

# MoLE-Flow: Mixture of LoRA Experts with Normalizing Flow for Continual Anomaly Detection

## 지속적 이상 탐지를 위한 LoRA 전문가 혼합 정규화 흐름

Hun Im, Pilsung Kang\*

Seoul National University

January 8, 2026

# Contents

- 1 Introduction
- 2 Previous Method
- 3 Overall Architecture
- 4 Feature Extraction
- 5 Task Adapters
- 6 MoLE-Flow
- 7 Training & Inference
- 8 Experiments Part 1: Hyperparameter Optimization
- 9 Experiments Part 2: Paper Experiment Design
- 10 Conclusion & Future Work

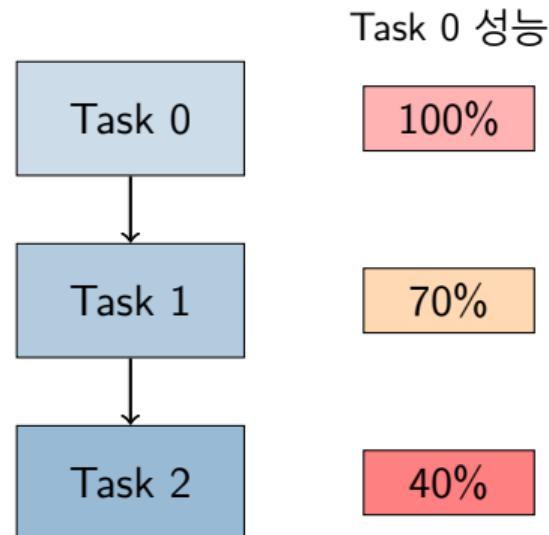
# Problem: Continual Anomaly Detection

## 산업 환경의 현실

- 새로운 제품이 지속적으로 추가됨
- 모든 데이터를 저장/재학습하기 어려움
- 이전 제품에 대한 성능 유지 필요

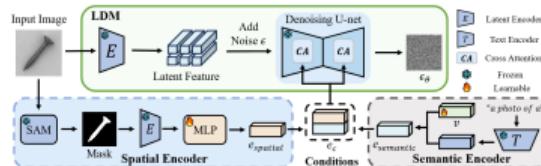
## 핵심 문제: Catastrophic Forgetting

- 새 작업 학습 시 이전 작업 성능 급감
- 기존 방법: 전체 모델 복제 → 메모리 비효율



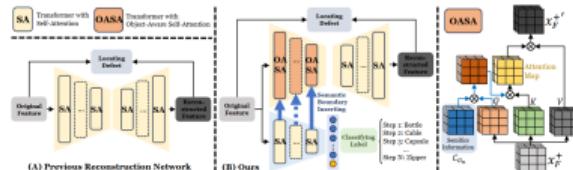
# 기존 연구 방법

## Replay 방식 (ReplayCAD)



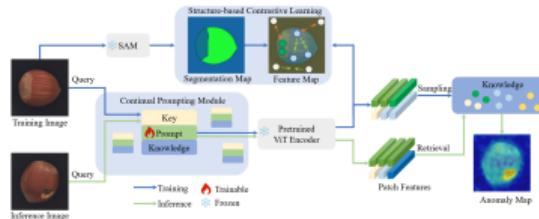
이전 작업 데이터를 재사용하여 학습

## Prompt 방식 (IUF)



Class-specific 정보를 학습하기 위한 별도의 Discriminator를 사용

## Adapter 방식 (UCAD)



Class-specific 정보를 학습하기 위한 별도의 Prompt를 학습

## 기존 연구의 한계점

- **Replay 방식:** 오랜 학습 시간이 소요되며, replay로 인한 memory cost 존재
- **Adapter 방식:** Discriminator 또한 Catastrophic Forgetting 유발 가능성 있으며, 높은 학습 cost 존재
- **Prompt 방식:** 추론 단계에서 Task ID 인식을 위한 별도의 프로세스 필요

# Our Solution: MoLE-Flow

핵심 아이디어: Parameter Isolation

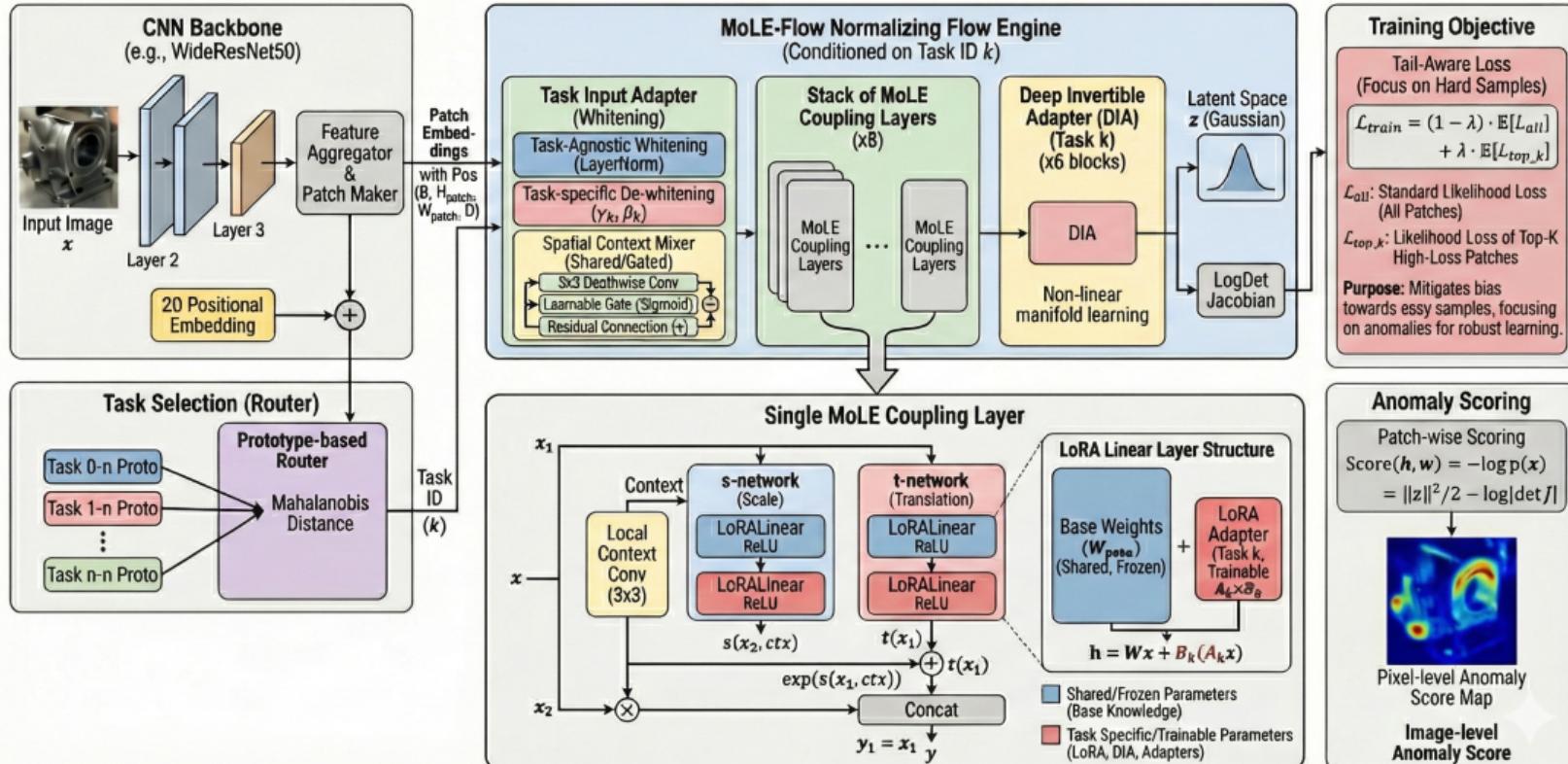
Base 파라미터는 공유/고정하고, 경량 어댑터를 통해 만 task-specific 파라미터를 학습

