메타데이터 윈도우 평탄화)는 Accuracy 99.47%와 PR-AUC 0.9929로 가장 높은 정확도와 정밀한 분류 성능을 기록하여 시계열 패턴 학습의 효과를 입증하였다. 또한 Method 4(이상 패턴 감지)는 Accuracy 98.71%, ROC-AUC 0.9975를 기록하며, 통계적 이상 탐지를 통한 안정적이고 해석 가능한 결과를 제공하였다. 반면 Method 3(시간 구간 분할)은 상대적으로 낮은 정확도와 과도한 처리시간(46분)을 보여

최종 시스템에서 제외되었다. 따라서 본 연구의 최종 모델은 Method 1, Method 2, Method 4를 통합하는 방향으로 설계되었다.

3.4.1.2. 최종 모델

최종적으로는 트랜스포머 임베딩을 기반으로 도출된 세 가지 방법(Method 1, 2, 4)을 통합하고, 여기에 고객·카드·리스크 관련 정보를 함께 반영하는 앙상블 네트워크를 설계하였다. 이 모델은 입력을 네 가지 그룹(고객, 카드, 리스크, 앙상블 예측 결과)으로 나누어 각각 별도의 서브 네트워크에서 처리한 뒤, 통합 네트워크(fusion network)에서 결합하여 최종적으로 사기 거래 여부를 판별한다.

서브 네트워크는 입력 특성의 성격에 따라 차별화되었다. 고객 및 카드 네트워크는 2개의 은닉층으로 구성되어 있고, 리스크 네트워크는 상대적으로 작은 입력 차원에 맞추어 얕은 구조로 설계되었다. 앙상블 네트워크는 외부 모델의 예측 확률(총 6차원 입력)을 처리하는 구조를 가진다. 모든 서브 네트워크의 출력은 동일한 차원으로 축소된 후 결합(torch.cat)되며, 통합 네트워크는 이 결합된 특징을 입력받아 3단계 비선형 변환(Linear → ReLU → BatchNorm → Dropout)을 거쳐 은닉 차원을 64로 축소하고, 최종적으로 단일 로짓(logit)을 출력한다.



[그림 3-14. MLP 모델 흐름도]

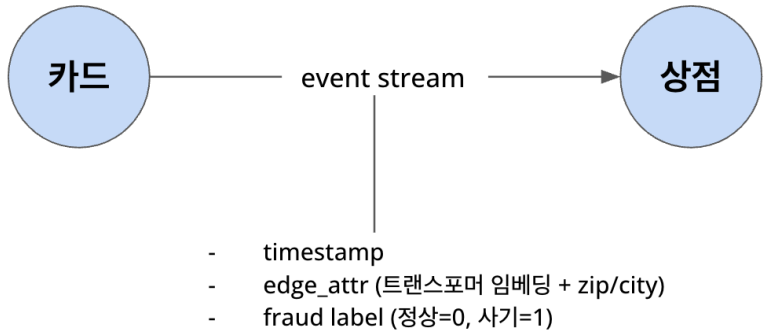
또한, 트랜스포머 임베딩 외에도 추가적인 특징 엔지니어링을 통해 데이터 표현력을 확장하였다. 여기에는 거래 시각, 요일, 주말 여부, 심야 거래 여부와 같은 시간 기반 특징, 소득 대비 부채 비율과 신용 점수 구간화 등 고객 프로필 특징, 최근 거래 금액 통계와 신용 한도 대비 사용률 등 카드 사용 패턴, 기본 및 복합 리스크 점수와 같은 리스크 지표, 거래 일관성·활동 다양성·거래 속도 등 행동 패턴 특징이 포함된다. 이러한 통합 피처 세트는 단순 임베딩만 사용하는 경우보다 더 풍부한 맥락 정보를 제공한다.

손실 함수로는 BCEWithLogitsLoss를 기본으로 사용하되, 클래스 불균형을 보정하기 위해 양성 비율 기반의 가중치(pos_weight)를 적용하였다. 대안으로는 Focal Loss를 도입하여 쉬운 샘플보다 어려운 샘플에 학습이 집중되도록 하였다. 최적화에는 Adam(학습률 0.001, weight_decay=1e-5)을 사용하였고, ReduceLROnPlateau 스케줄러(patience=10, factor=0.5)를 통해 검증 성능이 정체될 경우 학습률을 단계적으로 감소시켰다. 과적합 방지를 위해 Batch Normalization과 Dropout(0.2~0.3)을 적용했으며, Early Stopping(patience=20)을 도입하여 검증 성능이 일정 기간 개선되지 않으면 학습을 조기 종료하여 최적의 가중치를 보존하였다.

3.4.2. TGN 모델

TGN 모델 구현에 사용된 라이브러리는 다음과 같다. dataclasses, typing, PyTorch, pandas, Numpy, scikit-learn, matplotlib, tqdm 그리고 TGN[4]. TGN의 경우 TGN 논문 저자가 깃허브에 공개한 모델 구현을 클론하여 학습에 사용하였다.

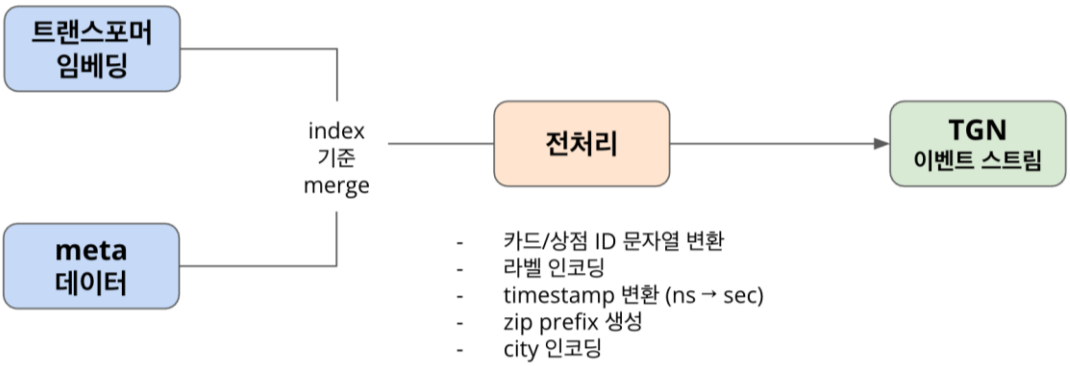
3.4.2.1. 모델 입력을 위한 그래프 구성:



[그림3-15. TGN 모델 입력을 위한 그래프 특성]

가장 먼저 TGN 모델에 입력하기 위해 거래 데이터를 그래프로 변환하였다. 그래프의 노드는 카드 노드와 상점 노드로 구별했으며, 이는 신용카드 거래가 본질적으로 카드 →

상점 관계로 정의되기 때문이다. 카드 노드는 개별 고객/카드의 사용 주체를, 상점 노드는 거래가 발생하는 대상(가맹점)을 나타낸다. 각 거래 이벤트는 카드 노드에서 상점 노드로 향하는 directed edge로 표현하였다. Edge에는 각 거래 발생 시점과 임베딩 피처, fraud 라벨을 부여하여 TGN 학습에 필요한 이벤트 스트림을 구축하였다.



[그림 3-16. 임베딩 및 메타데이터 기반의 이벤트 스트림 생성 과정]

이벤트 스트림은 시간에 따라 발생하는 개별 거래를 순차적으로 나열하여 생성하였다. 전처리 과정에서 카드 ID와 상점 ID를 문자열로 변환하여 일관성을 확보하였다. 카드 ID와 상점 ID가 범주형 데이터이므로 모델에서 수치적 의미를 갖지 않도록 모든 ID를 문자열로 변환한 후 라벨 인코딩을 적용하여 인덱스로만 사용하도록 하였다. 거래 시각은 datetime 형식으로 변환한 후 유효하지 않은 값을 제거하였다. 또한 우편번호의 앞 3자리와 상점 도시명을 라벨 인코딩하여 이벤트 단위의 추가 피처로 활용하였다.

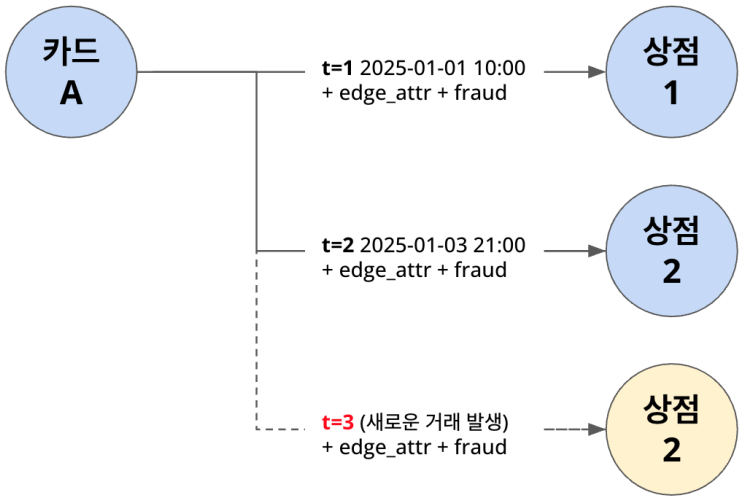
생성된 이벤트 스트림은 TGN 모델에 입력하기 위해 거래 발생 시간순으로 정렬하였다. 이후 각 데이터셋의 노드 인덱스, 거래 시각, 엣지 피처, 사기 여부 라벨을 PyTorch 텐서로 변환하여 모델 학습이 가능하도록 하였다.

	Tensor 이름	내용	자료형	예시 값
0	train_src	출발 노드 (카드 인덱스)	torch.int64	[19, 2834, 2806]
1	train_dst	도착 노드 (상점 인덱스)	torch.int64	[9318, 12860, 6566]
2	train_ts	거래 발생 시각 (초 단위)	torch.float32	[1262304128.0, 1262306048.0, 1262307968.0]
3	train_eX	엣지 피처 (임베딩+추가 특성)	torch.float32	[[1.7298113107681274, -2.66027569770813, 1.328...
4	train_y	라벨 (정상=0, 사기=1)	torch.float32	[0.0, 0.0, 0.0]

[표 3-8. 이벤트 스트림 PyTorch 텐서 변환 결과]

TGN 모델에서는 인접 노드를 탐색할 때 단순한 연결 관계뿐만 아니라 시간 정보를 함

께 고려해야 한다. NeighborFinder는 특정 카드 노드가 과거에 어떤 상점 노드와 언제 연결되었는지를 기록하고, 새로운 거래 이벤트가 발생했을 때 해당 시점의 이전 이웃을 효율적으로 조회하는 역할을 수행한다.



