

# SemanticKITTI 데이터셋 기반 PointTransformer 모델 재현

## 실험 목적 및 배경

SemanticKITTI는 시퀀스 기반의 라이다 포인트 클라우드 데이터셋으로, 3D 시맨틱 세분화 연구에 널리 사용됩니다. 본 실험의 목적은 PointTransformer 구조를 SemanticKITTI에 적용하여 semantic segmentation 성능을 검증하는 것입니다.

## 데이터셋 구성 및 전처리

SemanticKITTI는 '00'부터 '10'까지 총 11개의 sequence로 구성되어 있지만, 각 프레임당 포함된 포인트 수가 매우 많고, GPU 메모리 등의 자원에 제한이 있기 때문에 모든 sequence를 사용하기는 어렵습니다. 따라서 본 실험에서는 메모리 효율을 고려하여 일부 sequence만을 선택하여 사용하였습니다.

- 사용한 시퀀스: 00, 01, 02 (train), 03 (validation/test)
- 입력 포인트: 단일 프레임의  $(x, y, z)$  좌표와 intensity 값
- 클래스 수: 19개
- `voxel_size = 0.05`: 포인트 클라우드를 3D voxel grid로 다운샘플링할 때 사용되며, 단일 voxel의 한 변의 길이를 5cm로 설정
- `voxel_max = 80000`: 한 프레임에서 사용할 최대 voxel 개수를 80,000개로 제한

PointTransformer의 입력은 (`coord`, `feat`, `batch_offset`)이며, 전체 네트워크는 다음과 같은 단계로 구성됩니다:

- **Input**: 포인트의 3D 좌표 (`coord`), 포인트 특징 벡터 (`feat`), batch offset 정보 (`batch_offset`)를 입력으로 사용합니다.
- **Encoder**: 여러 단계의 **Transition Down** 연산과 **PointTransformerBlock**을 반복적으로 적용하여 포인트 수를 점진적으로 줄이면서 고수준 특징을 추출합니다.
- **Decoder**: 시맨틱 분할을 위한 high-resolution 복원을 위해 **Transition Up** 모듈을 사용하며, Encoder의 중간 단계 출력을 skip connection으로 연결합니다.
- **Output head**: 최종 포인트별 feature에 대해 MLP를 적용하여 point-wise class logits를 출력합니다.

## Model Architecture

### TransitionDown

TransitionDown은 포인트 수를 줄이고, 더 추상적인 특징을 추출할 수 있도록 포인트 클라우드를 하위 해상도로 다운샘플링하는 역할을 수행합니다.

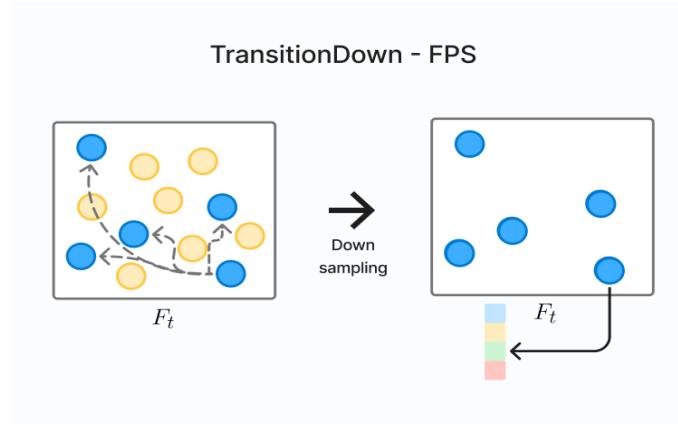


Figure 9: Transition down

- **Furthest Point Sampling (FPS):** 선택된 포인트 집합에서 가장 멀리 떨어진 포인트를 반복적으로 선택하여 공간 전체를 고르게 대표하는 서브샘플을 추출합니다.
- **k-NN:** 선택된 중심점을 기준으로 주변 이웃 포인트를 수집합니다.
- **Max Pooling:** 이웃 포인트들의 특징을 집계하여 중심점에 새로운 feature를 생성합니다.

예를 들어, 먼저, Furthest Point Sampling (FPS)을 통해 기존 포인트들 중에서 서로 멀리 떨어진 포인트들을 반복적으로 선택함으로써, 전체 공간을 고르게 대표하는 중심점들을 선택합니다. 그림의 파란색 점들이 바로 FPS로 선택된 중심 포인트들입니다. 이후 각 중심점을 기준으로 KNN을 적용하여, 해당 중심점 주변에 위치한 이웃 포인트들을 수집합니다. 이때 선택된 주변 이웃 포인트들은 노란색 점으로 나타나 있습니다. 마지막으로, 수집된 이웃 포인트들의 feature들을 Max Pooling 연산을 통해 집계하여, 각 중심점에 대응하는 새로운 feature를 생성합니다. 이 구조는 연산 효율성을 높이고, receptive field를 점진적으로 확장하는 데 기여합니다.

### Point TransformerLayer

Point TransformerLayer는 각 포인트의 지역 이웃에 대해 vector attention 기반의 self-attention 연산을 수행하여, 위치 정보를 반영한 정교한 특징 집계를 가능하게 합니다. 주어진 포인트의 feature를 주변 이웃들과의 관계를 통해 업데이트하며, 위치 차이와 채널 간의 상호작용을 모두 학습합니다.

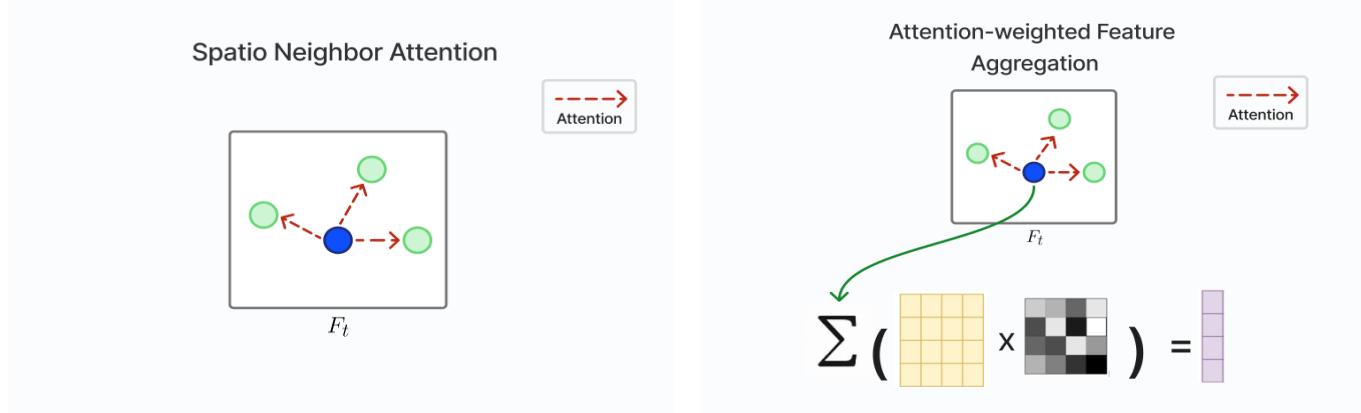


Figure 10: Spatio Neighbor Attention

- 입력 feature는 각각 Query, Key, Value로 선형 투영되며, attention 연산의 기반이 됩니다.
- Key는 query 포인트 주변의 이웃들로부터 queryandgroup 연산을 통해 그룹화되며, 위치 좌표(xyz)도 함께 포함됩니다.

- 상대 위치는 별도의 MLP (`linear_p`)를 통해 encoding되며, feature 차원에 더해집니다.
- 상대적 위치가 반영된 Key와 Query의 feature 차이 + position encoding을 더한 결과를 통해 attention weight 생성을 위한 임베딩을 계산합니다. 임베딩은 또 다른 MLP (`linear_w`)를 통해 벡터 attention weight로 변환되며, softmax로 정규화됩니다.
- Value와 position encoding의 합에 attention weight를 곱해 최종 attention feature를 계산하고, 이웃 축에 대해 sum하여 출력 feature로 만듭니다.

### TransitionUp

TransitionUp 모듈은 디코더 단계에서 고해상도 포인트 특징을 복원하거나, 같은 해상도에서 skip connection feature를 통합하는 데 사용됩니다. 입력에 따라 두 가지 방식으로 동작합니다.