5가지 핵심 기여:

- ① **MoLE (Mixture of LoRA Experts)**: NF coupling 블록에 LoRA 적용
- ② **Whitening Adapter**: 작업 간 분포 차이 정렬
- ③ **Deep Invertible Adapter (DIA)**: 비선형 매니폴드 적용
- ④ **Prototype-based Router**: Task ID 없이 자동 라우팅
- ⑤ **Tail-Aware Loss**: 분포 경계 학습 강화

# Overall Architecture

## MoLE-Flow: Continual Learning via Mixture of LoRA Experts & Deep Invertible Adapters for Anomaly Detection



## Parameter Isolation 전략:

- 회색 (Shared/Frozen): Backbone, Base NF weights - 모든 작업이 공유
- 빨간색 (Task-specific): LoRA, DIA, Whitening Adapter - 작업별 독립

## 학습 전략:

- Task 0: Base NF + 모든 어댑터 학습 → Base 동결
- Task  $t > 0$ : 새로운 어댑터(LoRA, Whitening, DIA)만 학습

### 핵심 장점

작업당 **약 10.8%**의 추가 파라미터만으로 완전한 작업 분리 달성

# Pipeline 수식

$$x \xrightarrow{\text{Backbone}} \mathbf{F} \xrightarrow{\text{PE}} \mathbf{F}' \xrightarrow{\text{Whitening}} \hat{\mathbf{F}} \xrightarrow{\text{SCM}} \tilde{\mathbf{F}} \xrightarrow{\text{NF+LoRA}} \mathbf{z}' \xrightarrow{\text{DIA}} (\mathbf{z}, \log |\det \mathbf{J}|)$$

## 공유/고정 파라미터

- Backbone (ViT-Base)
- Base NF weights ( $\mathbf{W}_{\text{base}}$ )
- Spatial Context Mixer

## Task-specific 파라미터

- LoRA:  $\mathbf{A}_t, \mathbf{B}_t$
- Whitening:  $\gamma_t, \beta_t$
- DIA: 별도 flow 블록

## 핵심 장점

작업당 **약 10.8%**의 추가 파라미터만으로 완전한 작업 분리 달성

# Notation: 주요 표기법

## 입력 및 특징

- $\mathbf{X} \in \mathbb{R}^{B \times H \times W \times D}$ : 패치 임베딩 텐서 (배치  $B$ , 공간 해상도  $H \times W$ , 특징 차원  $D$ )
- $\mathbf{x}_{u,v} \in \mathbb{R}^D$ : 위치  $(u, v)$ 의 개별 패치 벡터
- $\mathbf{z}$ : 잠재 공간 변수 (Normalizing Flow 출력)

## 작업 및 변환

- $t \in \{0, 1, \dots, T - 1\}$ : 작업 인덱스
- $\mathbf{J}_f$ : 함수  $f$ 의 Jacobian 행렬
- $\log |\det \mathbf{J}|$ : Log-determinant (변환의 부피 변화율)

## 손실 함수 가중치

- $\lambda_{\text{tail}}$ : Tail-Aware Loss 가중치 (default: 0.7)
- $\lambda_{\logdet}$ : Log-determinant 정규화 계수 (default:  $10^{-4}$ )
- $\lambda_{\text{reg}}$ : 공분산 정규화 항 (default:  $10^{-5}$ )

# Parameter Analysis (Backbone 제외)

**Default Configuration:** embed\_dim=512, coupling\_layers=8, lora\_rank=32, dia\_n\_blocks=4

## Shared Parameters (Task 0 후 Freeze)

Module	Params
Base NF (8 subnets)	5,255,168
s_layer1 (512→512)	2,101,248
s_layer2 (512→256)	1,050,624
t_layer1 (256→512)	1,052,672
t_layer2 (512→256)	1,050,624
Context Conv (x8)	40,968
SpatialContextMixer	5,121
<b>Total Shared</b>	<b>5,301,257</b>

## Per-Task Parameters (매 Task 생성)

Module	Params
LoRA Adapters (A+B)	851,968
s_layer1 LoRA	262,144
s_layer2 LoRA	196,608
t_layer1 LoRA	196,608
t_layer2 LoRA	196,608
Task Bias	12,288
WhiteningAdapter	1,024
DIA (4 blocks)	1,579,008
<b>Total Per-Task</b>	<b>2,444,288</b>

# Parameter Breakdown Visualization

## 모듈별 파라미터 비중

■ Shared (Frozen) ■ Per-Task

|| Context/Bias <1%

■ LoRA 11.0%

■ DIA 20.4%

■ Base NF 67.8%

## Task 수에 따른 누적 파라미터

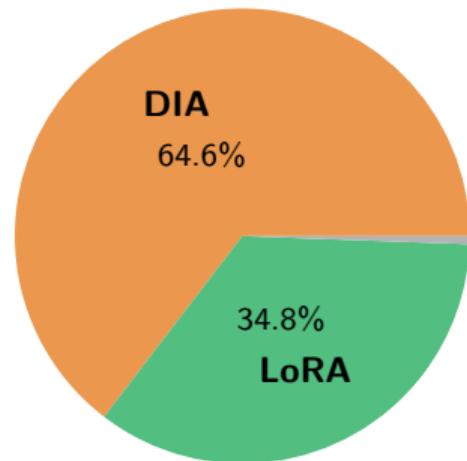
Tasks	Total Params	증가량
Task 0	7.75M	-
Task 1	10.19M	+2.44M
Task 2	12.63M	+2.44M
Task 3	15.08M	+2.44M
<b>15 Tasks</b>	<b>41.97M</b>	-

### Per-Task 비율

$\frac{2.44M}{7.75M} \approx 31.6\% \text{ (Task 0 대비)}$   
Backbone (86M) 제외 시 매우 효율적

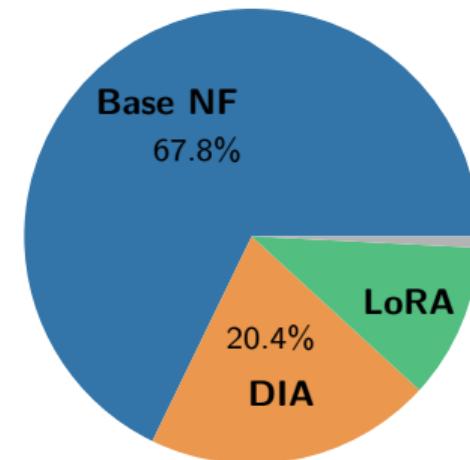
# Parameter Distribution (Per-Task 기준)

