Implementation and Reproducibility Notes on Robust 3D Human Pose Estimation from mmWave Radar via Spatio-Temporal Representation Learning

Kai-Ming Cao[1], Ming-Han Lee[1], Wei-Che Hsu[1], Kun-Ru Wu[1], Hong-Dun Lin[3], Ren-De Xie[3], Bo-Yang Chen[3], and Yu-Chee Tseng[1,2]

[1] Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
{kmc8910.cs12, mhlee.cs09, wesleyshi.en10, wufish, yctseng}@nycu.edu.tw
[2] College of AI, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
[3] Cultural & Sport Technology Department, Service Systems Technology Center, Industrial Technology Research Institute, Taiwan
{hongdunlin, itriB20662, boyang_chen}@itri.org.tw

Abstract. This companion paper to the ICPR 2026 work "Robust 3D Human Pose Estimation from mmWave Radar via Spatio-Temporal Representation Learning" details the implementation configurations and training strategies of the proposed two-stage framework to facilitate reproducibility. We elaborate on the preprocessing of the mmFi dataset, focusing on sliding-window aggregation and point normalization to mitigate extreme radar point sparsity. Furthermore, we provide in-depth implementation details for the core modules, including the Sparsity- and Permutation-Resilient Spatial Encoder (SPSE), Masked Motion Consistency Learning (MMCL), and the conditional diffusion module. Crucially, we decompose the progressive three-phase training strategy into actionable steps. Providing pseudocode, configuration excerpts, and an analysis of inference efficiency, this paper serves as a practical guide for seamlessly reproducing and extending robust mmWave-based pose estimation.

Keywords: Reproducibility · mmWave Human Pose Estimation · Spatio-Temporal Representation Learning · Radar Point Cloud · Diffusion Models.

1 Introduction

In our ICPR 2026 paper [1], we introduced a two-stage spatio-temporal representation learning and diffusion-based refinement framework to address extreme sparsity, noise, and temporal fragmentation in mmWave radar-based 3D human pose estimation. While the original work focuses on theoretical design, this companion paper presents the concrete implementation configurations, data

Table 1: Data Sparsity Analysis

Statistical Metric	Value	Description
Average points/frame	30.97	Extremely sparse
Minimum	0	Completely point-less frames
Maximum	112	High reflection environment
Standard Deviation	17.35	High variability
Zero-point frames	3,531	Absolute count
Zero-point frame ratio	1.1%	Frames with no targets
Total frames	320,760	Training + Testing

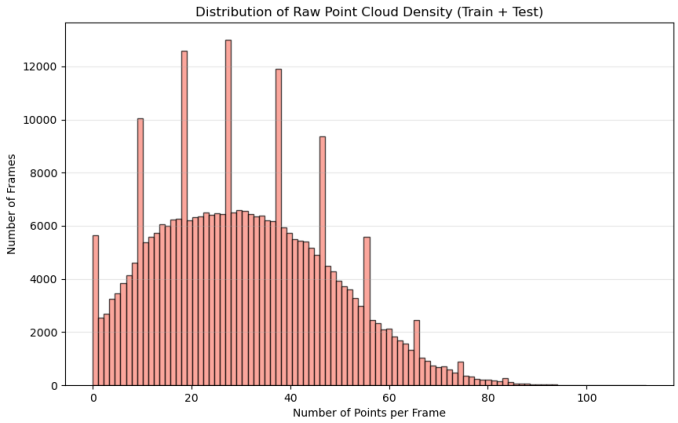

Fig. 1: Distribution of raw point cloud density across the mmFi dataset (Train + Test). The histogram highlights the extreme sparsity and high variability of points per frame, with notable peaks at specific low-density values.

preprocessing pipelines, and a step-by-step training strategy to ensure seamless reproducibility.