[그림 3-17. NeighborFinder를 활용한 과거 거래 이웃 조회 예시]

예를 들어, 카드 A가 상점 1, 2와 각각 시점 t=1, t=2에서 거래한 이력이 있다면, t=3 시점의 새로운 거래가 발생했을 때 NeighborFinder는 t=1, t=2의 과거 거래를 메모리에서 불러와 이웃 정보로 활용한다.

TGN 모델에 입력되는 노드 피처는 아래와 같이 카드 노드와 상점 노드를 구분하기 위한 정보를 포함하도록 설계하였다. 메모리 모듈과의 차원을 맞추기 위해 전체 노드 수(num_nodes)와 메모리 차원(mem_dim) 크기(num_nodes x mem_dim)의 행렬을 생성한 뒤, 카드 노드에는 첫 번째 열에 1.0 값을 부여하고, 상점 노드에는 두 번째 열에 1.0 값을 부여하였다.

이를 통해 모델이 카드 노드와 상점 노드를 구분할 수 있게 하고, 나머지 열은 0으로 초기화하였다. 0으로 초기화한 열은 차원을 맞추기 위한 패딩 역할을 하지만, 학습 과정에서 TGN의 메모리 모듈이 노드 피처와 메모리 벡터를 더하며 해당 열을 갱신하며 의미 있는 값으로 변환된다. 이러한 초기화 방식을 통해 모델은 카드 → 상점 거래 관계를 학습할 때 노드 타입을 구분하고 지금까지 발생한 모든 거래들의 패턴을 함께 반영할 수 있게 된다.

카드 노드 예시 (card flag=1):

	dim0	dim1	dim2	dim3	dim4	dim5	dim6	dim7	dim8	dim9	...	dim52	dim53	dim54	dim55	dim56	dim57	dim58	dim59	dim60	dim61
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

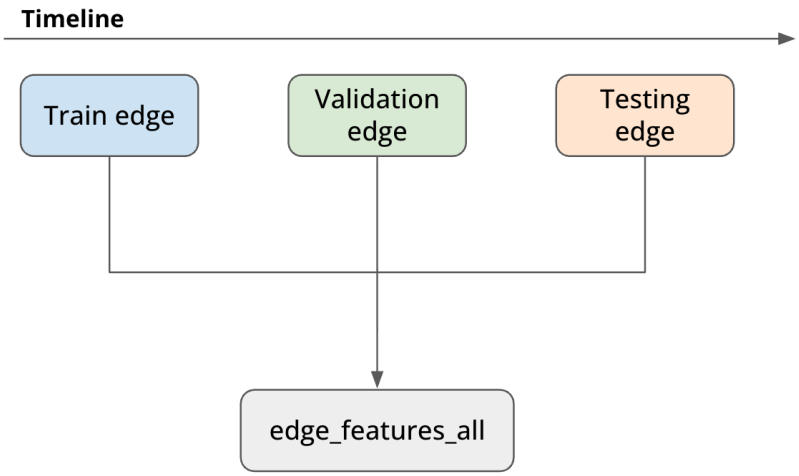
[표 3-9. 카드 노드 피처 벡터 (첫 번째 차원에 flag=1 설정)]

상점 노드 예시 (merchant flag=1):

	dim0	dim1	dim2	dim3	dim4	dim5	dim6	dim7	dim8	dim9	...	dim52	dim53	dim54	dim55	dim56	dim57	dim58	dim59	dim60	dim61
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[표 3-10. 상점 노드 피처 벡터 (두 번째 차원에 flag=1 설정)]

TGN 모델에 입력되는 엣지 피처는 시계열 정보가 중요한 TGN 모델의 특성상, train edge, validation edge 그리고 test edge는 시간 순으로 붙어 있다. 따라서 모델 초기화 시에 아래와 같이 train edge, validation edge와 test edge를 합한 벡터를 입력하지만, train 루프와 validation 루프, test 루프 내에서는 각각 train_eX_scaled, val_eX_scaled, test_eX_scaled를 별도로 사용하고 있다. 따라서 학습 시 테스트 데이터가 섞이는 것이 방지된다.



[그림 3-18. Train/Validation/Test 엣지 피처 분할 및 통합 과정]

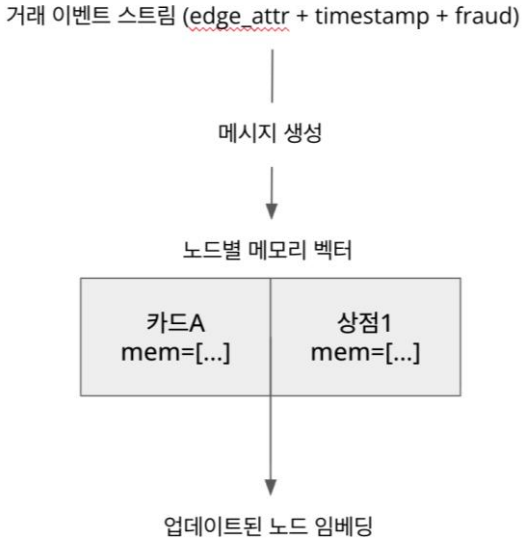
3.4.2.2. TGN 모델 옵션:

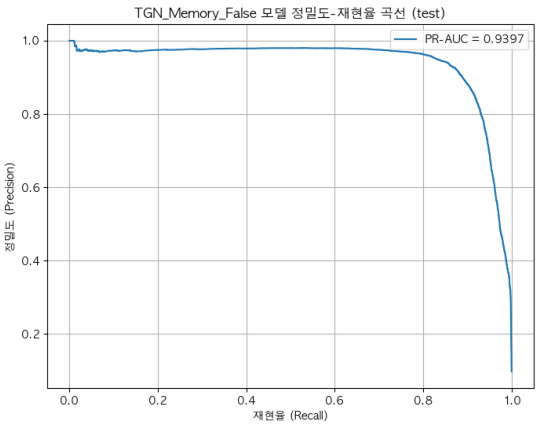
메모리 모듈 옵션에서 use_memory=True를 설정하면, TGN의 핵심 모듈인 메모리가 활성화된다. 각 노드는 고유한 메모리 벡터를 유지하며, 과거 발생한 거래 정보를 시간 순으로 누적·갱신한다. 이때 TGN에서의 메시지(message)란 이벤트가 발생할 때 노드 메모리를 갱신하기 위해 생성되는 정보로, 어떤 노드가 언제 어떤 속성(edge_attr 등)으로 연결되었는지를 조합하여 만들어진다. 이러한 과정을 통해 모델은 단일 거래뿐만 아니라

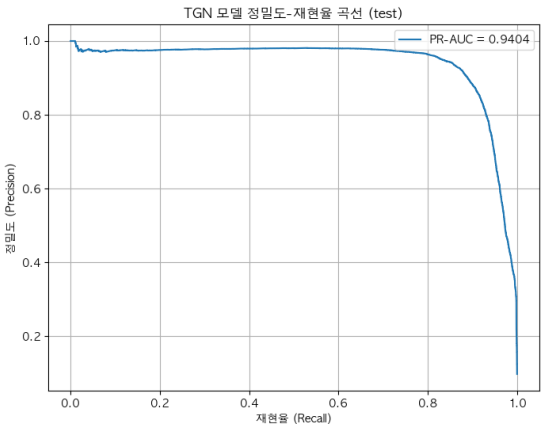
노드별 과거 거래 이력까지 반영한 임베딩을 학습할 수 있다.

반면에 use_memory=False로 설정하면 메모리 모듈을 사용하지 않으면 과거 거래 정보는 반영되지 않고, 현 시점의 거래 정보만으로 노드 임베딩이 생성된다. 따라서 거래 이벤트가 독립적이고 강한 시계열 패턴이 존재하지 않는 데이터의 경우에는 오히려 단순한 구조(use_memory=False)에서 더 잘 일반화될 수 있다.

<table>
<tr><th>use_memory = False</th><th>use_memory = True</th></tr>
<tr><td>


</td><td>


</td></tr>
<tr><td>


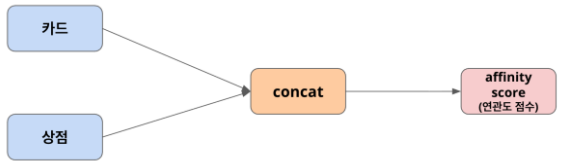
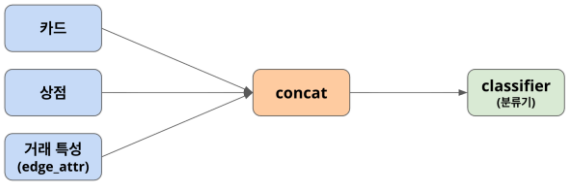
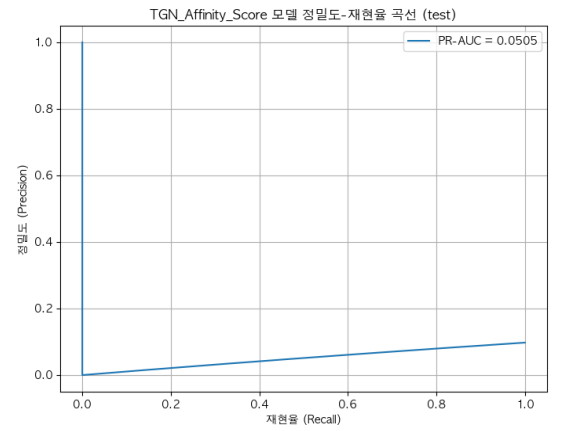
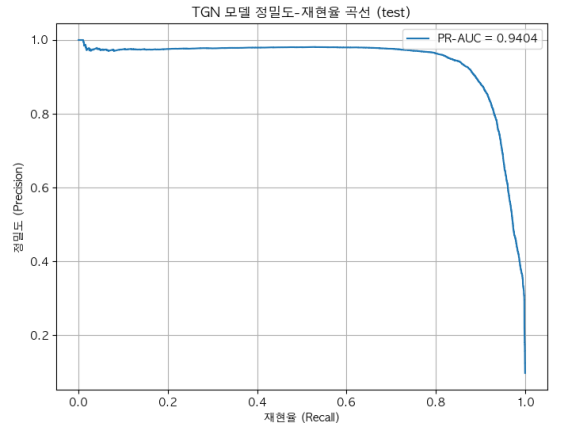
</td><td>


</td></tr>
</table>

[표 3-11. use_memory=False, use_memory=True 비교 (PR-AUC 커브)]

3.4.2.3. TGN 모델 학습:

TGN 모델이 사기 거래를 얼마나 잘 예측하는지 검사하기 위해 두 가지 방식을 비교하였다. 초기에는 TGN 기본 코드에 구현되어 있는 affinity score 모듈을 그대로 사용하였다. 이는 카드-상점 노드 임베딩을 결합하여, 이 조합이 과거 사기 거래 패턴과 얼마나 유사한지를 점수화한 뒤 사기 여부를 예측한다. 그러나 임베딩 결합 만으로는 거래 금액, 위치 정보 등 거래의 맥락을 충분히 반영하기 어렵다는 한계를 확인하였다. 따라서 노드 임베딩에 추가적인 거래 특성을 결합한 분류기(fraud classifier)를 도입하여 성능을 개선해 보고자 하였으며, 실험 결과 fraud classifier가 더 높은 성능을 보였다.