Per-Task 파라미터 구성



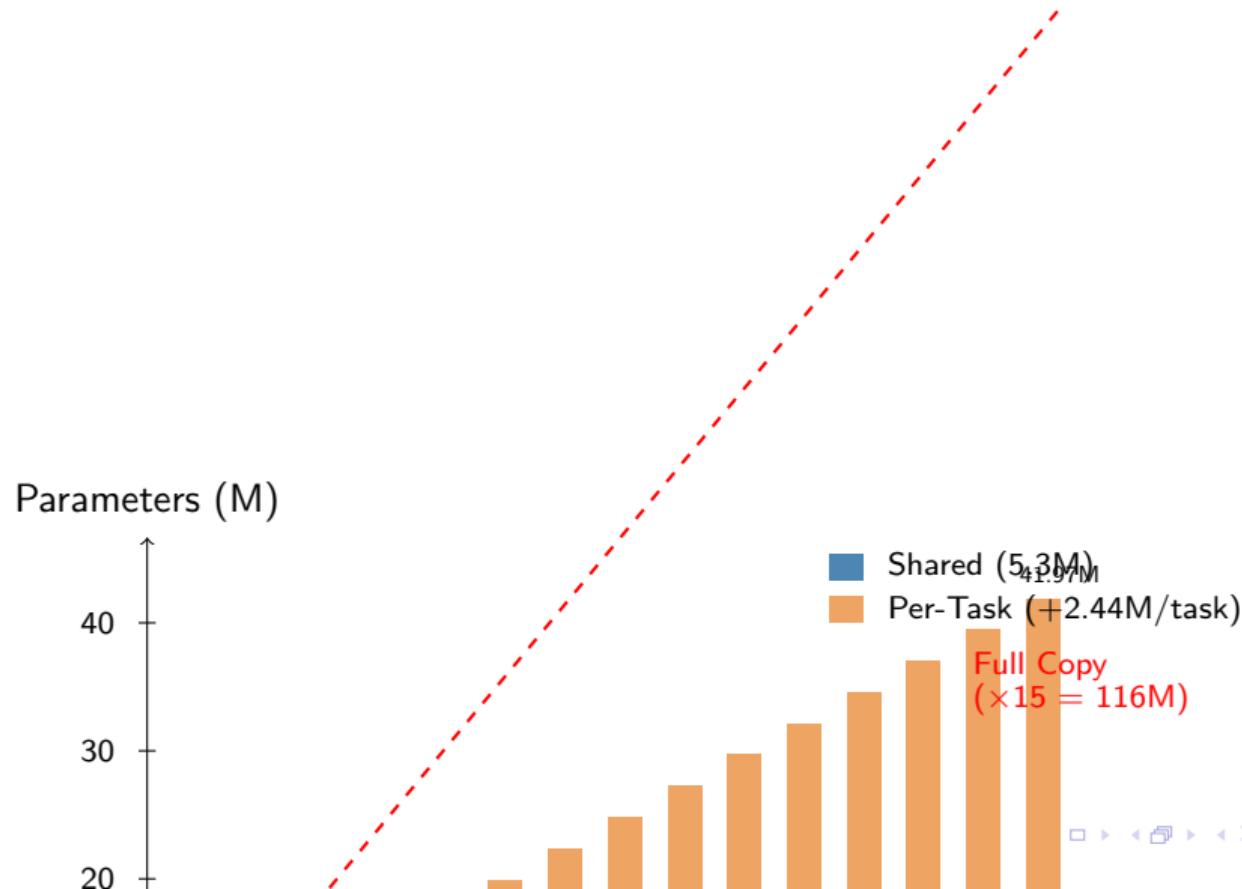
Task Bias + Whitening < 1%

전체 모델 파라미터 구성 (Task 0)



■ Shared (Frozen) ■ Per-Task

# Task 수에 따른 누적 파라미터 증가



# DIA (Deep Invertible Adapter) Parameter Details

## DIA 구조: 4개의 AffineCouplingBlock

### AffineCouplingBlock 구성

- $\text{split\_dim} = 512 / 2 = 256$
- $\text{hidden\_dim} = 512 \times 0.5 = 256$

### SimpleSubnet (s\_net, t\_net)

- Linear( $256 \rightarrow 256$ ): 65,792
- Linear( $256 \rightarrow 256$ ): 65,792
- Linear( $256 \rightarrow 256$ ): 65,792
- **Subnet Total:** 197,376

### 파라미터 계산

Component	Params
s_net (per block)	197,376
t_net (per block)	197,376
<b>Block Total</b>	394,752
<b>4 Blocks Total</b>	<b>1,579,008</b>

### DIA의 역할

비선형 매니폴드 적응으로  
LoRA가 표현 못하는 복잡한 분포 변환 학습

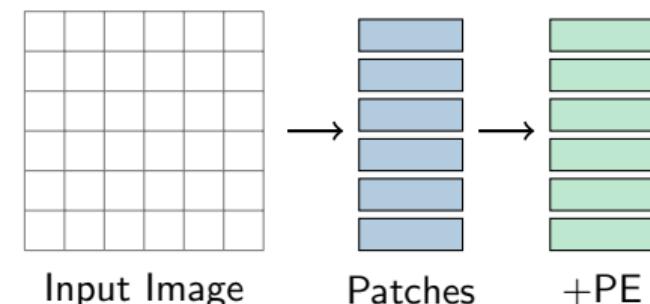
# Feature Extraction & Preprocessing

## 1. Backbone (ViT-Base)

- 사전 학습된 ViT 사용 (Frozen)
- 다중 스케일 특징: 블록 {1, 3, 5, 11}

## 2. Patch Maker

- Feature Map을 패치 단위로 분할
- 출력:  $\mathbf{F} \in \mathbb{R}^{B \times H \times W \times D}$
- PatchCore의 방식과 동일하게 패치 단위로 분할하여 패치 임베딩을 생성



## 3. Positional Encoding

- NF의 순열 불변성 극복
- 2D Sinusoidal PE 추가

$$\mathbf{F}' = \mathbf{F} + \mathbf{P}$$

# Why Positional Encoding?

## Normalizing Flow의 구조적 특성:

- NF는 입력을 1D 벡터로 평탄화하여 처리
- 각 패치 임베딩이 **독립적**으로 처리됨 (Permutation Invariance)
- 패치의 **공간적 위치 정보가 소실**

## Positional Encoding 적용:

$$\mathbf{F}' = \mathbf{F} + \mathbf{P}, \quad \mathbf{P} \in \mathbb{R}^{H \times W \times D}$$

### 2D Sinusoidal PE

- 행과 열에 대해 독립적인 주파수 사용
- 연속적인 위치 관계 표현
- 학습 파라미터 없음

### 효과

- 동일한 특징이라도 위치에 따라 다르게 처리
- 이상치의 “어디”가 중요한지 인식
- 공간적 패턴 학습 가능

### 핵심

NF는 “무엇(What)”만 처리 → PE 추가로 “어디(Where)” 정보 주입 → 위치 기반 이상 탐지

# Task Adapters: WhiteningAdapter

## 문제: 클래스별 이미지 분포 차이

- 클래스별로 다른 이미지 통계량 → NF 학습 불안정
- Task 전환 시 급격한 입력 분포 변화 (Covariate Shift)
- 고정된 Base Flow가 새로운 분포에 최적 반응 불가

## 해결: 2단계 Whitening 전략

### Step 1: Task-Agnostic Whitening

$$\mathbf{x}_{\text{white}} = \frac{\mathbf{F}' - \mathbb{E}[\mathbf{F}']}{\sqrt{\text{Var}[\mathbf{F}']} + \epsilon}$$

### Step 2: Task-Specific De-whitening

$$\hat{\mathbf{F}}_t = \gamma_t \odot \mathbf{x}_{\text{white}} + \beta_t$$

- 모든 Task를 동일한 시작점으로
- LayerNorm 기반 (학습 파라미터 없음)
- 분포 스케일 차이 사전 제거

- $\gamma_t$ : 채널별 중요도 조절
- $\beta_t$ : Base Flow 최적 영역으로 이동
- 범위 제한으로 안정성 확보

핵심 원리: Affine 변환을 사용하는 이유

# WhiteningAdapter: Why Affine Transformation?

De-whitening에서 Affine 변환 ( $y = \gamma \odot x + \beta$ )을 사용하는 이유:

## 1. 통계적 정합성

- 정규분포  $\rightarrow$  정규분포 변환의 자연스러운 방법
- $\mathcal{N}(0, 1) \rightarrow \mathcal{N}(\mu_t, \sigma_t^2)$
- 비선형 변환 시 분포 모양 왜곡 (Bell curve 소실)

## 2. 구조 보존

- 공간 늘림(Scale) + 이동(Shift)만 수행
- 직선  $\rightarrow$  직선, 평행선  $\rightarrow$  평행선 유지
- 데이터 “내용(Content)” 보존