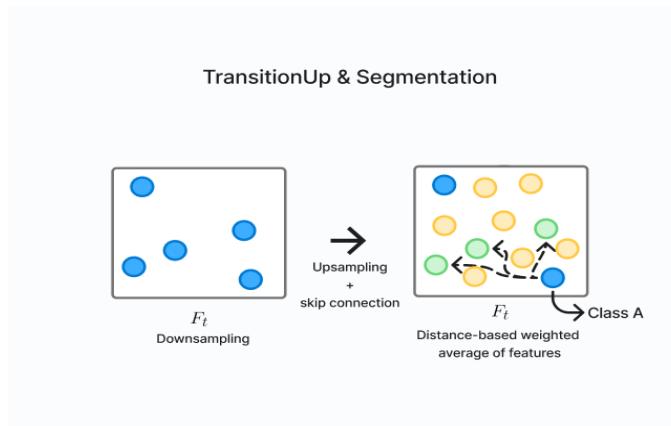


Figure 11: Transition up

- 동일 해상도 통합:** 같은 해상도의 feature만 주어진 경우, 먼저 각 배치에 대해 전체 포인트들의 feature 평균을 구합니다. 이렇게 얻은 global feature는 모든 포인트의 feature에 concat(연결)되어, 각 포인트가 전체 문맥 정보를 함께 갖도록 합니다. 이후, 연결된 feature는 MLP를 통해 정제되어 최종 출력 feature로 변환됩니다.
- Up sampling:** 해상도가 낮은 point의 feature는 먼저 선형 변환을 거친 후, 3-NN 기반의 interpolation을 통해, 고해상도 fine point 위치로 feature를 전파합니다. 이와 동시에, fine point의 기존 feature도 별도로 선형 변환되며, 이렇게 변환된 coarse feature와 더해져 skip connection을 형성합니다.

예를 들어, 좌측 박스의 파란색 포인트들이 저해상도 point 집합을 나타냅니다. 이 포인트들은 이전 다운샘플링 과정에서 선택된 대표 포인트이며, 각 포인트는 feature 정보를 가지고 있습니다. 오른쪽 박스에는 고해상도 포인트 집합이 나타나며, segmentation 등의 최종 출력이 이루어질 위치입니다. 노란색 점은 좌측의 파란 점들과 3-NN 관계를 기반으로 연결됩니다. 노란색 포인트 하나가 주변에 있는 파란색 3개 포인트와 연결되어 있다고 하면, 이때 이 노란 점의 feature는 단순히 파란 점 3개의 feature를 평균내는 것이 아니라, 거리 기반 가중 평균(distance-weighted average) 방식으로 계산됩니다. 가까운 파란 점일수록 더 큰 영향을 주는 방식입니다. 결과적으로, 각 노란색 포인트는 3-NN으로부터 보간된 feature와 skip connection을 통해 전달된 같은 해상도 feature의 조합을 통해 최종 feature를 갖게 됩니다.

### 학습 세팅

- Batch size: 4
- Base learning rate: 0.5
- Epochs: 1

- Learning rate multiplier: 0.1
- Momentum: 0.9
- Weight decay: 0.0001
- Dropout rate: 0.5
- Optimizer: SGD
- Learning rate Scheduler: MultiStepLR
- Loss function: CrossEntropyLoss
- Metric : Accuracy, mIoU, Loss, Time Consistency

## 실험 결과

전체 데이터의 크기가 크고 연산량이 많아, 한 에폭을 도는 데 시간이 오래 걸리는 문제가 있었습니다. 특히 GPU 메모리 및 연산 자원의 한계로 인해 학습 속도에 제약이 있었으며, 이에 따라 모든 에폭을 소화하기 어려운 상황이었습니다. 그럼에도 불구하고 batch size를 200까지 증가시켜 학습을 진행한 결과, 해당 구간에서 약 80% 수준의 accuracy를 얻을 수 있었고, 본 보고서에서는 해당 조건(batch size 200)에서의 결과를 기반으로 분석을 수행하였습니다.

- Accuracy & Loss  
Best Accuracy: 0.8647, Minimum Loss: 0.4835

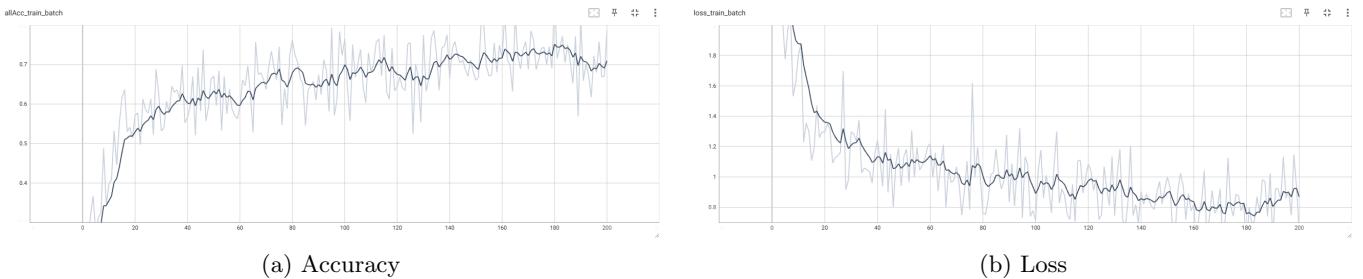


Figure 12: Accuracy & Loss

- mIoU & Time Consistency  
Best mIoU: 0.4763, Best Time Consistency: 0.7538

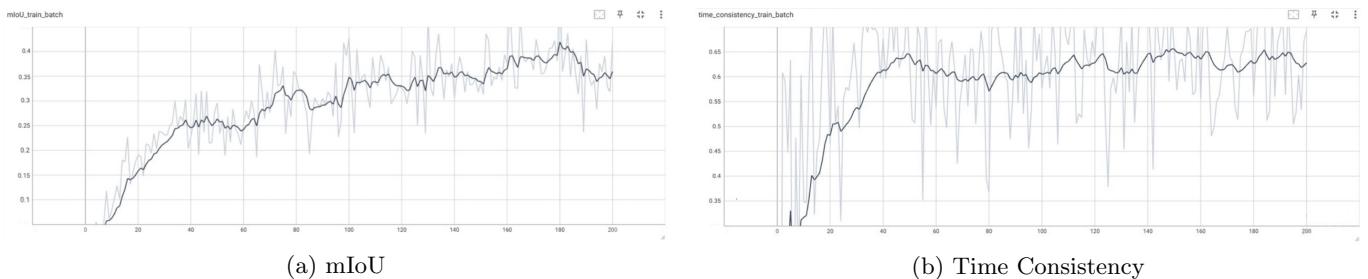


Figure 13: mIoU & Time Consistency

- Result

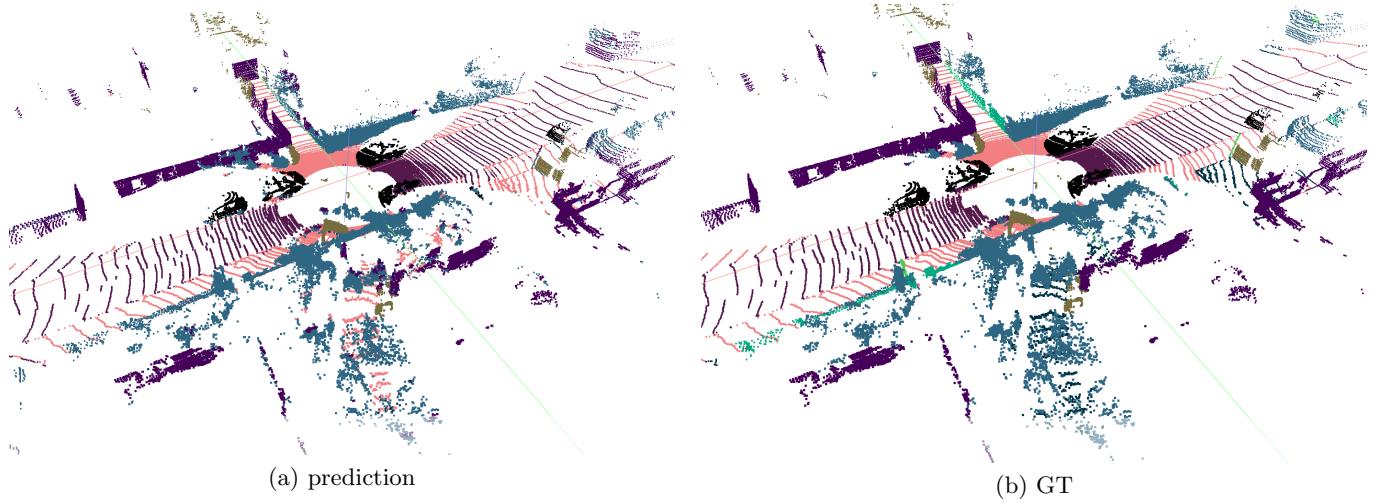


Figure 14: Comparison of Prediction and Ground Truth

## 분석 및 한계점

단일 프레임만 처리하는 경우, 연속된 프레임에서 같은 객체에 대해 서로 다른 라벨이 할당될 수 있어 시간적으로 일관된 semantic 결과를 보장하지 못합니다.