Affinity score	Fraud classifier
	
	

[표 3-12. Affinity Score, Classifier 비교 (PR-AUC 커브)]

순전파 과정 비교:

단계	Affinity score 방식 (초기)	Fraud Classifier 방식 (개선)
1. 시계열 임베딩 계산	model.compute_temporal_embeddings( )으로 카드, 상점 노드 임베딩 생성	Affinity score 방식과 동일
2. 엣지 표현 생성	카드, 상점 임베딩 두 개를 결	카드, 상점 임베딩 + 이벤트

	합	스트림의 거래 특성을 결합하여 edge_repr 생성
3. 예측	affinity_score(src_emb, dst_emb) → sigmoid → 사기 여부 확률	fraud_classifier(edge_repr) → sigmoid → 사기 여부 확률
특징	단순히 노드 임베딩 유사도로 판정	거래 맥락을 함께 반영한 판정

[표 3-13. Affinity Score, Classifier 순전파 과정 비교]

여기서 사용된 compute_temporal_embeddings 함수는 TGN 모델의 핵심 모듈로, 특정 시점의 카드 노드와 상점 노드 쌍에 대한 시계열 임베딩을 계산한다. 시계열 임베딩은 시간에 따라 변하는 노드의 상태와 과거 이웃 노드들의 상호작용을 반영한 표현으로, 이를 통해 모델이 과거 거래 패턴의 연속성을 학습하게 된다.

손실 계산:

- 예측 결과와 실제 라벨을 비교하여 로스를 계산

역전파 및 파라미터 업데이트:

- loss.backward()를 통해 기울기 계산
- 옵티마이저로 모델 파라미터 갱신

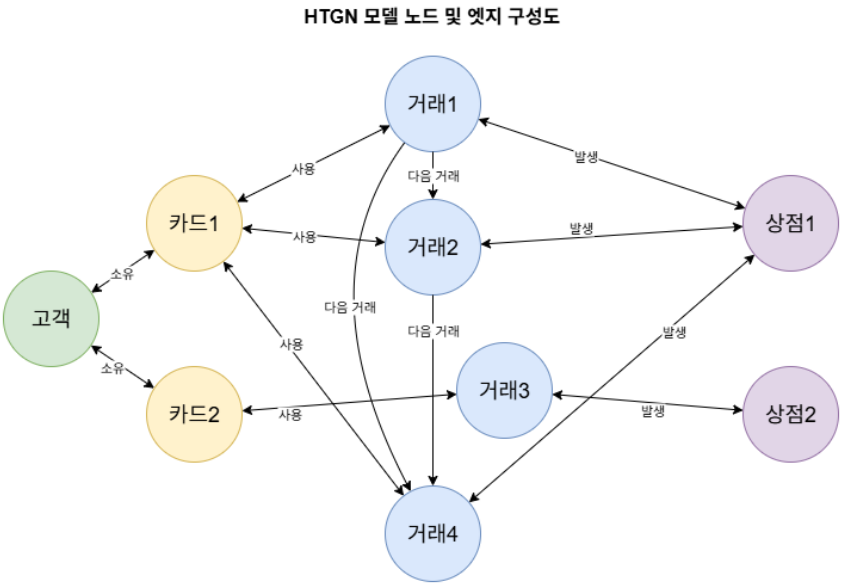
3.4.3. HTGN 모델

복잡하고 다양한 관계를 포함하는 거래 데이터의 특성을 반영하기 위해 이종 그래프 신경망(Heterogeneous Graph Neural Network, HTGN) 모델을 적용하였다. HTGN은 단순히 거래 내역을 분석하는 것을 넘어, 거래를 둘러싼 여러 개체(고객, 카드, 상점) 간의 복잡한 연결을 종합적으로 고려함으로써 이상 거래 탐지 성능을 향상시킨다.

3.4.3.1. HTGN 모델 노드 및 피처 정의

데이터는 PyG(PyTorch Geometric) 라이브러리의 HeteroData 객체를 활용하여 customer, card, transaction, merchant의 네 가지 노드 타입과 이들 사이의 관계를 정의하였다. 각

노드에는 고유한 피처가 부여되며, 특히 transaction 노드는 트랜스포머 모델을 통해 생성된 시계열 임베딩을 사용하여 거래의 시간적 속성을 반영하였다.



[그림 3-19. HTGN 모델 구조도]

고객(customer) 노드

- 수치형 피처: current_age, yearly_income, total_debt, credit_score, num_credit_cards는 StandardScaler를 사용하여 표준화하고 극단적인 이상치의 영향을 줄이기 위해 -3에서 3 사이로 클리핑(clip(-3, 3))을 적용하였다.
- 범주형 피처: gender는 LabelEncoder를 사용하여 정수 인코딩되었다.

카드(card) 노드:

- 수치형 피처: num_cards_issued, credit_limit, year_pin_last_changed는 StandardScaler를 사용하여 표준화 및 클리핑 처리하였다.
- 범주형 피처: card_brand, card_type, has_chip은 LabelEncoder를 사용하여 정수 인코딩되었다.

거래(transaction) 노드:

- 이상 거래 탐지의 목표 노드

- 32차원 트랜스포머 시계열 임베딩과 last_amount, num_transactions_in_window와 같은 메타데이터를 포함한다.
- 수치형 피처: 트랜스포머 임베딩(32차원), last_amount, num_transactions_in_window는 StandardScaler를 사용하여 표준화 및 클리핑 처리하였다.

상점(merchant) 노드:

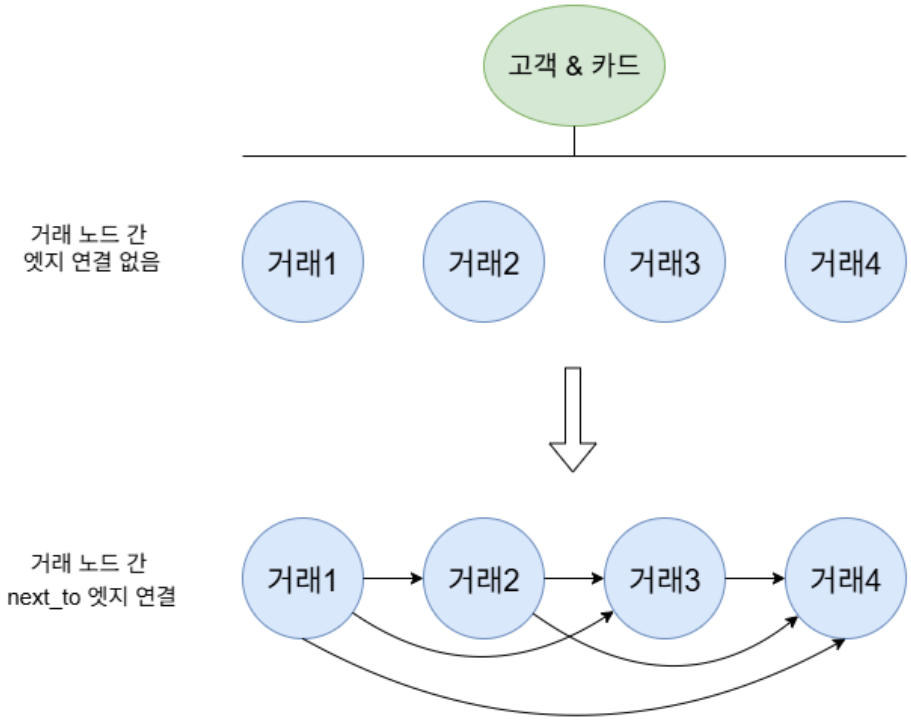
- 범주형 피처: 상점 ID를 nn.Embedding 레이어의 입력으로 변환하였다.

범주형 피처는 LabelEncoder로 인코딩하되, 학습 시 보지 못한 새로운 범주 처리를 위해 'UNK' 토큰을 추가하였다. 또한 학습 데이터 기준으로 생성된 스케일러와 인코더를 저장하여 테스트 데이터에도 동일하게 적용, 데이터 일관성을 확보하였다.

3.4.3.2. HTGN 모델 엣지 구성 및 next_to 엣지 도입

엣지는 다음과 같이 정의하였다.