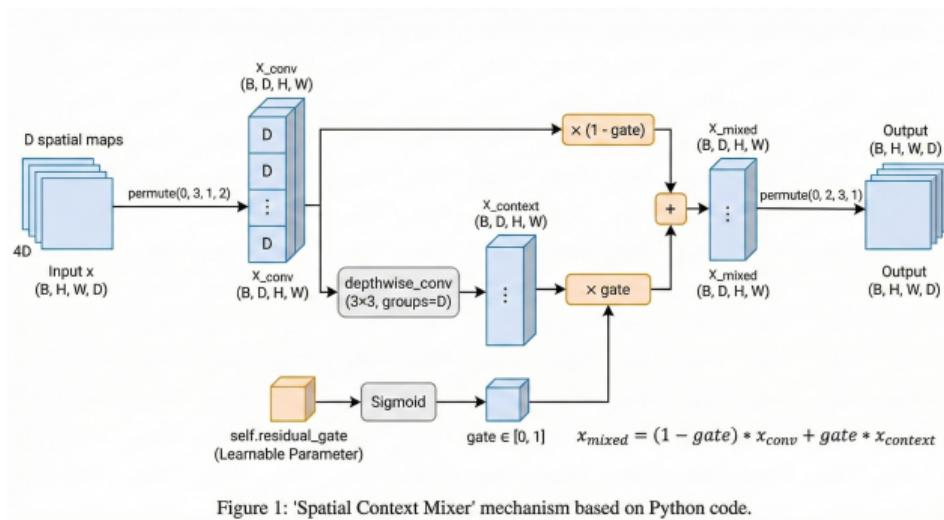
## 3. 역할 분담

- Adapter: 1차/2차 모멘트(평균, 분산) 정렬
- Base Flow + LoRA: 복잡한 비선형 분포 모델링
- BatchNorm [?]의  $\gamma$ ,  $\beta$ 와 동일 원리

## 4. 채널별 중요도 조절

- $\gamma_t \uparrow$ : 해당 특징 증폭
- $\gamma_t \downarrow$ : 해당 특징 억제
- 예: 가죽  $\rightarrow$  질감 채널 증폭, 나사  $\rightarrow$  형태 채널 증폭

# MoLE-Flow: Spatial Context Mixer



**문제:** 패치 독립 처리의 한계

- 기존 NF:  $p(\mathbf{X}) \approx \prod_{u,v} p(\mathbf{x}_{u,v})$
- 주변 맥락 정보 부족
- 결함 경계(Local Contrast) 인식 어려움

**해결:** Gated Depthwise Aggregation

$$\mathbf{x}_{\text{mixed}} = (1 - g) \cdot \mathbf{x} + g \cdot \mathbf{x}_{\text{ctx}}$$

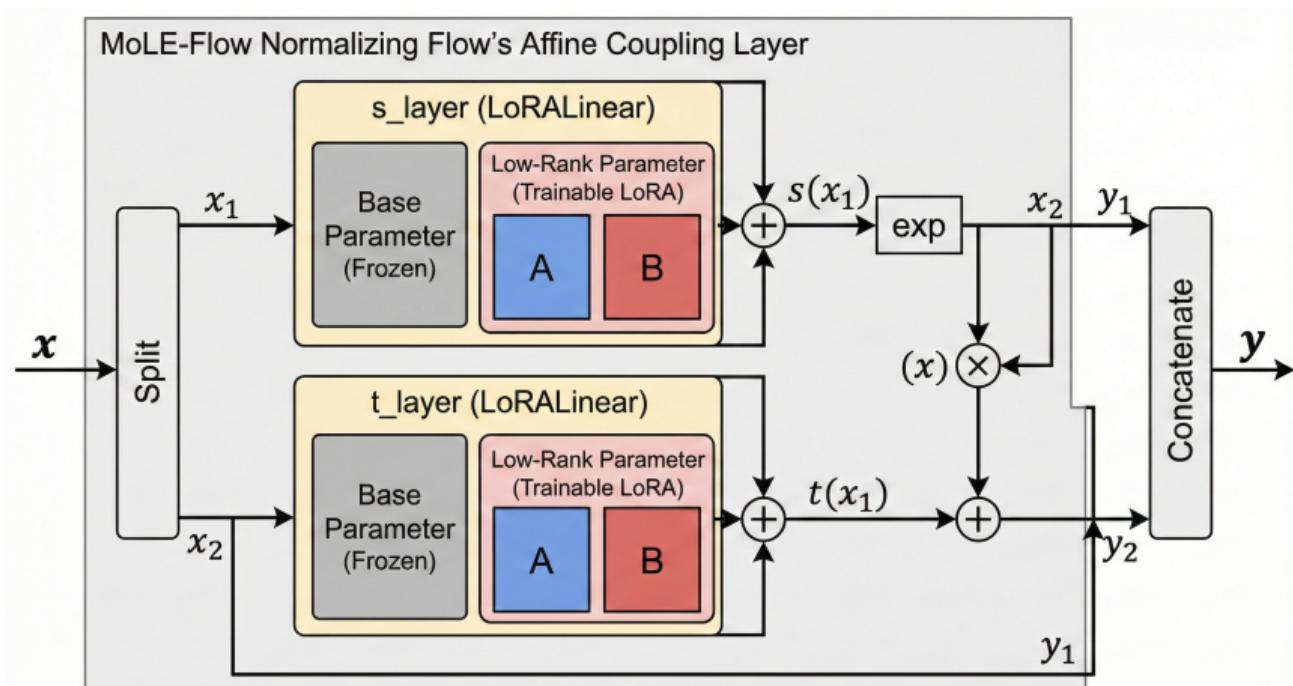
- $\mathbf{x}_{\text{ctx}}$ : DepthwiseConv $K \times K$
- $g = \sigma(\theta)$ : 학습 가능 게이트
- 채널 독립적 공간 집계

**핵심 원리:** Pseudo-Dependency Modeling

# MoLE-Flow: LoRA-based Coupling Layer

**문제:** 전체 모델 Fine-tuning의 한계

- 전체 파라미터 학습 → **Catastrophic Forgetting** 발생
- Task별 별도 모델 저장 → 메모리 비효율



# MoLE-Flow: LoRA-based Coupling Layer (상세)

## Base Linear (Frozen Anchor)

- Task 0에서 학습 후 영구 고정
- 범용적인 특징 변환 역할
- 모든 작업이 공유하는 “공통 지식”

## Why Frozen?

- 새 Task 학습 시 Base 변경 → 이전 Task 성능 저하
- Parameter Isolation의 핵심 원리

## LoRA Linear (Task-Specialist)

- 작업 고유 분포 특성 보정
- 저랭크 행렬:  $\Delta\mathbf{W} = \mathbf{BA}$
- 작업마다 독립적인  $\mathbf{A}_t, \mathbf{B}_t$  생성

## 초기화 전략:

- $\mathbf{A}$ : Xavier uniform (다양성)
- $\mathbf{B}$ : Zero → 학습 시작 시  $\Delta\mathbf{W} = 0$
- Base에서 점진적으로 분기

## 핵심 원리: Parameter Isolation

Base(공유) + LoRA(분리) 구조로 작업 간 간섭을 원천 차단 → Forgetting  $\approx 0$

# MoLE-Flow: Scale Context (s-network only)

**문제:** s-network가 anomaly 판단에 주변 정보 필요

- 이상치의 “크기(scale)”는 주변 패치와의 상대적 비교에 의존
- t-network는 분포 중심 이동만 담당 → context 불필요

**해결:** s-network에만 Context 입력

Scale Network (Context-Aware)

$$\mathbf{s} = f_s([\mathbf{x}; \mathbf{ctx}_{K \times K}])$$

Translation Network (Context-Free)

$$\mathbf{t} = f_t(\mathbf{x})$$

- 원본 +  $K \times K$  문맥 정보 결합
- 이상치 “돌출” 정도 판단
- DepthwiseConv로 효율적 집계

- 원본 특징만 사용
- 분포 중심은 패치 고유 특성
- Context 추가 시 오히려 노이즈

핵심 원리: 역할 분리

Scale  $\mathbf{s}$ : 주변 대비 “얼마나 다른가” (상대적) / Translation  $\mathbf{t}$ : “어디로 이동할지” (절대적)