## SemanticKITTI 데이터셋 기반 4D 포인트 클라우드 실험 구현

### 실험 목적 및 배경

기존의 3D semantic segmentation 모델은 각 프레임을 독립적으로 처리하기 때문에 시간적 일관성을 보장하지 못하며, 연속된 프레임 간의 객체 추적이나 동적 장면 해석에도 한계가 있습니다. 본 연구에서는 시간축을 포함한 4D 포인트 클라우드 표현을 도입함으로써, 시계열 정보의 연속성을 활용하고 단일 프레임 기반 segmentation 보다 향상된 성능을 기대할 수 있다는 가설을 세웠으며, 이를 실험적으로 검증하고자 합니다.

### 제안 방법 1

#### Method

- 데이터셋 구성 및 전처리

본 연구에서는 시계열 정보를 활용하기 위해 SemanticKITTI 데이터셋의 연속된 세 프레임 ( $t-2$ ,  $t-1$ ,  $t$ )을 입력으로 사용하였습니다. 각 프레임의 포인트 클라우드는 개별적으로 불러와 병합되며, 포인트마다 해당 프레임의 상대적 시간 정보를 의미하는 frame index (-2, -1, 0)가 feature에 포함됩니다.

- 각 프레임의 포인트는 ( $x$ ,  $y$ ,  $z$ ) 좌표와 intensity 값을 포함하고, frame index는 feature로 추가되어 총 feature는 [intensity, time]로 구성됩니다.
- 세 프레임의 포인트를 모두 concat한 후, voxelization을 통해 downsampling을 수행하고, label은 마지막 프레임( $t$ )에 대해서만 부여됩니다.
- voxel size는 0.05m로 설정되었으며, 최대 voxel 개수는 학습 메모리에 맞춰 제한할 수 있도록 voxel\_max 옵션을 적용하였습니다.
- 학습용 시퀀스는 00, 01, 02, 검증 시퀀스는 03을 사용하였으며, 각 시퀀스에서 연속된 3프레임을 sliding window 방식으로 추출하여 학습 샘플로 사용합니다.

- Model Architecture

본 실험에서는 기존 3D 포인트 클라우드 기반의 semantic segmentation 네트워크인 PointTransformer 구조를 기반으로 하되, 연속적인 시계열 데이터를 활용할 수 있도록 시간 정보를 통합한 4D 형태로 모델

아키텍처를 확장하였습니다. 전체 네트워크는 U-Net 스타일의 인코더-디코더 구조를 따르며, 각 구성 요소는 다음과 같이 구성됩니다.

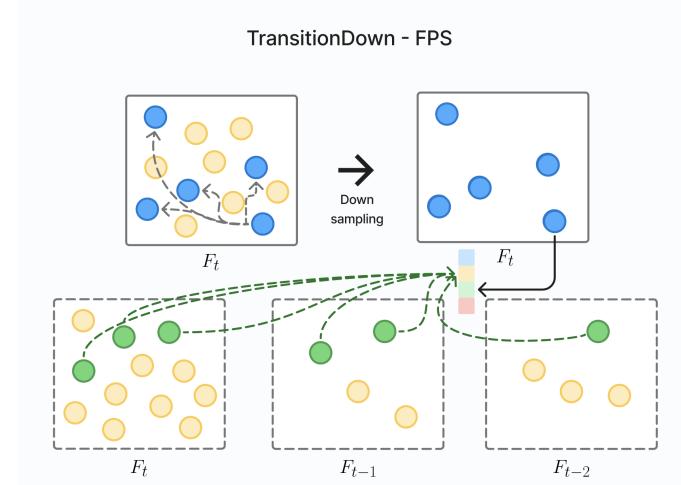


Figure 15: Transition Down

- **Temporal-aware Sampling (TransitionDown):** 다운샘플링 단계에서는 현재 시점(frame index = 0)에 해당하는 포인트만 대상으로 FPS(Furthest Point Sampling)를 수행하여 대표 포인트를 선택합니다. 이렇게 함으로써 시간 축의 불필요한 간섭 없이 현재 프레임의 대표성을 유지할 수 있습니다. 이후 선택된 포인트 주변의 이웃은 과거 프레임의 포인트까지 포함하여 k-NN을 적용하여 neighborhood를 형성합니다.

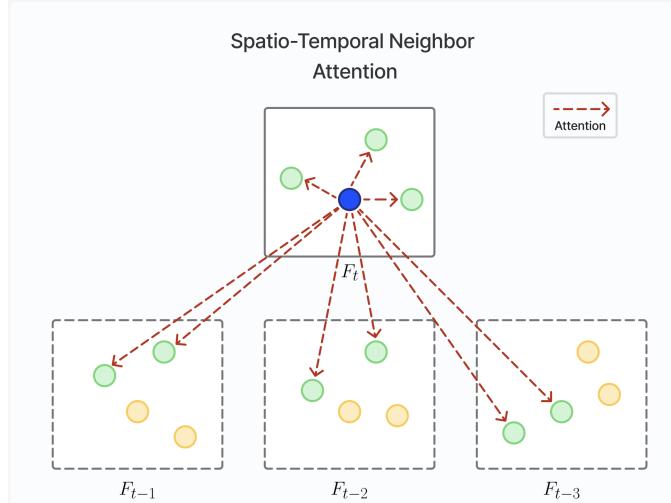


Figure 16: Point Transformer Layer

- **Spatio-temporal Attention (PointTransformerLayer):** 각 쿼리 포인트에 대해 attention weight를 계산할 때, 이웃 포인트와의 상대적 위치뿐만 아니라 시간 차이( $\Delta t$ )를 고려합니다. 시간 차이는 프레임 간 제곱 차이 ( $\Delta t$ )<sup>2</sup>에  $\lambda$  가중치를 곱하여 거리 기반 attention에 더해지며, 이는 과거 프레임의 feature가 현재 프레임에 얼마나 영향을 줄지를 제어합니다. 이로 인해 시간 감쇠(temporal decay)가 적용된 attention map이 생성되어, 공간-시간적으로 일관된 feature aggregation이 가능해집니다.

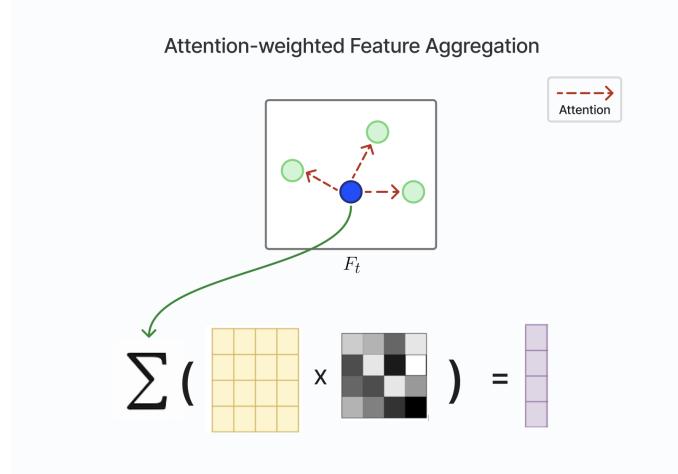


Figure 17: Enter Caption

- **Point-wise Feature Aggregation:** attention weight는 각 이웃 포인트의 feature 및 위치 차이 (position embedding)에 대해 곱해진 후 가중 평균되어 쿼리 포인트의 새로운 feature로 집계됩니다. 이는 기존 PointTransformer의 aggregation 방식에 시간 기반 weight 조절이 추가된 형태입니다.

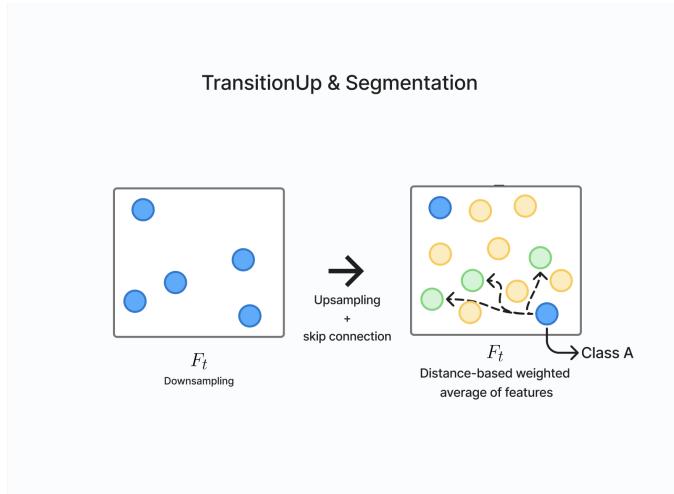


Figure 18: Enter Caption

- **Decoder (TransitionUp):** 업샘플링 단계에서는 coarse-level feature를 finer-level resolution으로 복원하기 위해 interpolation 연산을 수행하며, 이를 skip connection된 encoder feature와 통합합니다. 특히, decoder의 첫 블록에서는 coarse feature가 없기 때문에 skip feature만으로 context 정보를 강화하는 경로가 존재합니다. Interpolation은 3-NN 기반 거리 가중치로 이루어지며, 시간 정보는 interpolation에는 직접 활용되지 않지만 이전 단계에서 학습된 temporal-aware feature가 전달되므로 시계열 일관성을 유지됩니다.
- **Classification:** 최종 decoder output에 MLP 기반 classification head를 적용하여 현재 프레임(t)의 각 포인트에 대한 semantic 클래스를 예측합니다.