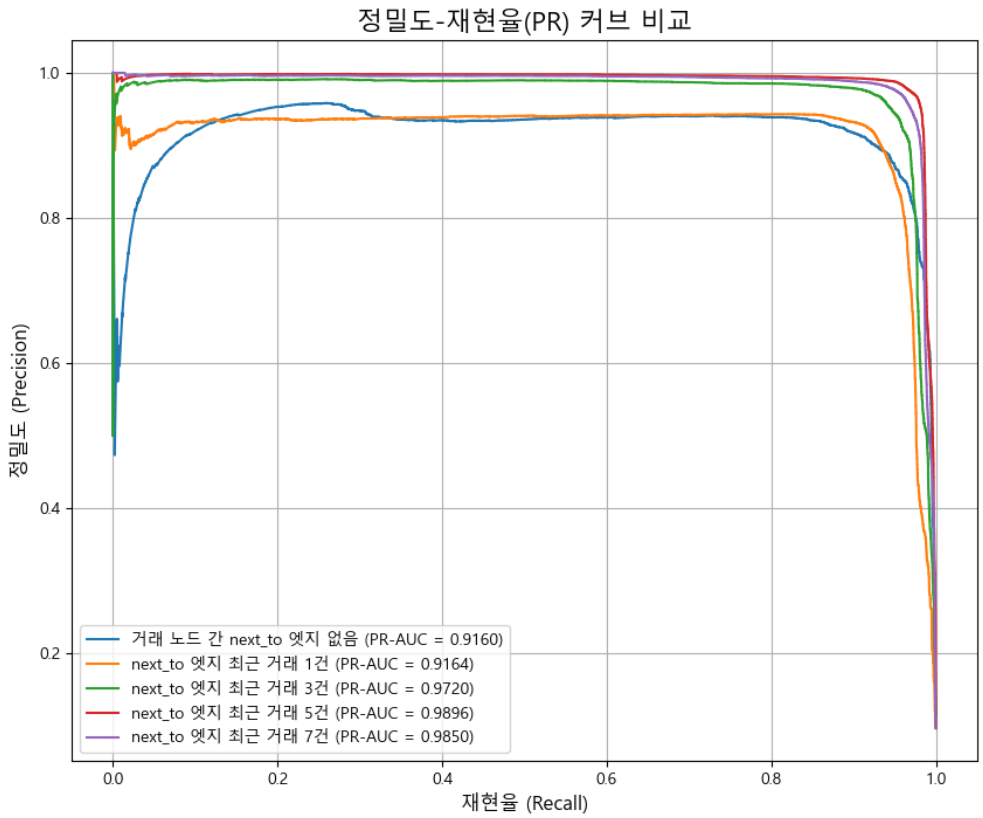
- customer ↔ card: 고객과 카드의 소유 관계
- card ↔ transaction: 카드가 거래에 사용된 관계
- transaction ↔ merchant: 거래가 발생한 상점 관계
- transaction ↔ transaction: 동일 고객의 거래 시계열 관계



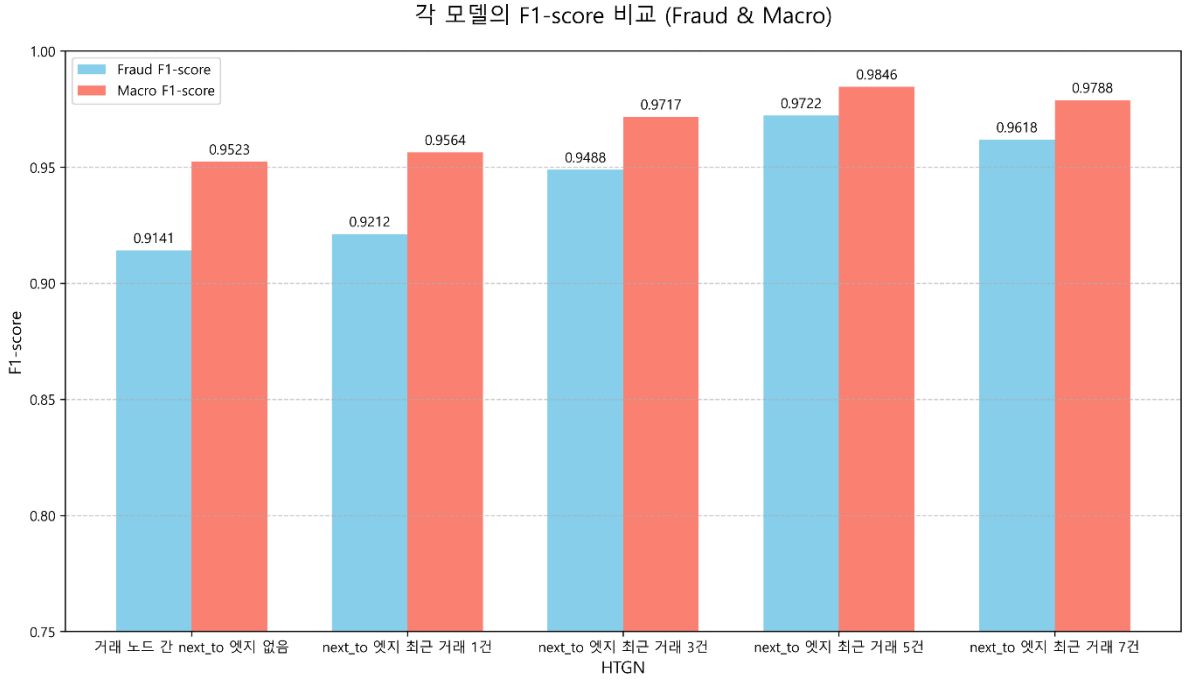
[그림 3-20. HTGN 모델 next_to 엣지 연결 전/후 구조]

본 연구의 핵심 아이디어는 거래 노드 간의 시간적 순서를 그래프 구조에 직접 반영하는 것이다. 이를 위해 transaction 노드 간의 next_to 엣지를 설계했다. next_to 엣지는 동일 고객과 동일 카드에서 발생한 거래를 window_end_date 기준으로 정렬한 뒤, 각 거래가 최대 k건의 직전 거래와 연결되도록 했다. 이 방식은 단순히 거래들을 연결하는 것을 넘어 동일한 고객과 카드로 발생한 연속된 거래의 시간적 흐름을 그래프 구조에 직접적으로 반영하여 모델이 최근 거래 패턴의 맥락을 효과적으로 파악하고 이상 탐지 성능을 높이는 데 핵심적인 역할을 한다.

실험적으로 k 값을 달리하여 성능 변화를 비교한 결과 next_to 엣지를 두지 않은 경우보다 k = 1, 3, 5, 7로 설정했을 때 모델의 성능이 개선되었으며, k = 5에서 PR-AUC가 0.9896, Fraud F1-score가 0.9722로 사기 거래 탐지 성능이 가장 우수하여 next_to 엣지 개수 제한은 최대 5개로 설정하였다.



[그림 3-21. HTGN 모델 next_to 엣지 연결 k건 별 PR 커브 비교]



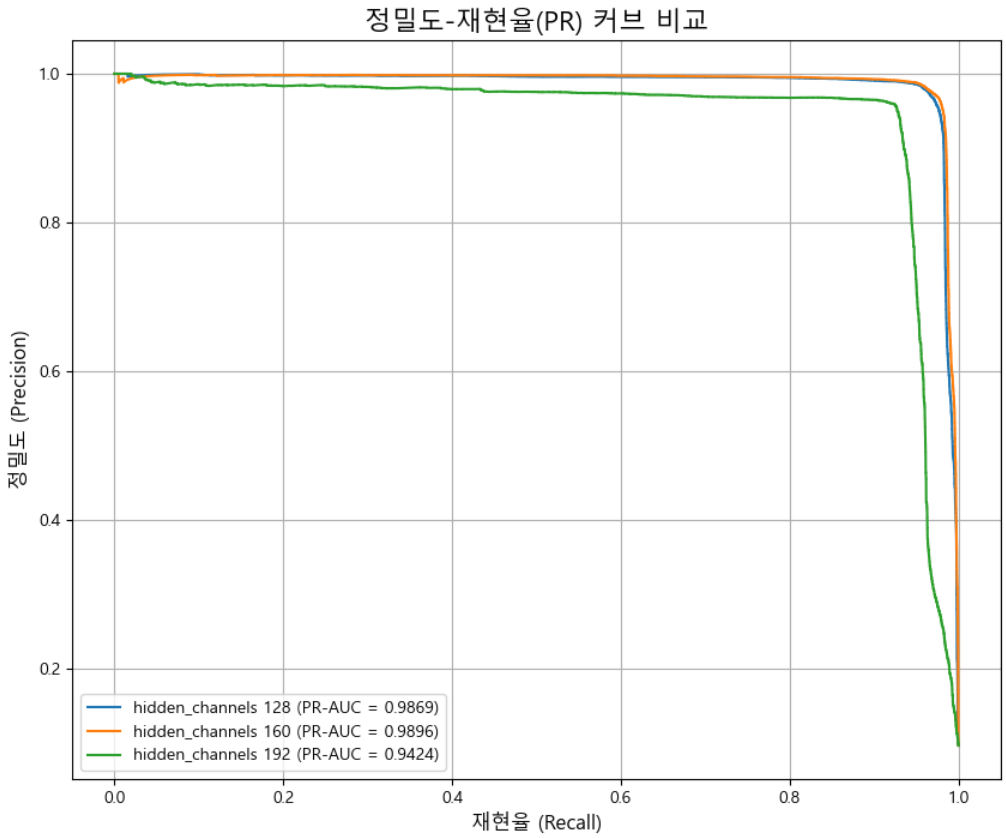
[그림 3-22. HTGN 모델 next_to 엣지 연결 k건 별 (Fraud & Macro) F1-score 비교]

3.4.3.3. HTGN 모델 하이퍼파라미터 최적화

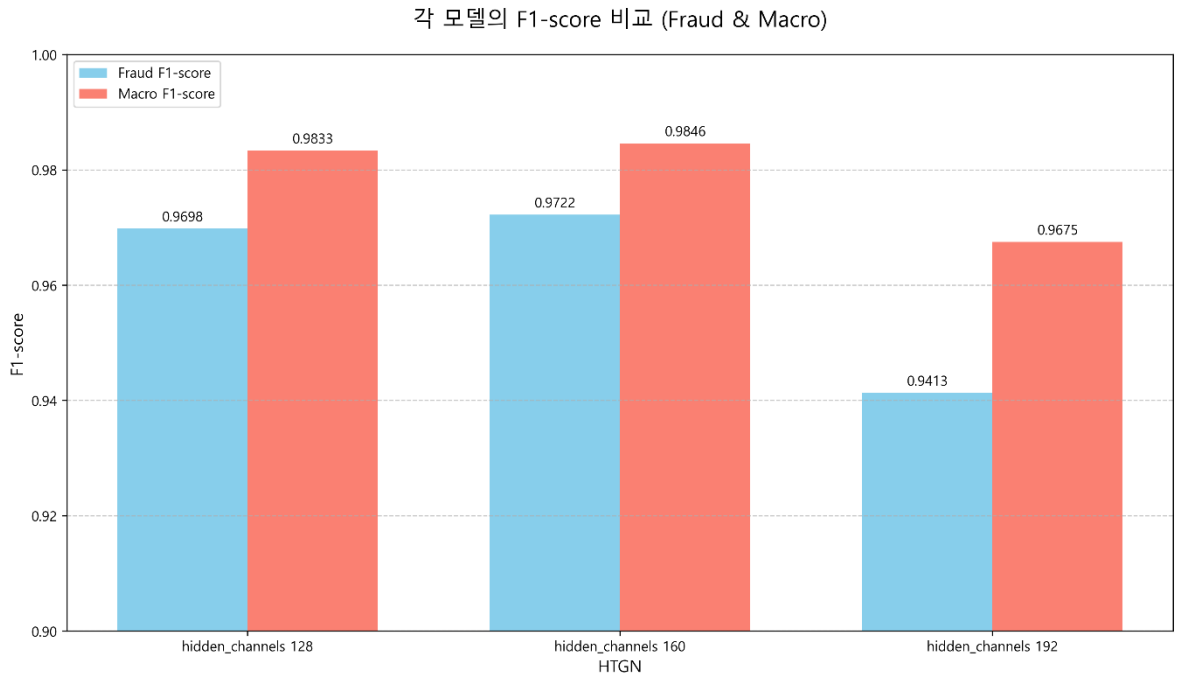
HGT(Heterogeneous Graph Transformer) 모델은 PyG의 HGTConv를 기반으로 설계하였다. 모델의 주요 하이퍼파라미터는 다음과 같이 설정하였다.