# MoLEContextSubnet: Context-Aware & Task-Adaptive Design

Coupling Layer 내부 핵심 모듈: 변환 파라미터 ( $s$ ,  $t$ ) 생성

## 1. Spatial Structure Recovery & Context Extraction

- 일반 NF: 1D 벡터 평탄화 → 공간적 인접성 소실
- 본 모듈: 2D 이미지 재구성 후  $3 \times 3$  DepthwiseConv로 문맥 추출
- 게이팅 적용:  $\text{ctx} = \alpha \cdot \mathbf{c}(\mathbf{x})$ , where  $\alpha = \alpha_{\max} \cdot \sigma(\theta_{\text{scale}})$

## 2. 비대칭 네트워크 설계

Context-Aware  $s$ -network

$$\mathbf{s} = \text{MLP}([\mathbf{x}; \text{ctx}])$$

Context-Free  $t$ -network

$$\mathbf{t} = \text{MLP}(\mathbf{x})$$

- 결함 = 주변과의 불연속성
- 국소적 대비(Local Contrast) 포착
- 분포 중심 이동은 패치 고유 특성
- 문맥 추가 시 불필요한 간섭

가역성 보장

LoRA는 subnet 내부 서열 레이어에만 적용 ↳ Coupling 그룹의 가역성에 영향 없음

# Hierarchical Context Processing: 문맥 모듈 비교

## Spatial Context Mixer vs MoLEContextSubnet 내부 Context Conditioning

### Spatial Context Mixer

- 위치: Flow 입력 이전
- 역할: “이 패치가 어떤 이웃을 가지는가”를 특징 자체에 인코딩
- 성격: 전처리 모듈

### MoLEContextSubnet Context

- 위치: 각 Coupling Layer 내부
- 역할: “이 이웃 관계에서 어떤 스케일 변화가 적절한가” 결정
- 성격: 변환 가이드



### 두 모듈이 모두 필요한 이유

입력 수준 문맥 인코딩 + 변환 수준 적응적 스케일링 = 계층적 문맥 처리 (Hierarchical Context Processing)

# MoLE-Flow: Deep Invertible Adapter (DIA)

## 문제: 선형 변환의 표현력 한계

- LoRA:  $\mathbf{W} + \mathbf{B}\mathbf{A} \rightarrow$  선형 변환
- WhiteningAdapter:  $\gamma \cdot \mathbf{x} + \beta \rightarrow$  선형 변환
- 복잡한 비선형 매니폴드 차이 보상 어려움

## 해결: DIA (Deep Invertible Adapter)

$$\mathbf{z}_{\text{final}} = f_{\text{DIA}}^{(t)}(\mathbf{z}_{\text{base}}), \quad \text{with } \log |\det \mathbf{J}_{\text{DIA}}|$$

### DIA 구조:

- AffineCouplingBlock 스택
- Task별 독립 파라미터
- **Near-identity 초기화**:  $f_{\text{DIA}}(\mathbf{z}) \approx \mathbf{z}$

### Why Invertible?

- NF의 밀도 추정 속성 보존
- Jacobian 계산 가능
- 학습 안정성 향상

## 핵심 원리: 비선형 매니폴드 적응

LoRA(선형)가 표현 못하는 복잡한 분포 변환 학습. Base Flow 출력을 Task별 최적 잠재 공간으로  
비선형 매핑

# DIA: AffineCouplingBlock 상세 구조

## 각 AffineCouplingBlock의 변환 과정:

- ① **Split**:  $[x_1, x_2] = \text{Split}(x)$  (채널 절반 분할)
- ② **Subnet**:  $s, t = \text{SimpleSubnet}(x_1)$  (scale, translation 예측)
- ③ **Soft-Clamping**:  $\tilde{s} = \alpha_{\text{clamp}} \cdot \tanh(s/\alpha_{\text{clamp}})$  (수치 안정성)
- ④ **Transform**:  $y_2 = x_2 \odot \exp(\tilde{s}) + t$
- ⑤ **Concat**:  $y = \text{Concat}(x_1, y_2)$

## Log-Determinant 계산

$$\log |\det \mathbf{J}_{\text{block}}| = \sum_{i=1}^{D/2} \tilde{s}_i = \sum_{i=1}^{D/2} \alpha_{\text{clamp}} \cdot \tanh(s_i/\alpha_{\text{clamp}})$$

## MoLEContextSubnet vs DIA 역할 차이

- **MoLEContextSubnet**: “공간 인지형 범용 변환기” - Base anchor + LoRA로 범용 변환 + 작업별 적응
- **DIA**: “작업 특화 분포 보정기” - 완전히 독립적인 파라미터로 정밀 분포 보정

# DIA가 Spatial Context를 사용하지 않는 이유

MoLEContextSubnet과 DIA는 모두 Affine Coupling 기반이지만 다른 설계 철학

## MoLEContextSubnet

- “공간 인지형 범용 변환기”
- Base weights를 anchor로 범용 변환 보존
- LoRA로 작업별 적응
- Spatial context로 국소 이상 패턴에 민감
- “정상 → 표준 정규분포” 변환의 빠대

## DIA

- “작업 특화 분포 보정기”
- Task별 완전 독립 파라미터
- Spatial context **불사용**:
  - ① 잠재 공간에서 동작 - 공간 정보는 이미 MoLEContextSubnet에서 인코딩
  - ② 역할 분리: 공간 인지 변환 vs 분포 정밀 보정
  - ③ 파라미터 오버헤드 최소화

## 핵심 설계 원리

MoLEContextSubnet이 “공간 인지 변환”을, DIA가 “분포 정밀 보정”을 담당하는 **명확한 역할 분리**가 효율적

# Training Objective: Likelihood Calculation

## Normalizing Flow의 핵심: 변수 변환

$$\log p(\mathbf{x}) = \log p(\mathbf{z}_{\text{final}}) + \log |\det \mathbf{J}_{\text{total}}|$$

잠재 확률

$$\log p(\mathbf{z}) = -\frac{1}{2}\|\mathbf{z}\|_2^2 - \frac{D}{2} \log(2\pi)$$

- 표준 정규분포 가정
- 원점에 가까울수록 높은 확률

Jacobian 누적

$$\log |\det \mathbf{J}_{\text{total}}| = \log |\det \mathbf{J}_{\text{flow}}| + \log |\det \mathbf{J}_{\text{DIA}}|$$

- 각 변환의 부피 변화율
- 밀도 보정 역할

# Training Objective: Tail-Aware Loss

## 문제: 일반 NLL의 한계

- 모든 패치의 평균 NLL 최소화
- “쉬운” 정상 패치에만 집중 → 경계 학습 부족
- 정상/이상 경계의 “어려운” 패치 무시

## 해결: Tail-Aware Loss

$$\mathcal{L}_{\text{train}} = (1 - \lambda) \cdot \mathbb{E}[\mathcal{L}_{\text{all}}] + \lambda \cdot \mathbb{E}[\mathcal{L}_{\text{top-k}}]$$

### 핵심 아이디어:

- $\mathcal{L}_{\text{top-k}}$ : 상위 k% 고순실 패치
- 분포 경계(tail)에 위치한 “어려운” 샘플 집중 학습

### Why Tail?

- Anomaly는 정상 분포의 “꼬리”에 위치
- 경계 패치 학습 → 분리 능력 향상
- Hard negative mining과 유사 개념

### 핵심 원리: 분포 경계 학습 강화

쉬운 패치만 학습하면 경계가 불명확 → Tail 패치 가중치로 결정 경계 선명화

# Training Objective: LogDet Regularization

## 문제: Jacobian 불안정성

- NF의 log-determinant가 불안정 → density estimation 품질 저하
- Deep NF에서 Jacobian explosion/vanishing 문제 발생
- 변환의 volume 변화가 극단적 → 밀도 추정 왜곡