The remainder of this paper is organized as follows. Section 2 details the dataset preprocessing pipeline, specifically sliding-window aggregation and point normalization used to mitigate radar point sparsity. Section 3 describes the structural configurations of our core modules (SPSE, MMCL, and CCDM) and decomposes our progressive three-phase training strategy. Finally, Section 4 details the evaluation metrics (MPJPE and PA-MPJPE) and analyzes inference efficiency, parameter count, and computational complexity.

2 Dataset Processing

2.1 Data Sparsity Analysis

The mmFi [6] mmWave radar dataset exhibits an extremely sparse point cloud characteristic. As illustrated in Table 1, statistical analysis of 320,760 frames

reveals that each frame contains an average of only 30.97 target points, which is significantly lower than the point density of optical cameras or LiDAR. Furthermore, the point distribution is highly uneven, as shown in Fig. 1:

- **Zero-point frames:** A total of 3,531 frames (1.1%) completely lack radar echoes. This is primarily caused by subjects turning rapidly or the radar beam temporarily deviating from the target area.
- **Standard deviation of 17.35:** This indicates a massive discrepancy in the number of points across different trajectories. Some trajectories consistently maintain 20-40 points, while others fluctuate between 50-112 points.
- **Sparsity challenge:** The extreme deficiency of points in a single frame makes pure point cloud feature extraction highly difficult. It is imperative to rely on temporal contextual information to compensate for the missing geometric data.

2.2 Sliding-window Aggregation

```
Input:
  raw_sequences: Set of original sequences, each is a variable-length frame sequence sequence[i] = [frame_0, frame_1, ..., frame_T-1],
                 where frame_t ∈ ℝ^{N_t × C}, N_t is the number of points in that frame

Hyperparameters:
  window_size = W        # Number of frames to aggregate (e.g., W=3)
  stride = s             # Sliding stride (e.g., s=1)

Algorithm Sliding_Window_Aggregation:

1. Initialize the aggregated sequence container
   aggregated_sequences = []

2. Process sequence by sequence
   for sequence in raw_sequences:
       T = sequence.length
       aggregated_frames = []

       3. Sliding window iteration
          for t in range(0, T - W + 1, s):

              4. Get all frames within the window
                 window = [sequence[t], sequence[t+1], ..., sequence[t+W-1]]

              5. Aggregate all points within the window
                 aggregated_points = []
                 for frame in window:
                     aggregated_points.extend(frame.points)

              6. Store the aggregated frame
                 aggregated_frames.append(aggregated_points)

       7. Store the aggregated sequence
          aggregated_sequences.append(aggregated_frames)

Output:
  aggregated_sequences (Aggregated sequences, density increased by ~W times)

Effectiveness Statistics:
  Before aggregation: Mean = 30.97 points/frame
  After aggregation: Mean ≈ 30.97 × 3 ≈ 92.91 points/frame (W=3)
  Point increment: +200%
  Zero-point frame ratio: 1.1% → ~0% (almost eliminated)
```

Listing 1: Sliding Window Aggregation

To mitigate extreme sparsity, we aggregate adjacent temporal frames to increase effective point density, as detailed in Listing 1. For an original sequence of length T, $\{\mathbf{X}_t\}_{t=0}^{T-1}$, we perform sliding aggregation with a window size of $W = 3$ and stride of $s = 1$, yielding $\mathbf{X}'_t = \mathbf{X}_t \cup \mathbf{X}_{t+1} \cup \mathbf{X}_{t+2}$ for $t \in [0, T-3]$. This aggregation increases the average point density by 200% (from 30.97 to $\approx$ 92.91 points per frame) and virtually eliminates empty frames, with the zero-point frame ratio

dropping from 1.1% to near zero. Furthermore, merging adjacent frames implicitly injects temporal context, resolving geometric ambiguities caused by rapid subject turns or temporary radar beam misalignment. The resulting sequence length is reduced to $\lceil (T - W)/s \rceil + 1$, which is subsequently compensated for during coarse pose regression.