이러한 구조적 확장은 기존의 3D 모델이 갖는 시간적 비일관성 문제를 완화하고, 연속적인 포인트 클라우드에서 발생하는 객체의 motion 흐름 및 scene 변화를 보다 안정적으로 반영할 수 있게 하고자 합니다.

- **학습 세팅**  
기존 3D point cloud 실험과 동일하게 진행하였습니다.

## Result

- 실험 결과

  - Accuracy & Loss

Best Accuracy: 0.7950 , Minimum Loss: 0.6528

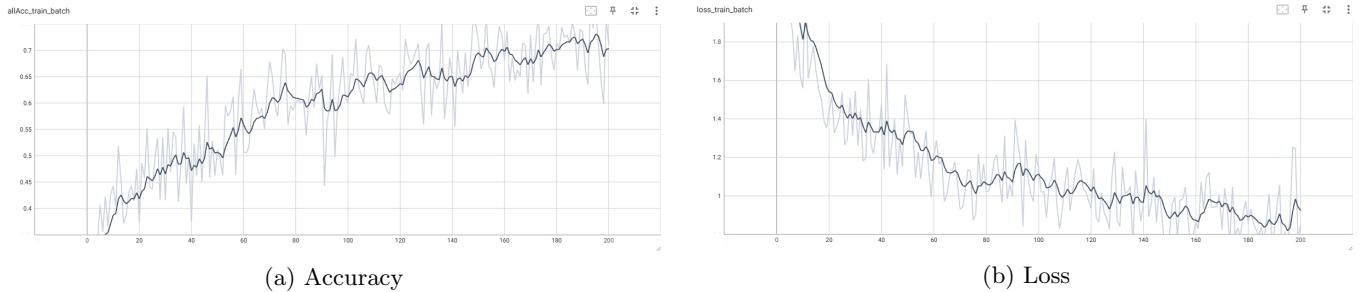


Figure 19: Accuracy & Loss

  - mIoU & Time Consistency

Best mIoU: 0.4521, Best Time Consistency: 0.9312

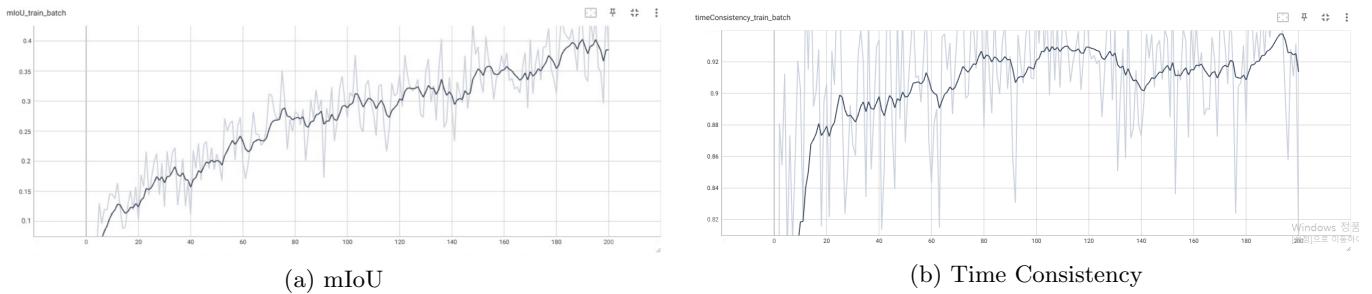


Figure 20: mIoU & Time Consistency

  - Result

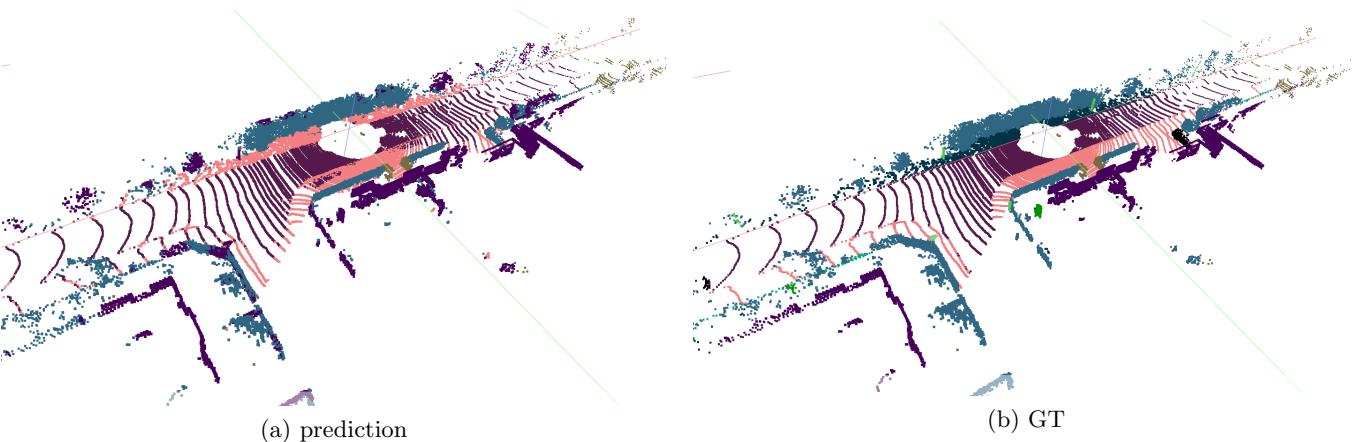


Figure 21: Comparison of Prediction and Ground Truth

- 분석

기존 단일 프레임 기반 실험에서는 KNN을 위한 이웃 개수인 nsample을 16으로 고정하여 사용하였습니다. 그러나 4D 구조에서는 메모리 사용량을 고려하여 전체 nsample 수는 유지하되, 이를 시간적으로 분할하여 현재 프레임과 과거 프레임 모두에 이웃을 할당하는 방식으로 조정하였습니다. 즉, 단일 프레임에서는

16개의 이웃을 모두 현재 프레임 내에서 찾았다면, 4D 구조에서는 예를 들어 현재 프레임에서 8개, 과거 두 프레임에서 각각 4개씩 이웃을 선택하여 시간 축 전반에 걸친 neighborhood 구성을 반영하였습니다.

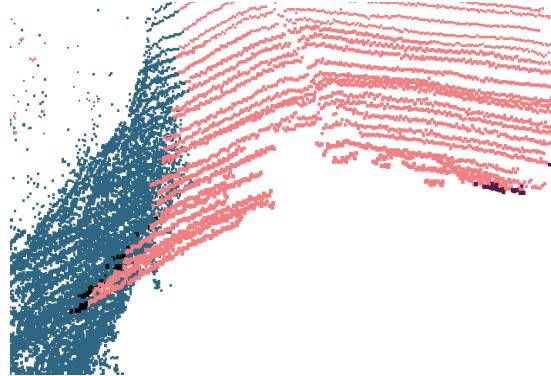


Figure 22: Point Cloud Distribution

기존 단일 프레임만 사용했을 때에 비해 Time Consistency는 향상했지만, Accuracy와 mIoU 값이 떨어지고, loss가 증가하는 경향을 보였습니다.

Figure 22를 보면, SemanticKITTI와 같이 한 프레임에 10만 개가 넘는 포인트가 존재하는 경우(포인트가 많은 경우), 현재 프레임 내에서 16개의 이웃을 선택하더라도 이미 충분히 촘촘한 국소 정보를 반영할 수 있습니다. 오히려 시간 축 전반으로 이웃을 분산시키면, 국소 표현의 밀도가 낮아져 정확도가 오히려 저하되는 경향이 나타났습니다. 이는 현재 프레임에서 밀집된 고품질 이웃 정보를 활용하는 것이 정밀한 semantic segmentation 결과에 더 효과적일 수 있다 판단하였습니다.

이를 증명하고자 추가 실험을 진행하였습니다

## Additional Experiments

해당 실험은 3D 연구와 4D 연구를 동일한 실험 환경에서 추가로 3D와 4D 모두 동일하게 이웃수를 더 늘리고, 4D는 과거프레임의 입력을 t-1, t-2, t-3 총 3프레임을 받을 때 성능이 더 저하될 것인지 확인해보고자 진행했습니다.