- hidden_channels = 160: 노드 잠재 임베딩 차원
- num_layers = 4: 정보 전달 범위를 4-hop까지 확장
- out_channels = 2: 이진 분류 (정상/사기)
- dropout = 0.3: 과적합 방지

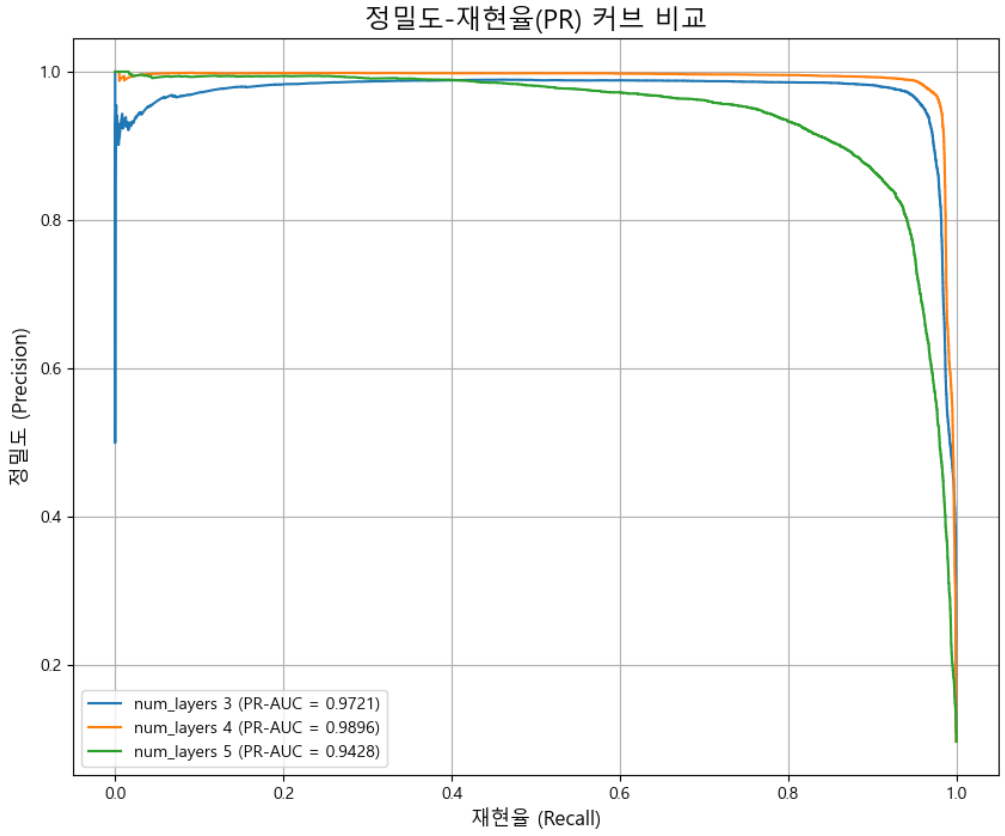
모든 노드 피처는 Linear 레이어를 통해 160차원의 hidden 공간으로 투영되며 상점 노드는 nn.Embedding으로 임베딩을 생성하였다. 이후 4개의 HGTConv 레이어로 메시지 전달을 수행하고 transaction 노드의 최종 임베딩은 분류 레이어(lin_out)를 거쳐 이상 거래 여부를 예측한다.
hidden_channels와 num_layers의 파라미터에 대해 개별적인 변화 실험을 수행한 결과, hidden_channels=160과 num_layers=4에서 PR-AUC 0.9896, 사기 거래에 대한 F1-score 0.9722로 최고 성능이 관찰되었다. 두 파라미터 값을 증가 또는 감소시킬 경우 성능 저하가 확인되어, 본 연구에서는 해당 값을 HTGN 모델의 최적 파라미터로 채택하였다.



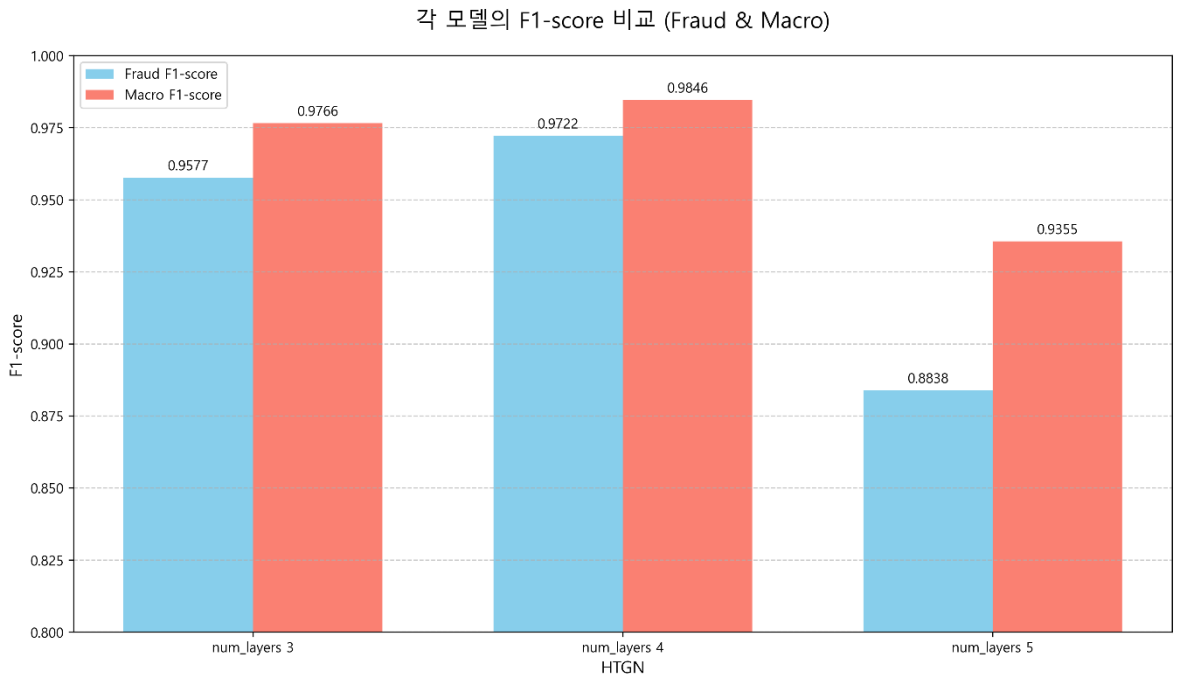
[그림 3-23. HTGN 모델 hidden_channels값 별 PR 커브 비교]



[그림 3-24. HTGN 모델 hidden_channels값 별 (Fraud & Macro) F1-score 비교]



[그림 3-25. HTGN 모델 num_layers값 별 PR 커브 비교]



[그림 3-26. HTGN 모델 num_layers값 별 (Fraud & Macro) F1-score 비교]

3.4.3.4. HTGN 모델 학습

모델은 AdamW 옵티마이저와 클래스 불균형을 고려한 가중치 CrossEntropyLoss로 학습하였다. 사기 거래(1)의 수가 정상 거래(0)에 비해 현저히 적으므로, 각 클래스 개수의 역수를 가중치로 설정하여 모델이 소수 클래스(사기)에 더 집중하도록 유도했다.

학습 과정에서는 최적점 탐색을 돕기 위해 ReduceLROnPlateau 스케줄러를 적용하여 검증 손실이 정체될 경우 학습률을 동적으로 감소시켰다. 초기 학습률은 0.001로 설정했으며, 10번의 에포크 동안 검증 손실이 개선되지 않으면 학습률을 0.5배로 줄였다.

또한, 과적합을 방지하고 최적의 모델을 찾기 위해 EarlyStopping 클래스를 구현했다. 이 클래스는 학습 중 검증 손실이 15번의 에포크 동안 감소하지 않으면 학습을 조기에 종료하고, 손실이 가장 낮았던 시점의 모델 가중치를 자동으로 저장한다.

모델 훈련은 train_one_epoch 메서드가 담당했다. 이 메서드는 각 에포크마다 모델을 훈련 모드로 전환하고, NeighborLoader로부터 미니배치를 불러와 순전파, 손실 계산, 역전파(loss.backward()), 그리고 optimizer.step()을 통한 가중치 업데이트를 반복적으로 수행했다. 손실 계산 시에는 현재 미니배치에서 목표 노드인 transaction 노드에 대한 손실만 정확하게 계산하도록 batch['transaction'].batch_size를 사용했다.

한편, validate_one_epoch 메서드는 각 훈련 에포크가 끝난 후 모델의 성능을 평가하고 과적합 여부를 확인하는 역할을 했다. 이 메서드는 model.eval() 모드와 torch.no_grad() 컨텍스트 내에서 동작하여 가중치 업데이트 없이 검증 데이터셋의 평균 손실을 계산했다. 이 평균 검증 손실은 EarlyStopping과 ReduceLROnPlateau의 동작을 결정하는 핵심 지표로 활용되었다.

최종적으로 학습이 완료된 모델은 분리된 테스트 데이터셋으로 최종 성능을 평가하였다. 이상 거래 탐지 문제의 특성을 고려하여 Fraud F1-Score를 최대화하는 최적의 임계값을 탐색하고 이 임계값을 기준으로 Accuracy, Precision, Recall 등의 지표를 산출했다. 마지막으로, 예측 결과를 혼동 행렬, PR 곡선, ROC 곡선 등으로 시각화하여 모델의 성능을 다각도로 분석했다.