2.3 Point Normalization

```
Input:
  aggregated_frame: Aggregated point cloud ∈ ℝ^{M×C},
  where M = Actual number of points after aggregation (may be < N or > N),
  C = Feature dimension (e.g., 5 for [x, y, z, velocity, intensity])

Hyperparameters:
  target_num_points = N = 192  # Unified target number of points

Algorithm Point_Normalize:

1. Check the actual number of points
   M = aggregated_frame.shape[0]

   if M == N:   # Target reached, return directly
       normalized_frame = aggregated_frame
       return normalized_frame

   elif M < N:  # Case 1: Zero-Padding
       num_pad = N - M
       # Create zero-padded points
       pad_points = zeros((num_pad, C))
       # Concatenate original points with padded points
       normalized_frame = vstack([aggregated_frame, pad_points])    # normalized_frame: [N, C]

       return normalized_frame

   else:  # Case 2 (M > N): Random Sampling Without Replacement
       indices = random_choice(M, npoint, replace=False)
       normalized_frame = aggregated_frame[indices, :]  # normalized_frame: [N, C]

       return normalized_frame

Output:
  normalized_frame ∈ ℝ^{N×C}  (Fixed at N = 192)

Statistical Coverage:
  Based on actual data statistics after aggregation:
  - Case M < N: ~87% (requires zero-padding)
  - Case M = N: ~0.5% (extremely rare)
  - Case M > N: ~12.5% (requires random sampling)
```

Listing 2: Point Normalization

To standardize the uneven point counts after aggregation into a fixed size $N = 192$ required by deep learning models, we employ two normalization strategies (Listing 2):

Zero-Padding ($M < N$, ≈87% of frames): When the aggregated point count M is insufficient, we append $N-M$ zero vectors to the end of the point set as $\mathbf{X}'_{\text{padded}} = [\mathbf{X}'_{\text{real}}; \mathbf{0}_{(N-M)\times C}]$. Because the subsequent SPSE module utilizes max pooling, these zero vectors contribute nothing to the aggregated features ($\max(\cdot, 0) = \cdot$), avoiding information contamination while ensuring numerical stability.

Random Sampling ($M > N$, ≈12.5% of frames): When a radar point cloud contains more than N points, we randomly select N points without replacement:

$$\mathbf{X}'_{\text{sampled}} = \mathbf{X}'_{\text{real}}[\text{indices}, :], \quad \text{indices} \sim \text{Uniform}(\{0, \ldots, M-1\})$$

Ultimately, all inputs are unified into tensors of shape $[N, C]$ ($N = 192, C = 6$ channels representing 3D position $[x, y, z]$ and velocity $[v_x, v_y, v_z]$), which serves as a crucial prerequisite for permutation-invariant spatial feature extraction.

3 Implementation Details

3.1 Stage 1: mmWave Feature Extraction

SPSE Module (Spatial Point-wise Structure Encoder)

Input:
  radar sequence $X \in \mathbb{R}^{B\times T\times N\times C}$, where B = batch size, T = time frames, N = number of points, C = radar_dim

Algorithm SPSE:

1. Flatten temporal dimension into batch dimension, apply point-wise 1D conv
   $X' \leftarrow$ reshape(X, $[B \cdot T, N, C]$) $\rightarrow$ $[B \cdot T, C, N]$

2. First stage shared point encoder (shared weights across points)
   F_1 = Conv1d$_{C\rightarrow 128}$ $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Conv1d$_{128\rightarrow 256}$ # $F_1 \in \mathbb{R}^{B\cdot T\times 256\times N}$

3. First max pooling (extract frame-level global token)
   $G_1 = \max_{n=1..N}(F_1[:,:,n]) \in \mathbb{R}^{B\cdot T\times 256\times 1}$

4. Inject global token back to each point (global-local concat)
   $\tilde{F}$ = concat(expand(G_1, N), F_1) $\in \mathbb{R}^{B\cdot T\times 512\times N}$