### - 데이터셋 구성 및 변경된 모델 구조

제한된 메모리 환경에서는 이웃 수(`nsample`)를 증가시키는 방식으로는 모델을 실행할 수 없었습니다. 따라서 본 연구에서는 기존 네트워크 구조를 경량화하는 방향으로 조정하였습니다.

첫째, `voxel_size`를 기존 0.05에서 0.08로 증가시켜 입력 포인트 수 자체를 줄였습니다.

둘째, `downsampling` 단계의 `stride`를 [1, 4, 4, 4, 4]에서 [4, 4, 4, 4, 4]로 변경하여 초기 단계에서도 포인트 수를 대폭 줄이고 연산량을 감소시켰습니다.

셋째, 각 인코더 및 디코더 단계에 적용된 PointTransformer 블록의 개수를 [2, 3, 4, 6, 3]에서 [2, 3, 4, 3, 3]으로 줄여 전체 모델의 파라미터 수를 감소시켰습니다.

모델을 경량화한 만큼 확보된 메모리 여유를 활용하여, 각 레이어에서 사용하는 이웃 수를 증가시킬 수 있었습니다. 기존에는 각 레이어마다 [8, 16, 16, 16, 16]개의 이웃을 사용했으나, 이를 [16, 32, 32, 32, 32]로 두 배씩 확장함으로써 더 넓은 국소 영역 정보를 활용할 수 있도록 하였습니다.

이웃 수 (`nsample`)가 증가함에 따라, 4D 실험에서는 더 많은 과거 프레임을 활용할 수 있게 되었습니다. 기존 실험에서는 t-1, t-2 두 프레임만을 사용하였으나, 이번 실험에서는 t-1, t-2, t-3 총 세 프레임의 정보를 활용하였습니다.

### - 실험 결과

1. 경량화된 모델을 활용한 단일 프레임 3D 포인트 클라우드 실험

메모리 제약으로 인해 네트워크 구조를 경량화한 모델을 사용하여, 단일 프레임 기반의 3D semantic segmentation 실험을 수행하였습니다. 이 실험은 4D 실험과 동일한 모델 구조를 유지하면서 시간 정보 없이 단일 시점의 포인트 클라우드만을 입력으로 활용하였습니다.