## 해결: LogDet Regularization

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{NLL}} + \lambda_{\logdet} \cdot |\log |\det \mathbf{J}||$$

### 핵심 아이디어:

- $|\log |\det \mathbf{J}|| \rightarrow 0$  유도
- Volume-preserving 변환
- 학습 안정성 향상

### Why Log-Det?

- $\log |\det \mathbf{J}| = 0 \Leftrightarrow$  부피 보존
- 극단적 축소/확대 방지
- Invertibility 보장 강화

### 핵심 원리: 변환의 부피 보존

Log-determinant를 0 근처로 정규화 → 변환이 정보를 압축/팽창하지 않고 형태만 변형

# Inference: Task Routing

핵심 문제: 추론 시 Task ID가 주어지지 않음

해결: Prototype-based Mahalanobis Router

$$t^* = \arg \min_t D_M(\mathbf{f}, t), \quad D_M(\mathbf{f}, t) = \sqrt{(\mathbf{f} - \boldsymbol{\mu}_t)^\top \boldsymbol{\Sigma}_t^{-1} (\mathbf{f} - \boldsymbol{\mu}_t)}$$

장점:

- **One-stage**: 별도의 라우팅 추론 불필요
- **공분산 고려**: 분포 형태를 반영한 거리 측정
- **스케일 불변**: 특징 차원 간 스케일 차이 자동 보정

기존 방법 대비 장점

기존: Two-stage (라우팅 → 탐지)

Ours: **One-stage** (라우팅 + 탐지 동시)

# Task Routing: Prototype Construction (상세)

## Training Phase에서 각 Task의 Prototype 구축

### Prototype 구성 요소

- 평균 벡터  $\mu_t \in \mathbb{R}^D$
- 공분산 행렬  $\Sigma_t \in \mathbb{R}^{D \times D}$

### 계산 방법

$$\mu_t = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathbf{f}_i^{(t)}$$

$$\Sigma_t = \frac{1}{N_t - 1} \sum_{i=1}^{N_t} (\mathbf{f}_i^{(t)} - \mu_t)(\mathbf{f}_i^{(t)} - \mu_t)^\top + \lambda_{\text{reg}} \mathbf{I}$$

### 특징 벡터 $\mathbf{f}$

- Backbone 마지막 layer 출력
- 이미지 레벨 특징 벡터
- $\mathbf{f}_{\text{backbone}} \in \mathbb{R}^D$

### Precision Matrix

- $\Sigma_t^{-1}$  사전 계산
- $\lambda_{\text{reg}} = 10^{-5}$  (수치 안정성)

### Adapter 활성화

선택된 Task  $t^*$ 에 따라: **LoRA**  $\{\mathbf{A}_{t^*}, \mathbf{B}_{t^*}\}$  + **Whitening**  $\{\mu_{t^*}^{\text{white}}, \sigma_{t^*}^{\text{white}}\}$  + **DIA** 활성화

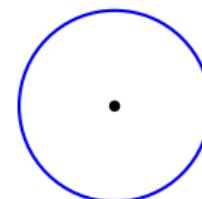
# Why Mahalanobis Distance?

## Euclidean vs Mahalanobis 거리 비교

### Euclidean Distance

$$D_E(\mathbf{f}, t) = \|\mathbf{f} - \boldsymbol{\mu}_t\|_2$$

- 모든 방향을 **동등하게** 취급
- 특징 차원 간 스케일 차이 무시
- 분포 형태 미반영



Euclidean: 원형

### Mahalanobis Distance

$$D_M(\mathbf{f}, t) = \sqrt{(\mathbf{f} - \boldsymbol{\mu}_t)^\top \boldsymbol{\Sigma}_t^{-1} (\mathbf{f} - \boldsymbol{\mu}_t)}$$

- 분포 형태 고려**: 공분산 반영
- 스케일 불변**: 자동 정규화
- 높은 구분력**: 중첩된 분포 구분 가능



Mahalanobis: 타원형 (분포 반영)

# Inference: Anomaly Scoring

## Step 1: Patch-wise Scoring

$$\text{Score}_{(h,w)} = \underbrace{\frac{1}{2} \|\mathbf{z}_{(h,w)}\|^2}_{\text{Distance Score}} - \underbrace{\log |\det \mathbf{J}|_{(h,w)}}_{\text{Distortion Score}}$$

## Step 2: Top-K Mean Aggregation

$$\text{Final Score} = \frac{1}{K} \sum_{i=1}^K S_{\text{top}}^{(i)}$$

### vs Max

- 단일 노이즈에 의한 오탐 방지

- 이상 신호 희석 방지
- 결함은 보통 군집 형성

# Full Inference Pipeline

추론 전체 흐름:

## Step 1: Feature Extraction

- 입력 이미지  $\mathbf{x}_{\text{img}} \in \mathbb{R}^{224 \times 224 \times 3}$  → WideResNet-50 통과
- 다층 특징 맵 추출 후 패치 분할 + Adaptive Average Pooling
- 공간 해상도  $(H, W) = (14, 14)$ , 임베딩 차원  $D = 768$
- 2D Sinusoidal Positional Encoding 추가

## Step 2: Normalizing Flow Transformation

- ① **Task Selection**: Mahalanobis 라우터로  $t^*$  결정
- ② **Whitening**: Task  $t^*$ 의 통계량으로 분포 정규화
- ③ **Spatial Mixer**: DepthwiseConv로 인접 패치 정보 교환
- ④ **MoLE Flow**: LoRA-equipped Affine Coupling Layers 통과
- ⑤ **DIA**: Task-specific 비선형 매니폴드 적응

## Step 3: Score Computation

$$\log p(\mathbf{x}_{h,w}) = \underbrace{-\frac{1}{2} \|\mathbf{z}_{h,w}\|_2^2 - \frac{D}{2} \log(2\pi)}_{\text{부분 사각}} + \underbrace{\log |\det \mathbf{J}_{h,w}|}_{\text{부분 사각}}, \quad S_{\text{image}} = \text{mean}(\text{top-K}(\{-\log p\}))$$

# Hyperparameter Optimization Overview

## 실험 규모

78개 이상의 실험을 수행하여 최적의 하이퍼파라미터 조합 탐색

Baseline 성능 (MVTec AD, WRN50-80ep):

Metric	Baseline	Target
Image AUC	0.9796	$\geq 0.98$
Pixel AP	0.4640	0.54 - 0.60
Routing Acc.	100%	100%

최종 달성:

- Pixel AP: 0.5350 (+15.3% 개선)
- Image AUC: 0.9824 (유지/향상)

# Individual Component Effects

각 하이퍼파라미터의 개별 효과 (vs Baseline 0.4640):

Component	Pixel AP	Delta	Impact
LogdetReg 1e-4	0.5055	+4.15%	★★★★★
ScaleCtxK5	0.4870	+2.30%	★★★★
TopK5-TailW0.5	0.4866	+2.26%	★★★★
lr 3e-4	0.4718	+0.78%	★★
DIA6	0.4606	-0.34%	★

핵심 발견

Log-determinant Regularization (1e-4)이 단일 최대 개선 효과 (+4.15%)

# Tail Weight Effect Analysis

TailWeight: 어려운 패치에 집중하는 정도

TailW	Pixel AP	Image AUC	최적 TailTopK
0.50	0.5221	0.983	5%
0.55	0.5256	0.983	5%
0.60	0.5290	0.983	3%
0.65	0.5324	0.983	1%
<b>0.75</b>	<b>0.5449</b>	0.981	2%
0.80	0.5447	0.981	3%