5. Second stage shared point encoder
   F_2 = Conv1d$_{512\rightarrow 512}$ $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Conv1d$_{512\rightarrow d}$ # where d = input_dim (e.g., 64)

6. Second max pooling (final spatial descriptor)
   $Z = \max_{n=1..N}(F_2[:,:,n]) \in \mathbb{R}^{B\cdot T\times d}$

7. Restore the batch and temporal dimensions
   Z_{spatial} = reshape(Z, $[B, T, d]$)

Output:
  $Z_{\text{spatial}} \in \mathbb{R}^{B\times T\times d}$
  Property: permutation-invariant over point order.

Listing 3: SPSE Module

The SPSE module extracts permutation-invariant spatial features using a PointNet-style architecture (Listing 3). Given mmWave radar point cloud input $\mathbf{X} \in \mathbb{R}^{B\times T\times N\times C}$, we first flatten the temporal dimension into the batch dimension ($\mathbf{X'} \in \mathbb{R}^{(B\cdot T)\times C\times N}$) and apply a shared point-wise 1D convolution to project single-point features into $\mathbf{F}_1 \in \mathbb{R}^{(B\cdot T)\times 256\times N}$. To capture global spatial context, a frame-level token $\mathbf{G}_1 = \max_{n=1..N}(\mathbf{F}_1[:,:,n])$ is pooled, replicated, and concatenated with the local features to form a fused representation $\tilde{\mathbf{F}} = [\text{expand}(\mathbf{G}_1, N); \mathbf{F}_1] \in \mathbb{R}^{(B\cdot T)\times 512\times N}$. Finally, a second-stage shared convolution projects $\tilde{\mathbf{F}}$ to the target dimension d, and a symmetric max pooling yields the reshaped spatial descriptor:

$$\mathbf{Z}_{\text{spatial}} = \text{reshape}\left(\max_{n=1..N}(\mathbf{F}_2[:,:,n]), [B, T, d]\right)$$

Sharing convolutional weights across points and utilizing symmetric max pooling ensures that the output feature representation is strictly invariant to input point permutations while preserving essential geometric structures.

MMCL Module (Masked Motion Consistency Learning)

Input:
  radar sequence X $\in \mathbb{R}^{B\times T\times N\times C}$
  mask_ratio r = 0.75

Algorithm MMCL Pretraining:

for each epoch:
    for each batch X:

        1. Point-wise temporal encoding # each frame is first converted into a frame-level feature

```
            Z = shared_MLP_module(X)  # Z: [B, T, D0]

         2. Token embedding
            Z = LinearProjection(Z)
            Z = PositionalEncoding(Z)

         3. Random masking on temporal tokens
            keep_len = int(T * (1 - r))
            Z_keep, mask, ids_restore = RandomMask(Z, ratio=r)
            # masking is applied on the temporal dimension only

         4. Transformer encoder
            Z_latent = PreNormTransformerEncoder(Z_keep)

         5. Transformer decoder
            Z_recon = TransformerDecoder(Z_latent, ids_restore)

         6. Reconstruction loss on masked tokens only
            ℒ_recon = MSE(Z_recon[mask = 1], Z[mask = 1])

         7. Uniformity regularizer
            for each timestep t:
                F_t = encoder features at timestep t
                ℒ_unif,t = 1/B² Σ_{i=1}^{B} Σ_{j≠i}^{B} (cos(f_{t,i}, f_{t,j}))²
            ℒ_unif = 1/T Σ_{t=1}^{T} ℒ_unif,t

         8. Total loss
            ℒ_total = ℒ_recon + λ_unif · ℒ_unif

         9. Backpropagation and update
            ℒ_total.backward()
            optimizer.step()

Output:
  Pretrained MMCL weights for later-stage feature extraction
```