### 단일 프레임 3D 실험 결과

- \* Accuracy & Loss
- Best Accuracy: 0.7762 , Minimum Loss: 0.5426

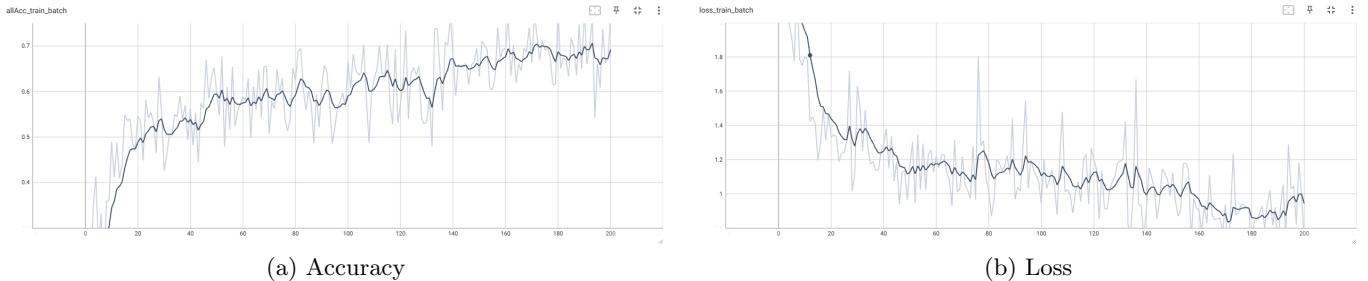


Figure 23: Accuracy & Loss

- \* mIoU & Time Consistency
- Best mIoU: 0.4487, Best Time Consistency: 0.7692

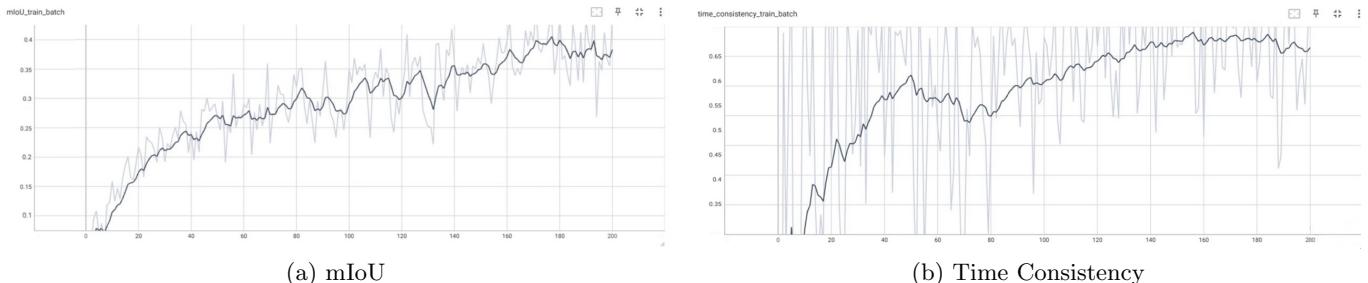


Figure 24: mIoU & Time Consistency

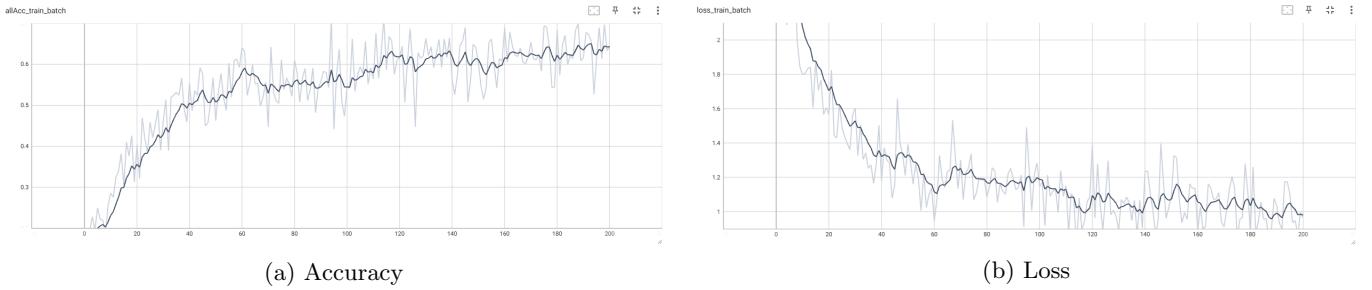
- \* Result



Figure 25: Comparison of Prediction and Ground Truth

### 연속된 4개의 프레임 4D 실험

- \* Accuracy & Loss
- Best Accuracy: 0.7124, Minimum Loss: 0.8396

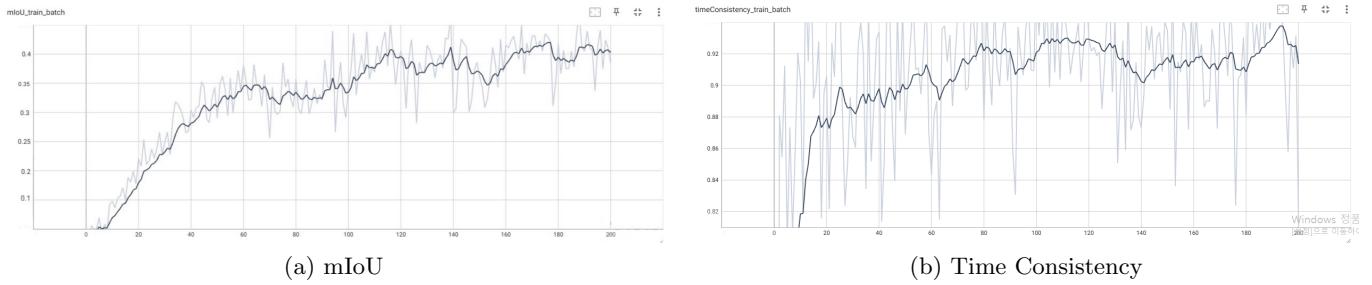


(a) Accuracy

(b) Loss

Figure 26: Accuracy & Loss

\* mIoU & Time Consistency  
Best mIoU: 0.4213, Best Time Consistency: 0.9257



(a) mIoU

(b) Time Consistency

Figure 27: mIoU & Time Consistency

\* Result

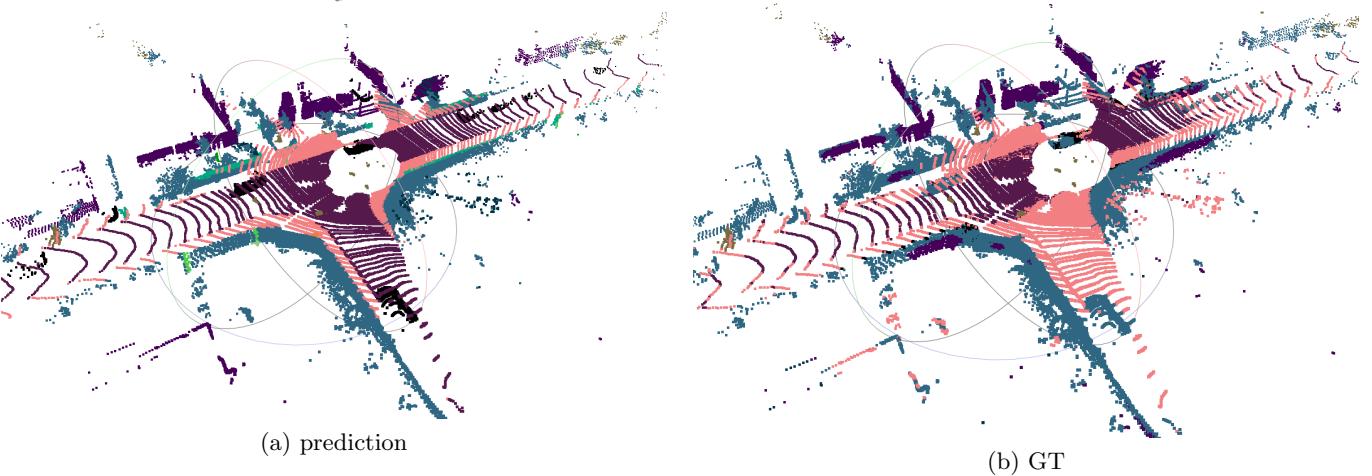


Figure 28: Comparison of Prediction and Ground Truth

### - 분석

실험 결과, accuracy와 mIoU 모두 기존에 과거 두 개의 프레임( $t-1, t-2$ )만을 참조하였을 때보다 성능이 하락하고, loss는 더 크게 관측되었습니다. 특히, 기존 단일 프레임 기반 3D 모델과 과거  $t-1, t-2$  프레임을 활용한 4D 모델을 비교했을 때, 4D 모델의 정확도가 약 5% 낮게 측정되었습니다. 더 나아가, 이웃 수(nsample)를 증가시키고 모델을 경량화한 단일 프레임 3D 실험과 과거  $t-3, t-2, t-1$  총 세 개의 프레임을 반영한 4D 실험을 비교한 결과, 4D 모델의 정확도가 약 7% 더 낮게 나타나는 현상이 관측되었습니다.

이러한 결과를 바탕으로, 이웃 수(nsample)를 시간적으로 분산시켜 더 많은 과거 프레임의 정보를 참고하려는 시도는 본래 더 풍부한 시계열 정보를 반영하고자 하는 의도를 가지고 있습니다. 그러나 이 과정에서 전체 연산량을 제어하기 위해 모델의 네트워크 크기를 축소할 수밖에 없었고, 결과적으로

모델의 표현력이 감소하게 되었습니다.

또한, SemanticKITTI와 같이 한 프레임에 약 10만 개 이상의 포인트가 존재하는 환경에서는 `nSample` 을 16으로 설정할 경우, 이는 매우 국소적인 neighborhood만을 반영하게 됩니다. 즉, 4D 구조에서는 시간적으로 분산된 이웃으로 인해 공간적인 밀도가 더욱 희석되며, 오히려 현재 프레임 내에서 밀도 높은 16개의 이웃만을 집중적으로 보는 것이 정확도 측면에서 더 유리할 것이라고 생각됩니다.

## 제안 방법 2

### 실험 목적 및 배경

기존 방식은 현재 프레임의 feature를 계산할 때, 이전 프레임에서 공간적으로 가까운 포인트의 feature를 함께 활용하여 시공간 정보를 통합하고자 하였습니다. 그러나 현재 프레임에서도 국소적인 이웃 정보는 이미 PointTransformer 내에서 반영되고 있기 때문에, 과거 프레임의 feature를 중복적으로 사용하는 것은 오히려 noise를 증가시키고 feature 혼선을 초래하여 성능 저하로 이어질 수 있다고 판단하였습니다. 따라서 본 연구에서는 이전 프레임의 feature는 사용하지 않고, 각 프레임에 대해 독립적인 PointTransformer 인코딩을 수행하되, 동일한 파라미터를 공유함으로써 모델 구조의 일관성을 유지합니다. 이후 현재 프레임의 쿼리 포인트에 대해 이전 프레임에서 공간적으로 인접한 포인트의 segmentation 결과와 양상블하여, 시간적으로 더 안정적인 예측을 도출하는 방식을 제안합니다. 이와 같은 구조는 이전 프레임의 정보가 현재 프레임의 예측에 직접적인 영향을 주되, feature 간의 간섭은 최소화함으로써, 더 나은 시공간적 일관성과 향상된 성능을 기대할 수 있습니다.

### Method

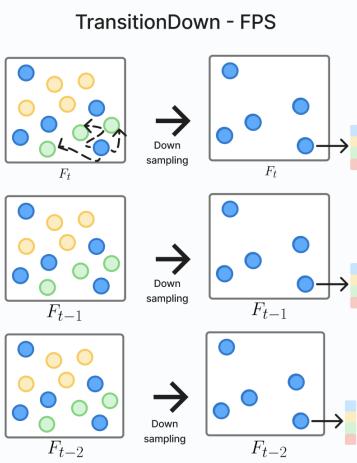
구분	제안 방법 1	제안 방법 2
입력 프레임 구성	연속된 3프레임( $t - 2, t - 1, t$ ) 포인트를 하나로 병합하여 입력 후, 각 포인트에 frame index(-2, -1, 0)를 feature로 추가	동일하게 $t - 2, t - 1, t$ 를 사용하지만, 프레임별로 독립적으로 네트워크를 통과시킴
전처리	병합 후 voxelization(0.05m), 마지막 프레임( $t$ )만 라벨 부여	전처리 방식은 제안 방법 1과 동일
다운샘플링(FPS)	FPS는 현재 프레임(0)에서만 수행하여 대표 포인트 선택한 이후 k-NN 이웃을 과거 프레임까지 포함하여 구성	각 프레임별로 독립 처리하므로, FPS도 각 프레임별로 수행
Attention 계산 방식	PointTransformerLayer에서 거리 항에 시간 차이( $\Delta t^2 \cdot \lambda$ )를 추가하여 시간 감쇠(temporal decay)가 적용된 spatio-temporal attention 수행	시간 정보는 attention에 직접 포함하지 않고 대신, 프레임별 예측 결과(logits)를 후단에서 결합
특징 합계 방식	Attention weight에 시간 기반 가중을 곱해 현재+과거 포인트 feature를 함께 합계	각 프레임별로 독립적인 logits 예측 후, 현재 프레임 기준으로 과거 프레임 예측을 KNN 정렬 후 가중 평균 양상을
시간 정보 반영 시점	인코더 내부에서 feature 수준에서 시간 정보 반영	후처리 단계에서 logits 수준에서 시간 정보 결합
Backbone 구조	단일 PointTransformer 인코더-디코더	동일한 PointTransformer 구조를 프레임별로 공유(shared weights)하여 3번 실행
출력 결합 방식	단일 스트림 출력	$\alpha_{t-2} = 0.2, \alpha_{t-1} = 0.3, \alpha_t = 0.5$ 가중치로 logits 양상을

Table 1: 제안 방법 1과 제안 방법 2의 방법론 비교

- 데이터셋 구성 및 전처리

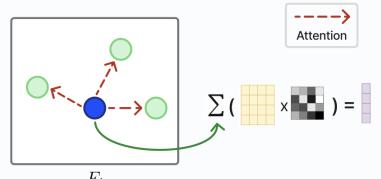
$t-2, t-1$  두 개의 과거 프레임을 함께 입력으로 활용하였으며, 데이터 전처리 방식은 제안 방법 1과 동일하게 적용하였습니다.