Trade-off 발견:

- TailW 증가 → Pixel AP 향상
- TailW > 0.75 → Image AUC 약간 하락 ( $0.983 \rightarrow 0.981$ )

# Optimization Journey

Baseline에서 최적 설정까지의 개선 과정:



최적 설정:

- TailW=0.55, TopK=5, LogdetReg=1e-4, ScaleCtxK=5, lr=3e-4

# Component Importance Analysis (MoLE6+DIA2 기준)

## Ablation Study: 컴포넌트별 Pix AP 기여도 순위

Component	w/o Pix AP	MAIN (56.18)	기여도	Impact
Tail Aware Loss	48.61	56.18	+7.57%p	★★★★★
Whitening Adapter	48.84	56.18	+7.34%p	★★★★★
LogDet Regularization	51.85	56.18	+4.33%p	★★★★
Spatial Context	52.93	56.18	+3.25%p	★★★
Pos Embedding	53.99	56.18	+2.19%p	★★
Scale Context	54.52	56.18	+1.66%p	★★
LoRA	55.31	56.18	+0.87%p	★

### 핵심 발견:

- ① **Tail Aware Loss**: 단일 최대 개선 (+7.57%p)
- ② **Whitening Adapter**: 두 번째로 중요 (+7.34%p)
- ③ **LoRA**: 기여도 낮음 (DIA가 역할 분담)

### 주의사항:

- MoLE $\geq$ 12: 학습 불안정
- spatial\_ctx\_k $\geq$ 5: Pix AP 급락
- tail\_weight $>$ 1.0: 역효과

# Key Trade-offs Discovered

실험을 통해 발견된 핵심 Trade-off:

## 1. DIA Blocks vs Pixel AP

DIA Blocks	Image AUC	Pixel AP
2 (MAIN)	0.9792	<b>0.5618</b>
4	0.9793	0.4735
6	0.9820	0.4606
8	<b>0.9825</b>	0.4546

↑ DIA → ↑ Image AUC, ↓ Pixel AP

## 2. TailWeight vs Stability

TailW	Image AUC	Pixel AP
0.50	<b>0.983</b>	0.5221
<b>0.70 (MAIN)</b>	0.9792	<b>0.5618</b>
0.80	0.981	0.5447

↑ TailW → ↑ Pixel AP (최적: 0.70)

최적 균형점 (MoLE6+DIA2)

권장 설정: MoLE=6, DIA=2, TailW=0.70, TopK=3, LogdetReg=1e-4  
→ Image AUC **97.92%**, Pixel AUC **97.81%**, Pixel AP **55.80%**

# Top 10 Experiments by Pixel AP

Rank	Configuration	Image AUC	Pixel AP	Routing
1	TailW0.55-TopK5-LogdetReg1e-4-ScaleCtxK5-lr3e-4	0.9824	<b>0.5350</b>	100%
2	TailW0.65-TailTopK3-TopK5-LogdetReg1e-4	0.9827	0.5324	100%
3	TopK3-TailW0.5-LogdetReg1e-4-ScaleCtxK5	0.9802	0.5317	100%
4	TopK5-TailW0.5-LogdetReg1e-4-ScaleCtxK5	0.9809	0.5317	100%
5	TailW0.6-TailTopK3-TopK5-LogdetReg1e-4-ScaleCtxK5-80ep	0.9826	0.5310	100%
6	TailW0.6-TopK5-LogdetReg1e-4	0.9827	0.5290	100%
7	TailW0.55-TopK5-LogdetReg1e-4	0.9827	0.5256	100%
8	TailW0.5-TailTopK3-TopK5-LogdetReg1e-4	0.9830	0.5242	100%
9	FullBest-80ep-lr3e-4-LoRA128-C10-DIA5-TailW0.55	<b>0.9836</b>	0.5242	100%
10	TopK5-TailW0.5-LogdetReg1e-4	0.9826	0.5221	100%

**Observation:** 모든 상위 설정에서 **Routing Accuracy 100% 유지**

# Per-Class Performance Improvement

Class	Baseline	Top Config	Improvement
carpet	0.3601	0.6167	+0.2566
bottle	0.4551	0.6774	+0.2223
leather	0.2292	0.3970	+0.1678
toothbrush	0.4028	0.5619	+0.1591
wood	0.3546	0.4453	+0.0907
hazelnut	0.5110	0.5798	+0.0688
metal_nut	0.8491	0.7776	-0.0715
cable	0.6575	0.6339	-0.0236
<b>Mean</b>	<b>0.4640</b>	<b>0.5350</b>	<b>+0.0710</b>

## Key Observations:

- **Textured classes** (carpet, leather): 가장 큰 개선
- **Fine-grained objects** (metal\_nut, cable): 약간의 성능 저하

# Hyperparameter Sensitivity Summary (MoLE6+DIA2 기준)

## lambda\_logdet (Critical)

$\lambda_{\text{logdet}}$	Img AUC	Pix AP
0 (disabled)	98.06	51.85
1e-6	98.08	51.88
1e-5	98.10	52.46
<b>1e-4</b>	97.92	<b>56.18</b>

- **1e-4**가 최적 (+4.30%p)
- 1e-6/1e-5는 효과 미미

## spatial\_context\_kernel (Critical)

Kernel	Img AUC	Pix AP
0 (disabled)	97.98	52.93
<b>3</b>	<b>97.92</b>	<b>56.18</b>
5	96.12	51.38
7	90.90	44.33

- **Kernel=3**이 최적
- Kernel $\geq$ 5에서 **급락**

Warning: Spatial Context Kernel

**kernel=5**: Pix AP -4.80%p, **kernel=7**: Pix AP **-11.85%p** (학습 불안정)

# Optimal Configuration (MoLE6+DIA2)

## Recommended Setting (최고 Pix AP)

- num\_coupling\_layers = 6
- dia\_n\_blocks = 2
- tail\_weight = 1.0
- tail\_top\_k\_ratio = 0.02
- lambda\_logdet = 1e-4
- scale\_context\_kernel = 5
- spatial\_context\_kernel = 3
- lora\_rank = 64
- lr = 3e-4
- epochs = 60

## Expected Performance

Metric	Value
Image AUC	97.92%
Pixel AUC	97.81%
Pixel AP	<b>56.18%</b>
Routing Acc.	100%

## 주의사항

- spatial\_ctx\_k ≠ 3 금지
- tail\_weight > 1.0 비권장
- MoLE ≥ 12 학습 불안정

# Experimental Setup

## Datasets

- **MVTec AD**: 15 classes, 5,354 images
- **VisA**: 12 classes, 10,821 images
- **MPDD**: 6 classes (metal parts)

## Evaluation Metrics

- Image AUROC / AP (primary)
- Pixel AUROC / AP / PRO
- Forgetting (F), BWT
- Routing Accuracy

## Implementation

- Backbone: ViT-Base-16 (frozen)
- NF: 8 Coupling Layers
- LoRA rank: 64
- DIA: 4 blocks
- Epochs: 60 per task
- LR:  $2 \times 10^{-4}$
- Input:  $224 \times 224$

**Hardware:** NVIDIA RTX 4090

# Continual Learning Scenarios

4가지 시나리오로 지속 학습 능력 검증:

## ① Scenario A: Standard 5-Task Protocol

- MVTec 15 classes → 5 tasks (각 3 classes)
- Task 0: leather, grid, transistor (텍스처 + 객체 혼합)

## ② Scenario B: Long Sequence (15-Task)

- 15개 클래스를 각각 독립 작업으로 구성
- 장기 망각(Long-term Forgetting) 분석

## ③ Scenario C: Class Order Sensitivity

- 5개 무작위 순서로 학습
- Parameter Isolation의 순서 견고성 검증

## ④ Scenario D: Task 0 Dependency Analysis

- Texture-first vs Object-first vs Mixed-first
- Base weights 학습이 후속 작업에 미치는 영향

# Comparison with SOTA Methods

비교 대상:

## Continual AD Methods

- **DNE**: 분포 저장 기반
- **UCAD**: Prompt tuning 기반 SOTA

## CL + AD Adaptation

- EWC: 중요 파라미터 정규화
- LwF: Knowledge Distillation
- Replay (5%): 데이터 저장

## Bounds

- **Joint Training**: Upper bound
- **Task-Separated**: 망각 없음, 비효율
- **Fine-tuning**: Lower bound

## 검증 목표:

- 이상 탐지 성능 우수성
- Forgetting  $\approx 0$  (Parameter Isolation)
- Routing Accuracy 100%

# Ablation Study: 컴포넌트별 중요도 분석 (MoLE6+DIA2 기준)

Configuration	Img AUC	Pix AUC	Pix AP	$\Delta$ Pix AP	Importance
<b>MAIN (MoLE6+DIA2)</b>	<b>97.92</b>	<b>97.81</b>	<b>56.18</b>	-	Baseline
<i>Loss Components (가장 중요)</i>					
w/o Tail Aware Loss	94.97	97.21	48.61	<b>-7.57</b>	★★★★★
<i>Core Adapters</i>					
w/o Whitening Adapter	97.90	97.69	48.84	<b>-7.34</b>	★★★★★
w/o LogDet Reg	98.06	97.70	51.85	<b>-4.33</b>	★★★★
w/o LoRA	97.96	97.77	55.31	-0.87	★
<i>Context Modules</i>					
w/o Spatial Context	97.98	97.54	52.93	<b>-3.25</b>	★★★
w/o Pos Embedding	97.40	97.47	53.99	-2.19	★★
w/o Scale Context	97.90	97.74	54.52	-1.66	★★

## Key Insight

Tail Aware Loss (+7.57%p)와 Whitening Adapter (+7.34%p)가 Pix AP에 가장 중요

# Training Strategy: Base Freezing의 중요성

## Base Weights Freeze vs Sequential Training

Strategy	Img AUC	Pix AP	Δ Img AUC	Δ Pix AP
<b>MAIN (Base Frozen)</b>	<b>97.92%</b>	<b>56.18%</b>	-	-
Sequential Training	60.10%	12.29%	<b>-37.82%</b>	<b>-43.89%</b>
Complete Separated	98.13%	52.49%	+0.21%	-3.69%

분석:

- **Sequential Training**: Base freeze 없이 순차 학습 → **Catastrophic Forgetting 발생**
- **Complete Separated**: 공유 weight 없이 완전 분리 → 공유 representation 학습 이점 상실
- **Base Frozen (Ours)**: Parameter Isolation으로 Forgetting 방지 + 공유 학습 이점 유지

### 핵심 결론

Base Frozen 전략이 Continual Learning에 필수: Sequential 대비 Img AUC +37.82%

# Architecture Comparison: MoLE vs DIA

Architecture	MoLE	DIA	Total	Img AUC	Pix AP	Note
MoLE+DIA	6	2	8	97.92	56.18	최적
MoLE+DIA	4	2	6	97.84	55.90	
MoLE+DIA	8	2	10	97.99	54.92	
MoLE+DIA	8	4	12	98.29	54.20	Old MAIN
DIA-Only	0	4	4	98.13	53.28	안정적
DIA-Only	0	8	8	98.19	50.74	
MoLE-Only	8	0	8	92.74	50.06	불안정
MoLE-Only	12	0	12	62.19	14.23	학습 실패

## 핵심 발견

- **MoLE-Only:** DIA 없이는 Img AUC 92.74%로 **-5.18%p 하락**, depth $\geq 12$ 에서 학습 실패
- **DIA-Only:** 안정적이나 Pix AP 53.28%로 제한적 (-2.90%p)
- **MoLE+DIA:** DIA가 MoLE 학습 안정화 → 안정성 + 정밀도 모두 달성

# Cross-Dataset Generalization

MVTec → VisA → MPDD 일반화 평가:

Dataset	Classes	Image AUC	Pixel AP	Routing Acc.
MVTec AD	15	<b>0.9824</b>	<b>0.5350</b>	100%
VisA	12	0.8566	0.2878	100%
MPDD	6	0.9019	0.2890	98.12%

VisA Analysis:

- 복잡한 텍스처와 미세 결함 → 더 도전적
- 병목 클래스: macaroni1/2 (Pixel AP < 0.1)
- ViT backbone: Image AUC 0.88 (WRN50 대비 +4.2%)

MPDD:

- 금속 표면의 유사성 → Routing 오류 (98.12%)

# Continual Learning Analysis

## Forgetting 분석:

- Parameter Isolation (LoRA + DIA) → **Forgetting**  $\approx 0$
- vs Fine-tuning: 급격한 성능 하락
- vs EWC/LwF: 완만한 하락

## Router Performance:

- 5-Task: 100% Routing Accuracy
- 15-Task: Oracle과의 Gap 분석
- Task 수 증가에 따른 Scalability 검증

## Storage Efficiency:

- MoLE-Flow: 약 2MB/Task (LoRA + DIA)
- vs Replay (5%): 50MB/Task → **25x 절약**

# Summary: Key Findings

## MoLE-Flow 최종 성능 (MVTec AD, 15 classes)

Metric	Value	비고
Image AUC	97.92%	MoLE6+DIA2
Pixel AUC	97.81%	
Pixel AP	<b>56.18%</b>	
Routing Accuracy	100%	

### 핵심 발견사항:

- ① 가장 중요한 컴포넌트: Tail Aware Loss (+7.57%p), Whitening Adapter (+7.34%p)
- ② 가장 민감한 하이퍼파라미터: spatial\_context\_kernel (kernel $\geq$ 5에서 급락)
- ③ 최적 아키텍처: MoLE=6, DIA=2 (총 8 blocks)
- ④ Parameter Isolation 효과: Sequential Training 대비 +37.82%p Img AUC

### 결론

Parameter Isolation (LoRA + DIA) + Tail-Aware Loss + Whitening Adapter가  
Continual Anomaly Detection에서 최적 성능 달성

# Key Results (MoLE6+DIA2 Configuration)

## MVTec AD (15 classes, 1x1 CL)

- Image AUC: **97.92%**
- Pixel AUC: **97.81%**
- Pixel AP: **56.18%**
- Routing Acc.: **100%**

## 컴포넌트 기여도 (Pix AP 기준)

- ① Tail Aware Loss: **+7.57%<sub>p</sub>**
- ② Whitening Adapter: **+7.34%<sub>p</sub>**
- ③ LogDet Reg: **+4.33%<sub>p</sub>**
- ④ Spatial Context: **+3.25%<sub>p</sub>**

## 핵심 하이퍼파라미터

Parameter	최적값	민감도
tail_weight	1.0	높음
lambda_logdet	1e-4	높음
spatial_ctx_k	3	높음
scale_ctx_k	5	중간
lora_rank	16-64	낮음

## Warning

- $\text{spatial\_ctx\_k} \geq 5$ : 성능 급락
- $\text{tail\_weight} > 1.0$ : 역효과

# Future Work

## 1. 성능 향상 방향

- Pixel AP 0.6+ 달성을 위한 추가 최적화
- 고해상도 입력 ( $448 \times 448$ ) 적용
- DINOv2 ViT-L/H 등 강력한 backbone 탐색

## 2. 일반화 강화

- VisA 병목 클래스 (macaroni) 특화 전략
- Multi-scale 평가 양상을
- Class-specific hyperparameter tuning

## 3. 실용화 연구

- 실시간 추론 최적화
- 경량화 (Quantization, Pruning)
- 온라인 학습 시나리오 확장

# Thank You!

Questions?

**GitHub:** [github.com/your-repo/moleflow](https://github.com/your-repo/moleflow)  
**Contact:** author@example.com