Listing 4: MMCL Module

The MMCL module learns label-free motion features using a temporal masked autoencoding self-supervised strategy [3,5] (Listing 4). Rather than masking raw points directly, a shared MLP first compresses each frame's point cloud into a temporal feature token $\mathbf{Z}$. We apply random masking with a ratio of $r = 0.75$ along this temporal dimension. The retained tokens are processed by a PreNorm Transformer encoder, and a symmetric decoder reconstructs the masked features $\mathbf{Z}_{\text{recon}}$. The reconstruction loss $\mathcal{L}_{\text{recon}}$ computes the MSE exclusively on masked timesteps:

$$\mathcal{L}_{\text{recon}} = \mathbb{E}_{b,t\in\text{masked}} \|\mathbf{Z}_{\text{recon}}[b,t,:] - \mathbf{Z}[b,t,:]\|_2^2 \tag{1}$$

To prevent representation collapse, we incorporate a uniformity regularizer $\mathcal{L}_{\text{unif}}$ that averages the mean square of pairwise cosine similarities of L2-normalized batch features $\mathbf{f}_{t,i}$ across all T timesteps:

$$\mathcal{L}_{\text{unif}} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{B^2}\sum_{i=1}^{B}\sum_{j\neq i}^{B}(\cos(\mathbf{f}_{t,i},\mathbf{f}_{t,j}))^2 \tag{2}$$

The total loss is $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{unif}}\mathcal{L}_{\text{unif}}$. After self-supervised pretraining, the MMCL weights are saved and frozen to serve as a stable temporal feature extractor for subsequent training phases.

STAT Module (Spatio-Temporal Alignment and Transformation)

```
Input:
  radar point cloud X ∈ ℝ^{B×T×N×C}
  ground truth 3D pose Y_gt ∈ ℝ^{B×17×3}

Algorithm STAT:

1. Spatial Feature Extraction (SPSE)
```

```
   Z_spatial = SPSE_Encoder(X)  # [B, T, 64] (default T = 25)

2. Motion Feature Extraction (MMCL - frozen)
   with no_grad():
       Z_motion = MMCL_Encoder(X, mask_ratio=0.75, mode="test")

3. Feature Fusion
   Z_motion_pooled = GlobalAveragePooling(Z_motion)
   Z_fused = ContextInjectionFusion(Z_spatial, Z_motion_pooled)

4. Pose Regression
   Ŷ_coarse, f_stage1 = PoseRegressor(Z_fused)  # pose_coarse: [B, 17, 3]

5. Compute Loss
   L_pose = MSE(pose_coarse, Y_gt)

Output:
  coarse_pose Ŷ_coarse ∈ ℝ^{B×17×3}
  features for next stage
```

Listing 5: STAT Module

The STAT module fuses Stage 1 spatial and motion features to generate initial 3D poses and stage-level feature representations (Listing 5). The process is executed in four steps: (1) SPSE extracts frame-level spatial features $\mathbf{Z}_{\text{spatial}} \in \mathbb{R}^{B\times T\times 64}$; (2) the frozen, pre-trained MMCL encoder extracts motion features $\mathbf{Z}_{\text{motion}}$ in test mode with gradient tracking disabled; (3) a Context Injection Fusion layer constructs a unified spatio-temporal representation $\mathbf{Z}_{\text{fused}} = \text{ContextInjectionFusion}(\mathbf{Z}_{\text{spatial}}, \mathbf{Z}_{\text{motion\_pooled}})$, where $\mathbf{Z}_{\text{motion\_pooled}}$ is globally average-pooled; and (4) a pose regression head predicts the coarse pose $\hat{\mathbf{Y}}_{\text{coarse}} \in \mathbb{R}^{B\times 17\times 3}$ and Stage 2 features $\mathbf{f}_{\text{stage1}}$:

$$\hat{\mathbf{Y}}_{\text{coarse}}, \mathbf{f}_{\text{stage1}} = \text{PoseRegressor}(\mathbf{Z}_{\text{fused}})$$

The training objective is to minimize the pose Mean Squared Error (MSE) loss: $\mathcal{L}_{\text{pose}} = \text{MSE}(\hat{\mathbf{Y}}_{\text{coarse}}, \mathbf{Y}_{\text{gt}})$. Both preliminary predictions and intermediate features are cached for the subsequent Stage 2 refinement.