4. 연구 결과 분석 및 평가

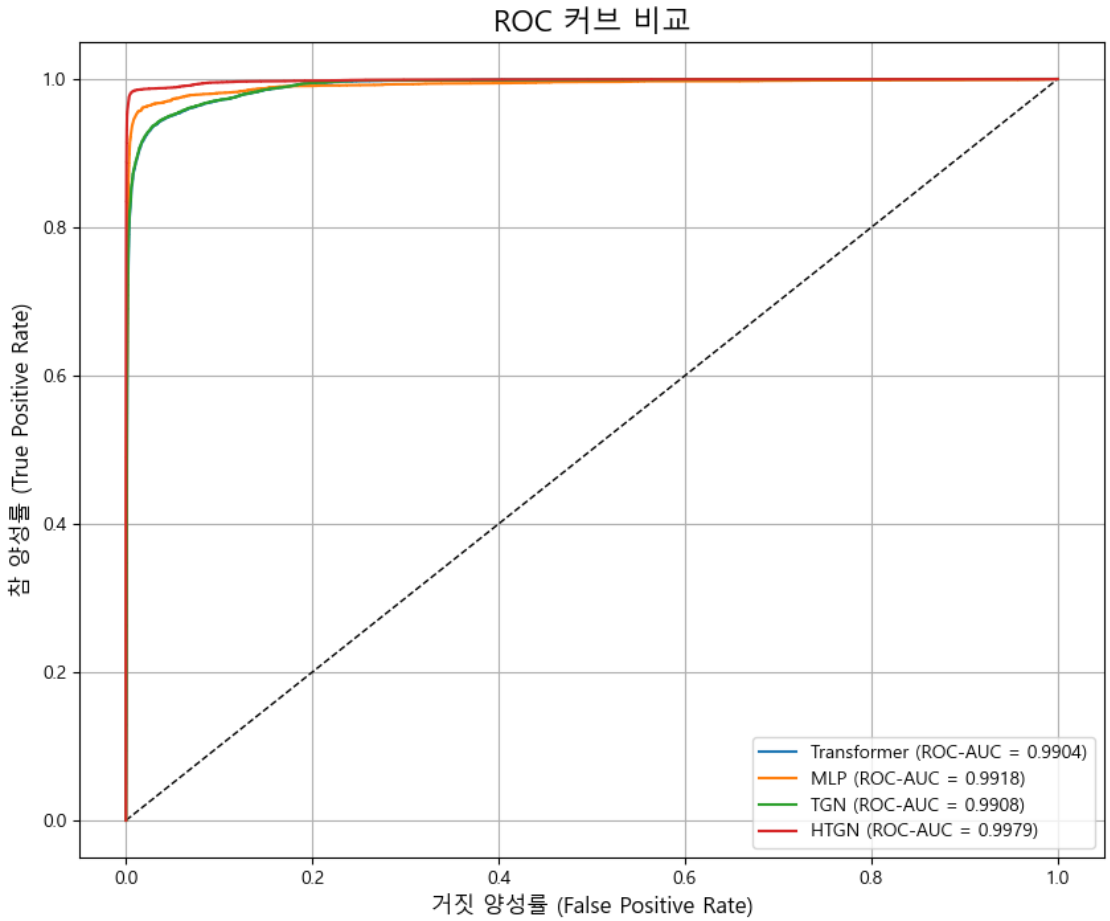
본 연구에서는 트랜스포머(Transformer), MLP, TGN, HTGN 네 가지 딥러닝 모델을 사용하여 이상 거래 탐지 성능을 비교하였다. 각 모델은 데이터 활용 방식에 차이가 있으며, 그에 따른 성능 차이를 분석하는 데 중점을 두었다. 트랜스포머 모델은 거래 데이터의 시계열 속성을 학습하는 데 사용되었으며, MLP, TGN, HTGN 모델은 트랜스포머의 임베딩 결과에 고객 및 카드 데이터를 추가로 활용하여 이상 거래 탐지 성능을 개선하고자 구현되었다. 다음은 각 모델의 정량적 성능 평가 결과이다.

4.1. ROC-AUC 및 PR-AUC를 활용한 모델 성능 평가

모델의 전반적인 분류 성능을 평가하기 위해 ROC-AUC와 PR-AUC 지표를 사용하였다. [표 4-1. 모델별 성능 비교]는 각 모델의 주요 성능 지표를 요약하며, [그림 4-1. ROC 커브 비교]와 [그림 4-2. PR 커브 비교]는 각각의 커브를 시각적으로 보여준다.

모델	Accuracy	ROC-AUC	PR-AUC
Transformer	0.9804	0.9904	0.9379
MLP	0.9882	0.9918	0.9645
TGN	0.9672	0.9908	0.9403
HTGN	0.9946	0.9979	0.9896

[표 4-1. 모델별 성능 비교]

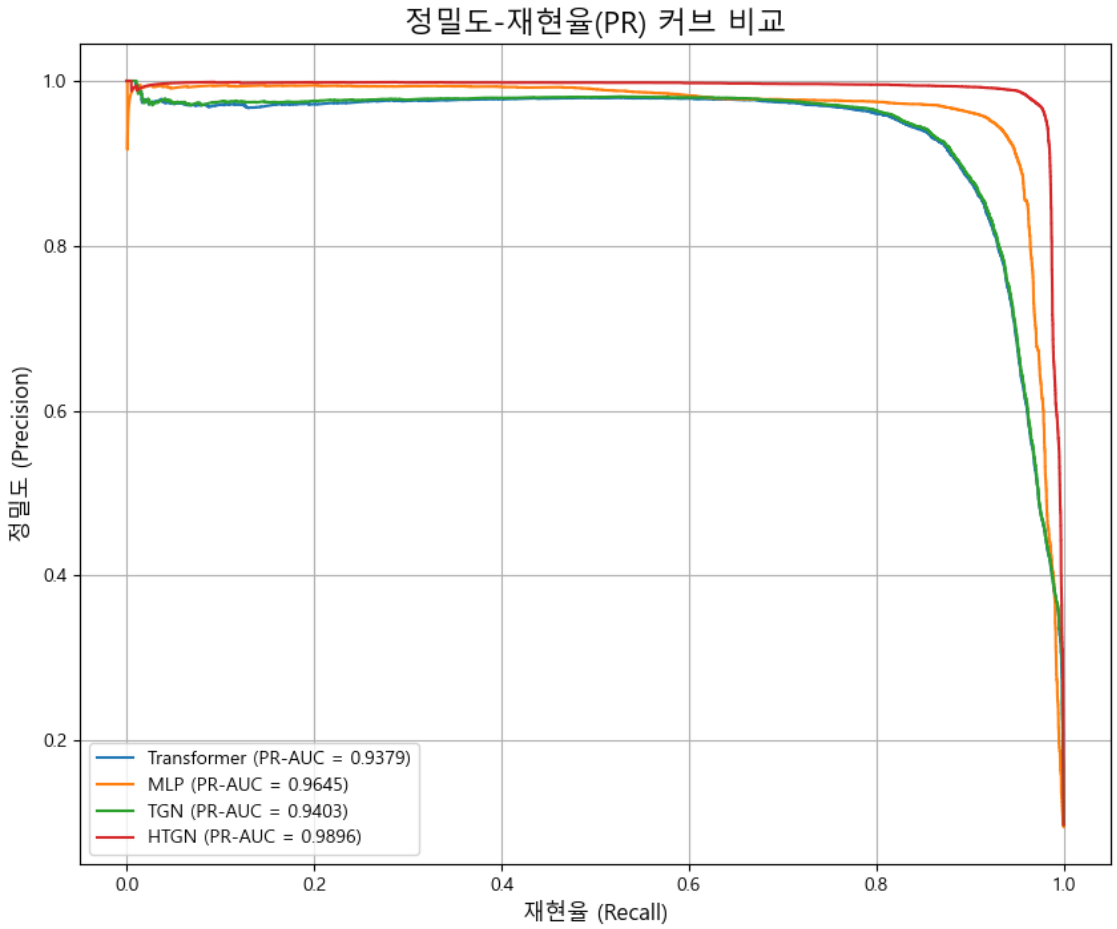


[그림 4-1. ROC 커브 비교]

ROC-AUC 지표는 모델의 전반적인 분류 능력을 평가하는 데 유용하다. 하지만 이상 거래 탐지와 같이 데이터 불균형이 심한 문제에서는 ROC-AUC가 실제 성능을 과대평가할 수 있는 한계가 있다. [그림 4-1. ROC 커브 비교]에서 보듯이 모든 모델이 0.99에 가까운 높은 ROC-AUC 값을 기록하며 뛰어난 성능을 보였다. 이는 정상 거래를 잘 분류하는 모델의 능력에 크게 영향을 받은 결과이다.

이러한 한계를 보완하기 위해 Precision과 Recall의 상충 관계를 표현하며 모델이 Positive 클래스(사기 거래: 1)를 얼마나 잘 예측하는지 평가하는데 초점을 맞춰 데이터 불균형에서 유용한 PR-AUC 지표를 함께 사용하였다. PR-AUC는 모델이 사기 거래를 얼마나 정확하게 예측하는지를 보다 현실적으로 보여준다. [그림 4-2. PR 커브 비교]와 [표 4-1. 모델별 성능 비교]를 보면, HTGN 모델은 PR-AUC 0.9896으로 다른 모델들(Transformer 0.9379, MLP 0.9645, TGN 0.9403)에 비해 우수한 성능을 기록하였다. 이는 HTGN 모델이 단순히 높은 정확도를 넘어 사기 거래를 효과적으로 탐지하는 뛰어난 능

력을 가졌음을 의미한다.