- Model Architecture



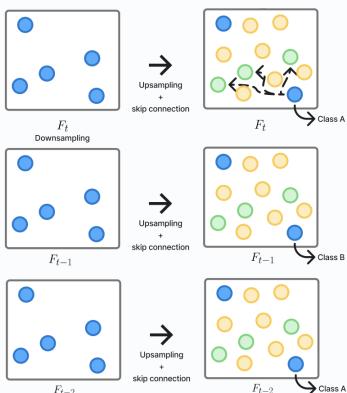
(a) TransitionDown - FPS

Spatio-Temporal Neighbor Attention



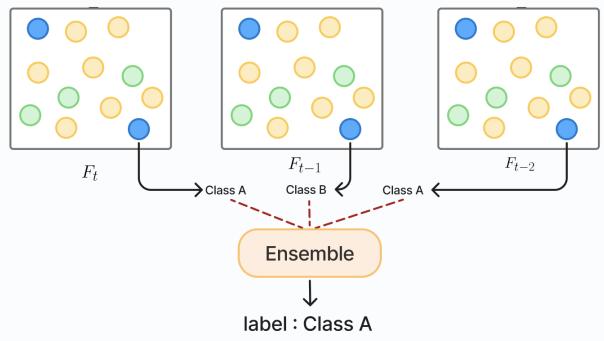
(b) Spatio-Temporal Neighbor Attention

TransitionUp & Segmentation



(c) TransitionUp Segmentation

TransitionUp & Segmentation



(d) TransitionUp Segmentation

본 연구에서 제안하는 모델은 기존 PointTransformer 기반의 인코더-디코더 구조를 유지하면서도, 시간 정보를 반영한 3프레임 기반 양상을 구조를 적용합니다. 각 프레임( $t-2, t-1, t$ )의 포인트 클라우드는 독립적으로 동일한 PointTransformer 네트워크를 통해 semantic 결과를 예측하며, 각 출력은 logit 형태로 추출됩니다. 이후 과거 프레임의 예측을 현재 프레임의 포인트 기준으로 정렬하고, 시간 감쇠 기반의 가중 평균 양상을 통해 최종 예측을 생성합니다. 세부적인 구성은 다음과 같습니다.

- Shared Backbone (단일 PointTransformer 모델)

세 프레임( $t-2, t-1, t$ )은 모두 동일한 구조의 PointTransformer 네트워크를 통하여 독립적인 예측 결과를 생성합니다. 인코더는 5단계로 구성되며 각 단계는 TransitionDown 모듈과 여러 개의 PointTransformerBlock으로 이루어집니다. 디코더 또한 TransitionUp과 동일한 블록 구조로 구성됩니다. 각 프레임의 입력은 별도의 경로로 처리되며, 예측은 클래스 수( $k = 19$ )에 대응되는 logits로 출력됩니다.

- Prediction Alignment (KNN 기반 정렬)

과거 프레임( $t-2, t-1$ )의 예측 결과는 현재 프레임( $t$ )의 포인트 위치에 정렬되도록 K-최근접 이웃

(KNN)을 이용해 정렬합니다. 이를 위해 현재 프레임의 각 포인트에 대해 공간적으로 가장 가까운 과거 프레임의 포인트를 찾고, 해당 위치의 logits 값을 복사합니다. 이 과정은 다음과 같이 정의됩니다.

$$\text{aligned\_label}_{t-i}(p) = \text{label}_{t-i} \left( \arg \min_{q \in P_{t-i}} \|p - q\|_2 \right), \quad i = 1, 2$$

여기서  $p \in P_t$ 는 현재 프레임의 포인트이며,  $P_{t-i}$ 는 과거 프레임의 포인트 집합입니다.

#### - Weighted Temporal Ensemble

정렬된 logits를 시간적 중요도에 따라 가중 평균하여 최종 예측 logits  $\hat{y}$ 를 계산합니다. 본 연구에서는 기본적으로 다음과 같은 가중치를 사용합니다:

$$\alpha_{t-2} = 0.2, \quad \alpha_{t-1} = 0.3, \quad \alpha_t = 0.5$$

최종 예측 logits는 다음 식으로 정의됩니다:

$$\hat{y} = \alpha_{t-2} \cdot y_{t-2}^{\text{aligned}} + \alpha_{t-1} \cdot y_{t-1}^{\text{aligned}} + \alpha_t \cdot y_t$$

- 학습 세팅

모델의 연산량과 메모리 사용량을 줄이기 위해 다음과 같은 경량화 전략을 적용하였습니다. 3D기반과 4D기반 모두 다음과 같은 세팅으로 진행하였습니다.

첫째, voxel 크기를 기존 voxel\_size=0.05에서 voxel\_size=0.1로 증가시켜 입력 포인트 수를 약 절반 수준으로 감소시켰습니다. 둘째, 각 네트워크 단계의 채널 수를 기존의 [32, 64, 128, 256, 512]에서 [16, 32, 64, 128, 256]으로 축소하여 모델의 전체 파라미터 수를 줄였습니다. 셋째, 시간 정보를 임베딩하는 feature의 차원 역시 기존 62차원에서 32차원으로 축소하여 temporal embedding의 연산량도 감소시켰습니다.

이 외의 나머지 구성 요소 및 처리 방식은 제안 방법 1과 동일하게 유지하였습니다.