3.2 Stage 2: Diffusion-Based Pose Refinement

TJA Module (Temporal Joint Alignment)

```
Input:
  feature history feat_hist ∈ ℝ^{B×T×17×F}, where T = past_frames (e.g., 8), F = feature dimension

Algorithm TJA:

1. Feature History Reorganization
   feat_hist = feat_hist.reshape(B, 17, T, F)
   feat_hist = feat_hist.permute(0, 2, 1, 3)  # [B, T, 17, F]

2. Temporal Transformer Encoding
   past_emb = TCATransformer(feat_hist)  # [B, 17, H]

   Inside TCATransformer:
   - input projection: [B, T, 17, F] -> [B, T, 17, H]
   - add temporal positional embedding
   - reshape to [B*17, T, H] for per-joint temporal modeling
   - temporal Transformer encoding
   - temporal pooling (CNN / attention / mean; default: CNN)
   - output: [B, 17, H]

3. Contrastive Learning (auxiliary)
   feat_hist_a = augment_feat_hist(feat_hist, drop_p=0.125, noise_std=0.01)
   feat_hist_b = augment_feat_hist(feat_hist, drop_p=0.125, noise_std=0.01)

   past_emb_a = TCATransformer(feat_hist_a)  # [B, 17, H]
   past_emb_b = TCATransformer(feat_hist_b)  # [B, 17, H]

   z_a = ContrastiveHead(past_emb_a.reshape(B*17, H))  # [B*17, d]
   z_b = ContrastiveHead(past_emb_b.reshape(B*17, H))  # [B*17, d]

   loss_temp = InfoNCE(z_a, z_b, temp=0.07)
```

```
Output:
  past_emb ∈ ℝ^{B×17×H}
  loss_temp (auxiliary temporal consistency loss)
```

Listing 6: TJA Module

The TJA module encodes joint-level historical features into a reliable temporal condition (Listing 6). Given $\mathbf{f}_{\text{hist}} \in \mathbb{R}^{B\times T\times 17\times F}$, the TCATransformer applies linear projection and temporal positional embeddings, reshaping the tensor to $[B \times 17, T, H]$ to perform temporal self-attention independently for each joint. To compress the sequence, temporal pooling (defaulting to CNN pooling over attention or mean pooling) is applied, outputting $\mathbf{z}_{\text{past}} \in \mathbb{R}^{B\times 17\times H}$.To enforce temporal consistency, TJA introduces an auxiliary contrastive task. We apply frame dropout and Gaussian noise to $\mathbf{f}_{\text{hist}}$ to create two augmented views, which are processed into unit-normalized embeddings $\mathbf{z}_a, \mathbf{z}_b$ via the TCATransformer and a ContrastiveHead. For a batch of $K = B \times 17$ joint-level feature pairs $\{(\mathbf{z}_{a,i}, \mathbf{z}_{b,i})\}_{i=1}^{K}$ (where B is the sequence batch size and 17 is the number of skeletal joints), the temporal contrastive loss $\mathcal{L}_{\text{temp}}$ is formulated as:

$$\mathcal{L}_{\text{temp}} = -\frac{1}{K}\sum_{i=1}^{K}\log\frac{\exp(\mathbf{z}_{a,i}\cdot\mathbf{z}_{b,i}/\tau)}{\sum_{j=1}^{K}\exp(\mathbf{z}_{a,i}\cdot\mathbf{z}_{b,j}/\tau)} \tag{3}$$

where $\tau = 0.07$. This cosine-similarity-based loss pulls representations of the same joint (positive pairs) closer while pushing apart different joints or sequences (negative samples). Finally, $\mathcal{L}_{\text{temp}}$ is incorporated into the Phase 3 training objective with a scaling weight of 5.