[그림 4-2. PR 커브 비교]

4.2. 클래스별 모델 상세 성능 분석 및 혼동 행렬 평가

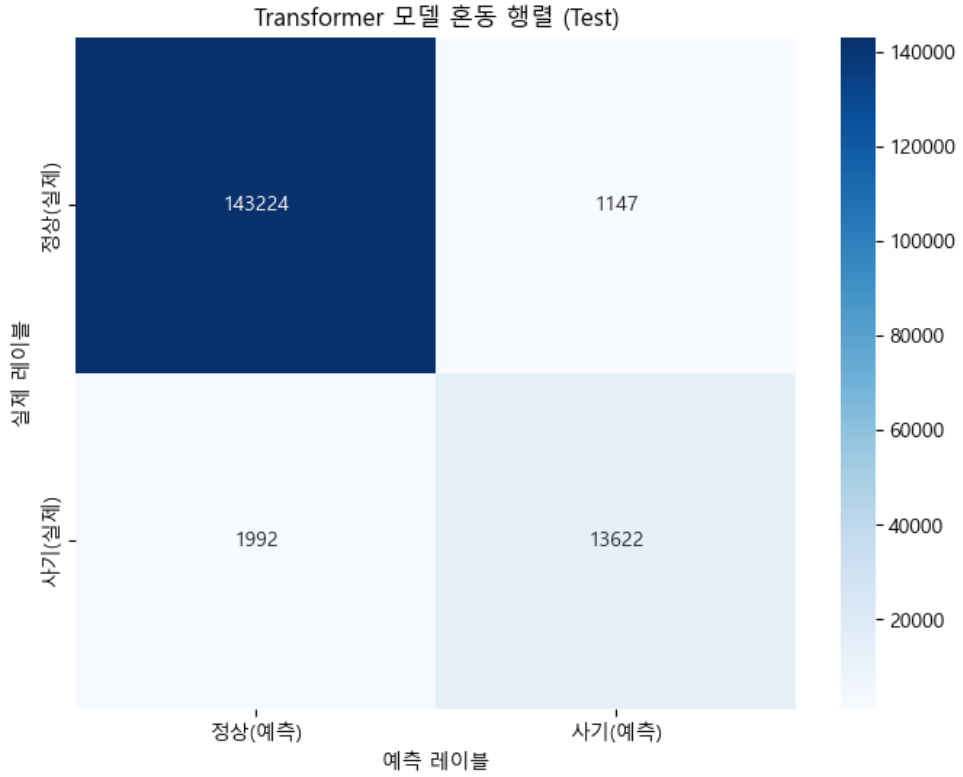
각 모델의 클래스별(정상/사기) 성능을 정밀도, 재현율, F1-score를 중심으로 분석하였다. 각 모델의 혼동 행렬을 첨부하여 지표들이 어떻게 도출되었는지 구체적인 근거를 제시한다.

4.2.1. 트랜스포머 모델 분석

트랜스포머 모델은 거래 데이터의 시계열 속성만을 활용한 모델이다. 정상 거래에 대한 F1-score는 0.9892로 매우 높았지만, 사기 거래에 대한 F1-score는 0.8967로 상대적으로 낮았다. 이는 거래 데이터만으로 사기 거래의 복잡한 패턴을 포착하는 데 한계가 있음을 보여준다. [그림 4-3. 트랜스포머 모델 혼동 행렬]을 보면 15,614개의 사기 거래 중 13,622개를 정확하게 예측하고 1,992개를 미탐지하였다.

Class	Precision	Recall	F1-Score	Support
정상 거래 (0)	0.9863	0.9921	0.9892	144,371
사기 거래 (1)	0.9223	0.8724	0.8967	15,614

[표 4-2. 트랜스포머 모델 클래스별 성능]



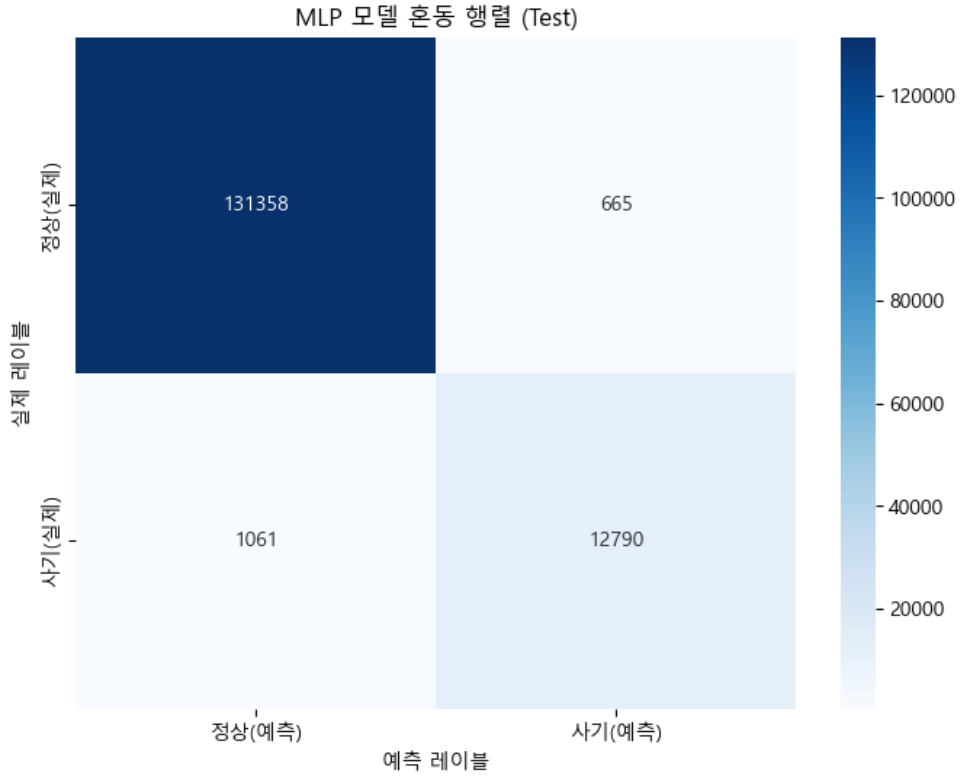
[그림 4-3. 트랜스포머 모델 혼동 행렬]

4.2.2. MLP 모델 분석

MLP 모델은 트랜스포머의 임베딩 결과에 고객 및 카드 데이터를 추가로 활용하였다. MLP 모델은 사기 거래에 대한 F1-score가 0.9368로 트랜스포머 모델보다 향상되었다. 이는 고객 및 카드 데이터를 추가로 활용하여 성능이 개선된 결과로 해석할 수 있다. [그림 4-4. MLP 모델 혼동 행렬]을 보면 정상 거래 오탐지 및 사기 거래 미탐지 수 모두 트랜스포머 모델보다 줄여 성능이 전반적으로 개선되었음을 확인할 수 있다.

Class	Precision	Recall	F1-Score	Support
정상 거래 (0)	0.9920	0.9950	0.9935	132,023
사기 거래 (1)	0.9506	0.9234	0.9368	13,851

[표 4-3. MLP 모델 클래스별 성능]



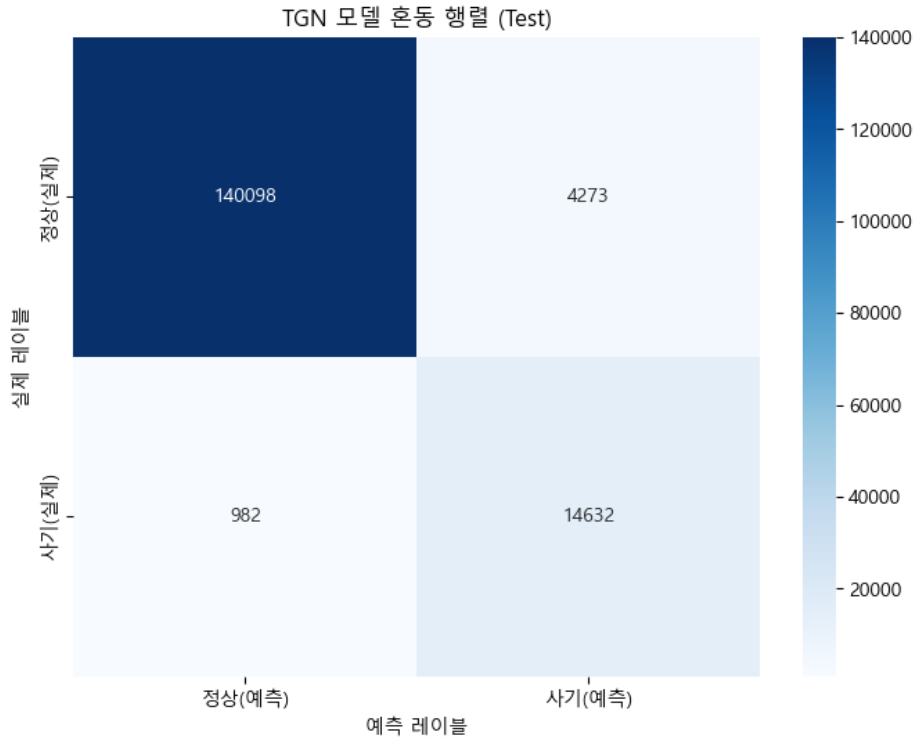
[그림 4-4. MLP 모델 혼동 행렬]

4.2.3. TGN 모델 분석

TGN 모델은 트랜스포머 임베딩에 고객 및 카드 데이터를 통합하여 시계열 정보와 그래프 구조를 함께 학습한 모델이다. TGN 모델은 사기 거래에 대한 재현율(Recall)이 0.9371로 트랜스포머와 MLP보다 우수한 성능을 보였지만, 정밀도(Precision)는 0.7740으로 상대적으로 낮아 사기 거래에 대한 F1-score가 0.8477에 그쳤다. [그림 4-5. TGN 모델 혼동 행렬]을 보면 사기 거래를 비교적 잘 탐지하였지만(14,632개), 정상 거래를 사기 거래로 오탐지한 수가 4,273개로 다른 모델에 비해 상당히 높아 사기 거래 예측 정확도가 떨어짐을 확인할 수 있었다.