## Result

- 실험 결과

### 단일 프레임 3D 실험 결과

- Accuracy & Loss

Best Accuracy: 0.8464, Minimum Loss: 0.5521

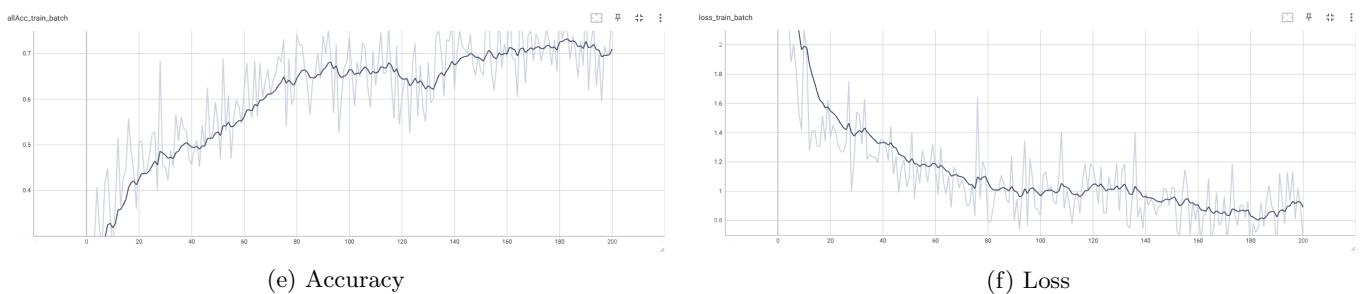


Figure 29: Accuracy & Loss

- mIoU & Time Consistency

Best mIoU: 0.4527, Best Time Consistency: 0.7629

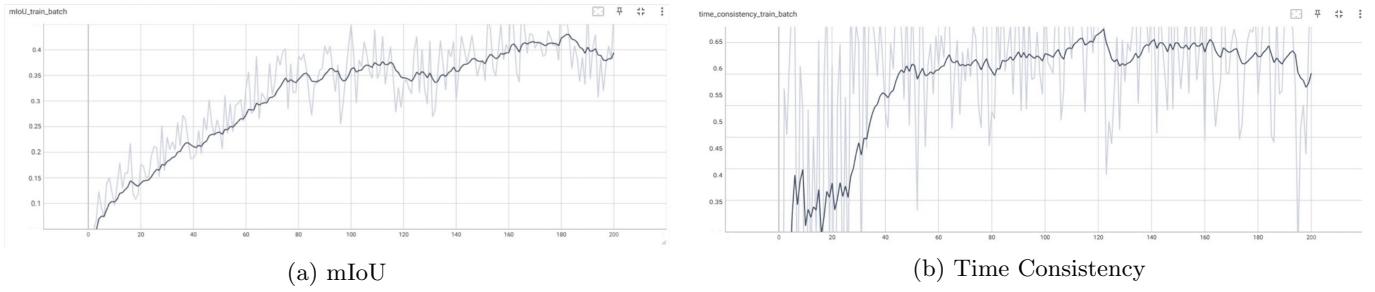


Figure 30: mIoU & Time Consistency

## - Result

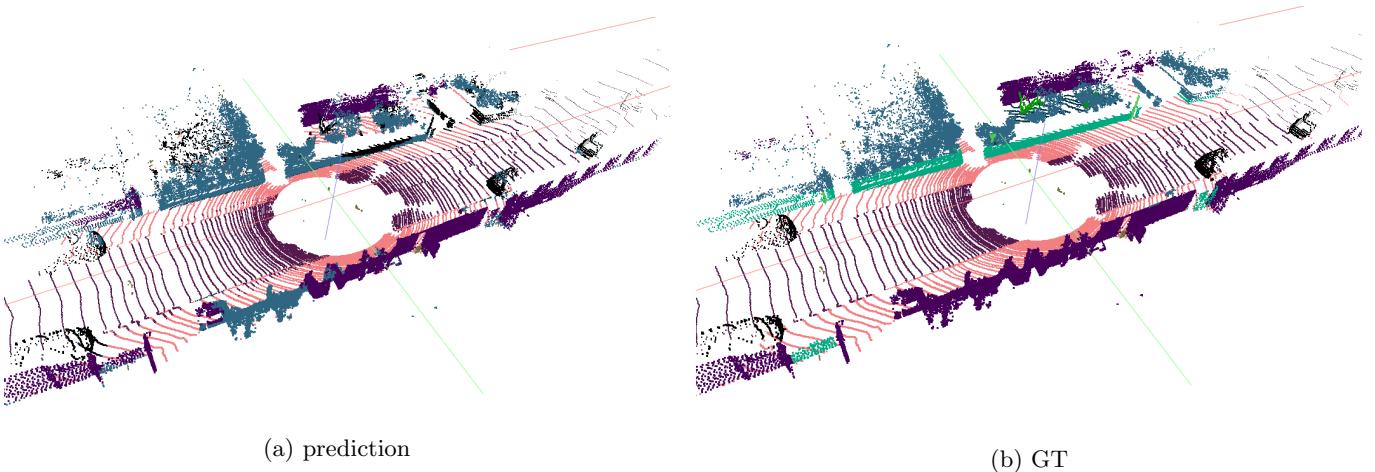


Figure 31: Comparison of Prediction and Ground Truth

## 연속된 3개의 프레임 4D 실험 결과

#### – Accuracy & Loss

Best Accuracy: 0.8830 , Minimum Loss: 0.4018

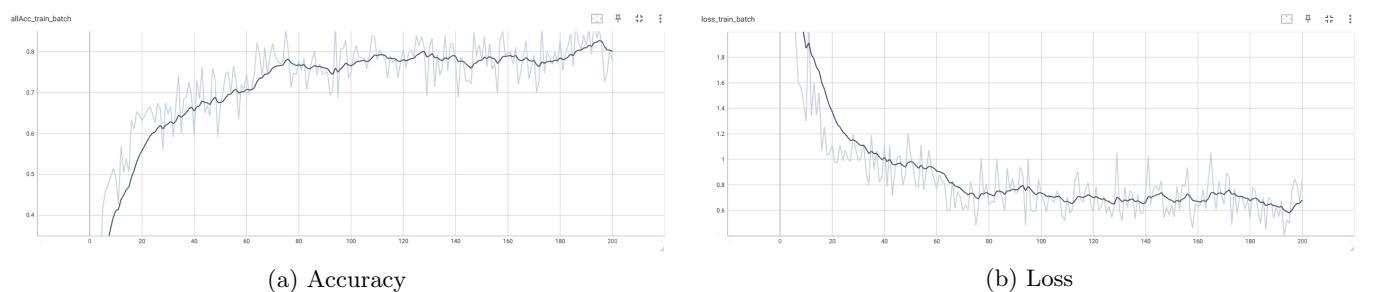


Figure 32: Accuracy & Loss

#### – mIoU & Time Consistency

Best mIoU: 0.4996, Best Time Consistency: 0.9545

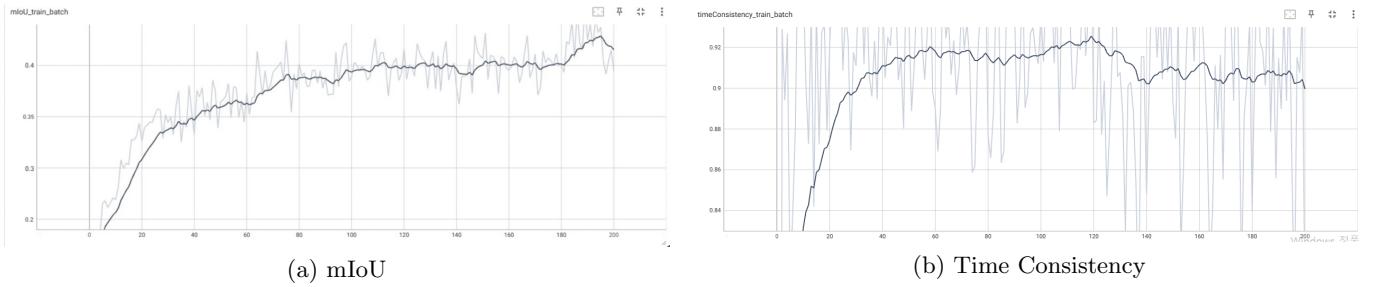


Figure 33: mIoU & Time Consistency

#### – Result

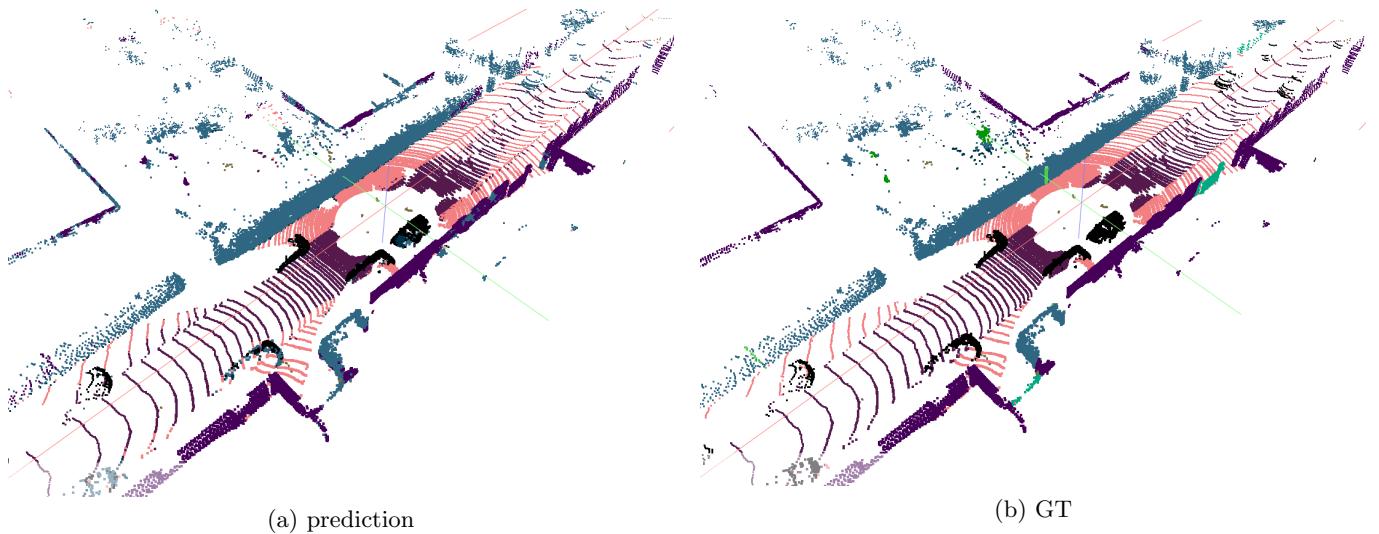


Figure 34: Comparison of Prediction and Ground Truth

#### • 분석

본 연구에서 제안한 3프레임 기반 양상을 구조는 정렬된 과거 프레임의 예측 결과를 현재 프레임 예측에 통합함으로써, accuracy 및 mIoU가 모두 향상되고 loss 값이 감소하는 결과를 보였습니다. 이러한 성능 향상은 다음과 같은 요인들에 기인하는 것으로 판단됩니다.

- 연속적인 프레임 간에는 정적 객체(예: 도로, 건물, 차량 등)가 높은 구조적 일관성을 보이기 때문에, 시간적으로 인접한 프레임에서 동일 객체가 반복적으로 관측됩니다. 이로 인해, 단일 시점에서 관측이 누락되거나 오류가 발생한 부분도 다른 시점의 정보를 통해 보완될 수 있습니다. 이러한 정보 중복 활용은 특히 경계 영역이나 객체 일부가 가려진 경우에 효과적으로 작용할 수 있습니다.
- 또한, 시점에 따라 조명이나 반사 조건이 달라지거나 occlusion의 정도가 다르기 때문에, 특정 시점에서는 어려웠던 분류가 다른 시점에서는 비교적 명확해지는 경우가 있습니다. 예를 들어, 시점  $t$ 에서는 부분적으로 보였던 물체가 시점  $t-1$ 이나  $t-2$ 에서는 더 명확히 관측되어 예측 정확도가 높아질 수 있습니다. 이러한 시점 간 보완 효과는 다수 프레임 예측을 양상을 함으로써 보다 강건한 결과를 유도합니다.
- 각 프레임의 예측은 noise 또는 센서 오차 등으로 인해 불완전할 수 있으나, 서로 독립적인 세 개의 예측을 결합하는 과정은 결과적으로 강건한 예측 유도로 이어지며, 잘못된 예측의 영향을 줄이고 일관된 클래스 결정을 내릴 수 있도록 합니다.

결과적으로, accuracy와 mIoU가 모두 향상되고 loss 값이 줄어든 것은 위에서 제시한 시공간 정보 보강과 예측 안정화 요인들이 함께 작용한 결과로 판단되며, 이는 시간 정보를 보다 효과적으로 활용하는 방향의 연구가 유의미함을 보여준다고 생각합니다.