Class	Precision	Recall	F1-Score	Support
정상 거래 (0)	0.9930	0.9704	0.9816	144,371
사기 거래 (1)	0.7740	0.9371	0.8477	15,614

[표 4-4. TGN 모델 클래스별 성능]



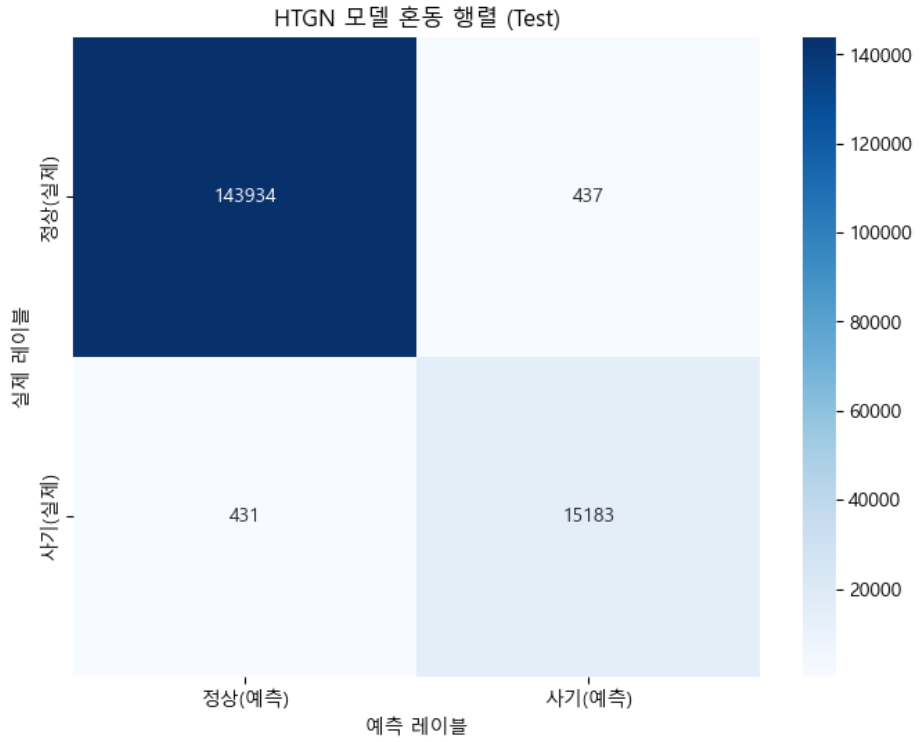
[그림 4-5. TGN 모델 혼동 행렬]

4.2.4. HTGN 모델 분석

HTGN 모델은 트랜스포머 임베딩과 고객 및 카드 데이터를 통합하여, 이종 그래프 상의 고객, 카드, 거래, 상점 간의 관계를 학습한 모델이다. HTGN은 사기 거래에 대한 정밀도 0.9720, 재현율 0.9724를 기록하며 가장 높은 사기 거래에 대한 F1-score인 0.9722를 달성하였다. 이는 HTGN 모델이 사기 거래를 높은 정확도와 재현율로 탐지했음을 보여준다. [그림 4-6. HTGN 모델 혼동 행렬]을 보면 다른 모델들에 비해 오탐지 및 미탐지 수가 가장 크게 줄어 144,371개의 정상 거래 중 437개, 15,614개의 사기 거래 중 431개만을 잘못 분류하였다.

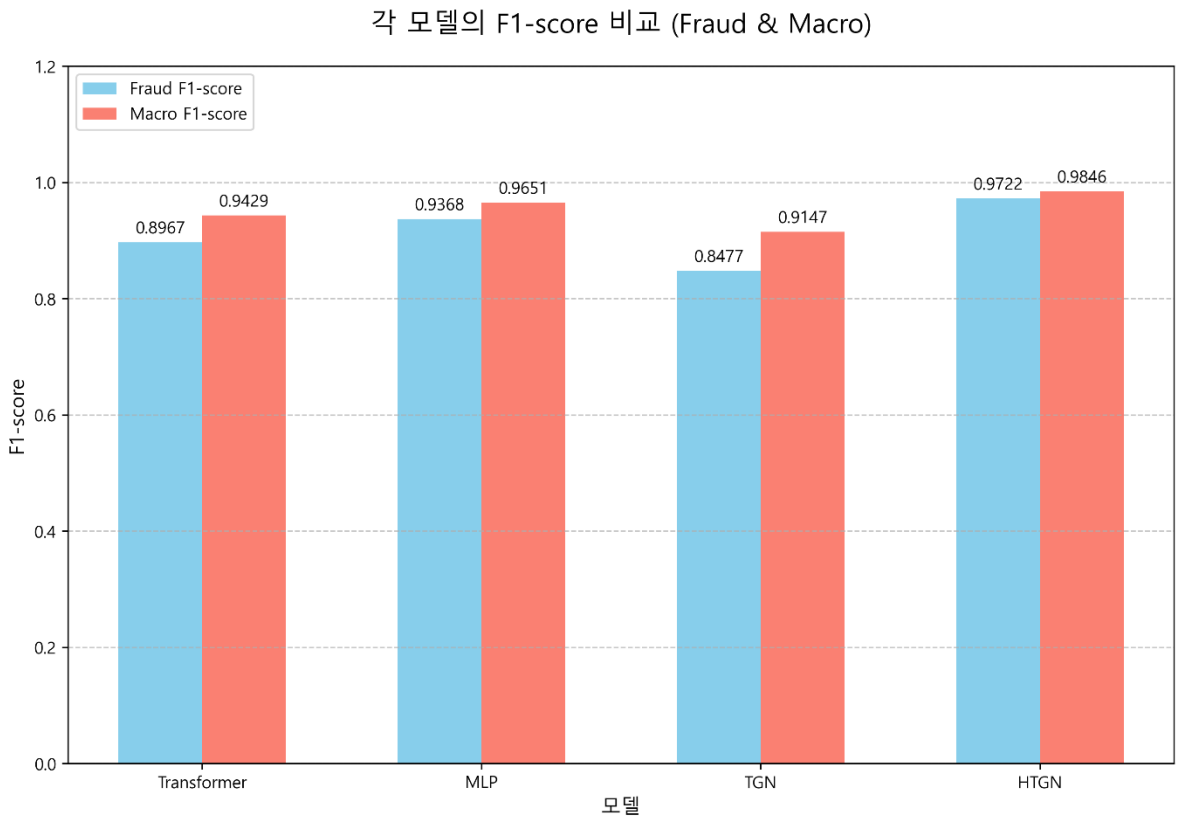
Class	Precision	Recall	F1-Score	Support
정상 거래 (0)	0.9970	0.9970	0.9970	144,371
사기 거래 (1)	0.9720	0.9724	0.9722	15,614

[표 4-5. HTGN 모델 클래스별 성능]



[그림 4-6. HTGN 모델 혼동 행렬]

4.2.5. (Fraud & Macro) F1-score 종합 비교



[그림 4-7. (Fraud & Macro) F1-score 비교]

결론적으로, HTGN 모델은 이종 그래프 데이터의 복잡한 관계를 학습함으로써 이상 거래를 탐지하는 데 가장 뛰어난 성능을 보였다. 특히, 오탐지 및 미탐지율이 가장 낮아 실용적인 측면에서도 가장 신뢰할 수 있는 모델로 평가된다.

5. 결론 및 향후 연구 방향

본 연구에서는 시계열 기반 트랜스포머 모델을 활용하여 거래 데이터의 시계열적 특성을 반영한 임베딩을 생성하고 여기에 고객 및 카드 데이터를 통합하여 다양한 딥러닝 모델(MLP, TGN, HTGN)을 설계 및 비교하였다. 성능 분석 결과, 다음과 같은 결론을 도출하였다.

첫째, 트랜스포머 모델은 시계열 패턴을 학습하여 정상 거래 분류에서는 높은 성능을 보였으나 사기 거래 탐지에서는 한계가 존재하였다.

둘째, 고객 및 카드 데이터를 추가로 반영한 MLP 모델은 트랜스포머 모델 대비 사기 거래 탐지 성능이 향상되어 단순한 시계열 정보만으로는 포착되지 않는 패턴을 보완할 수 있음을 확인하였다.
셋째, TGN 모델은 시계열 정보와 그래프 구조를 함께 학습함으로써 사기 거래를 적극적으로 탐지하였으나 정상 거래에 대한 오탐지가 크게 증가하는 문제를 보였다.
넷째, HTGN 모델은 이종 그래프 기반 학습을 통해 고객, 카드, 거래, 상점 간의 복잡한 관계를 효과적으로 반영하였으며 사기 거래 탐지 성능에서 가장 높은 정밀도와 재현율을 동시에 달성하였다. 이는 실제 금융 환경에서의 실용적 활용 가능성을 가장 높게 평가할 수 있는 결과이다.

종합하면, HTGN 모델이 다양한 이종 데이터 소스를 결합하여 이상 거래 탐지 성능을 극대화할 수 있음을 실증적으로 확인하였다.

향후 연구에서는 다음과 같은 보완이 필요하다.

1. 데이터 불균형 문제 정교화: 사기 거래의 희소성으로 인한 데이터 불균형 문제에 대해 단순한 증강이나 언더샘플링을 넘어 보다 정교한 데이터 증강 기법(TimeGAN 개선, contrastive learning 등)의 적용이 향후 성능 향상에 기여할 수 있다.
2. 실시간 탐지 환경 적용: 연구에서는 오프라인 데이터셋 기반 성능 평가에 초점을 두었으나, 실시간 거래 환경에서는 지연 시간(latency) 최소화가 중요하다. 따라서, HTGN 모델의 추론 속도를 개선하고 온라인 학습 구조로 확장하는 연구가 필요하다.
3. 설명 가능성 강화: 금융 분야에서는 탐지 결과에 대한 설명 가능성(Explainability)이 요구된다. SHAP, GNNExplainer 등의 기법을 활용하여 모델이 특정 거래를 사기 거래로 판단한 근거를 제시하는 방안이 필요하다.

6. 구성원별 역할 및 개발 일정

6.1. 구성원별 역할

조원	역할
배근호	데이터 전처리 - 거래 데이터 증강 및 언더샘플링 - 파생 속성 생성 및 중요 속성 분석 거래 데이터 시계열 정보 처리 트랜스포터 모델 개발 이상 거래 탐지 HTGN 모델 개발 및 평가 보고서 작성
추민	데이터 전처리 - 파생 속성 생성 및 중요 속성 분석 이상 거래 탐지 MLP 모델 개발 및 평가 보고서 작성
윤소현	데이터 전처리 - 거래 발생 위치를 위도/경도 좌표로 변환 - 파생 속성 생성 및 중요 속성 분석 이상 거래 탐지 TGN 모델 개발 및 평가 보고서 작성

6.2. 개발 일정

업무	6				7				8				9			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
데이터 수집 및 피처 분석																
데이터 불균형 해소 방안 적용 및 전처리																
트랜스포머 시계열 데이터 처리																
중간보고서 작성																
이상 거래 탐지 딥러닝 모델 개발																
모델 성능 평가 및 개선																
최종보고서 작성																

7. 참고 문헌

[1] Al-dahasi, Ezaz Mohammed, et al. "Optimizing fraud detection in financial transactions with machine learning and imbalance mitigation." *Expert Systems* 42.2 (2025): e13682.

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[3] E. Rossi,B. Chamberlain, F. Frasca, D. Eynard, F. Monti, M. Bronstein, "Temporal Graph Networks for Deep Learning on Dynamic Graphs", *ICML 2020 Workshop on Graph Representation Learning* , pp. 1, 2020.

[4] E. Rossi. (2020). TGN: Temporal Graph Networks [Online]. Available: https://github.com/twitter-research/tgn (Accessed: Sep. 1, 2025)