SJA Module (Structural Joint Alignment)

```
Input:
  x: coarse pose history with current pose ∈ ℝ^{B×17×3(past_frames+1)}
  cemd: global joint feature ∈ ℝ^{B×17×feature_size}
  limb_len: optional ground-truth limb length ∈ ℝ^{B×16}
  limb_flag ∈ {1, 2}

Hyperparameters:
  num_joints = 17, num_limbs = 16
  edge_emd = emd_dim / 16
  incidence_S ∈ {0, 1}^{16×17}
  deg = sum over edges of incidence_S, shape [17]

Algorithm SJA:

1. Limb Length Regression
   if limb_flag == 1:
       limb_len_pred = MLP_limb(cemd.view(B, 17 * feature_size))
   elif limb_flag == 2:
       limb_len_pred = MLP_limb(x.reshape(B, (past_frames + 1) * 17 * 3))

   if limb_len is None:
       limb_len = limb_len_pred
   # limb_len ∈ ℝ^{B×16}

2. Limb Embedding Encoding
   limb_emb = MLP_1(limb_len)          # [B, emd_dim]
   limb_emb = Swish(limb_emb)
   limb_emb = MLP_2(limb_emb)          # [B, emd_dim]

   limb_edge = reshape(limb_emb, [B, 16, edge_emd])
   limb_edge_H = EdgeMLP(limb_edge)    # [B, 16, H]

3. Incidence Matrix Projection (Edge → Node)
   limb_node_H = einsum('beh,ej->bjh', limb_edge_H, incidence_S)

   # Normalization by degree
   limb_node_H = limb_node_H / deg[None, :, None]

4. Structural Condition Injection
```

```
   cond_ctx = concat(global_emb, local_emb, past_emb, limb_node_H)
   # limb_node_H constitutes one branch of the unified diffusion condition

5. Auxiliary Supervision (Training Phase)
   loss_limb = L1_Loss(limb_len_pred, limb_len_gt)

Output:
  limb_node_H ∈ ℝ^{B×17×H}   (structural condition)
  cond_ctx ∈ ℝ^{B×L_c×H}  (unified condition context)
  loss_limb (auxiliary limb length loss)
```

Listing 7: SJA Module

The SJA module provides a learned structural condition to model skeletal properties within the diffusion denoiser (Listing 7). It regresses invariant limb lengths $\hat{\boldsymbol{\ell}} = \mathrm{MLP}_{\mathrm{limb}}(\mathbf{c}_{\mathrm{emd}} \text{ or } \mathbf{x}) \in \mathbb{R}^{B\times 16}$ from either global features $\mathbf{c}_{\mathrm{emd}}$ or historical coarse poses $\mathbf{x}$. These predictions are encoded into edge tokens $\mathbf{h}_{\text{limb, edge}} \in \mathbb{R}^{B\times 16\times H}$ via a Swish-MLP, and mapped to joint-level nodes using an incidence matrix $\mathbf{S} \in \{0,1\}^{16\times 17}$ and joint degree vector $\mathbf{d} \in \mathbb{R}^{17}$:

$$\mathbf{h}_{\text{limb, node}} = \frac{\mathrm{einsum}('\mathrm{beh,ej} \to \mathrm{bjh}', \mathbf{h}_{\text{limb, edge}}, \mathbf{S})}{\mathbf{d}[\mathrm{None}, :, \mathrm{None}]}$$

The joint-aligned tokens $\mathbf{h}_{\text{limb, node}}$ are concatenated with other embeddings to form the unified context $\mathbf{c}_{\mathrm{ctx}} \in \mathbb{R}^{B\times L_c\times H}$, guiding denoising via cross-attention. During Phase 3 training, the structural prior is supervised via an auxiliary L_1 loss:

$$\mathcal{L}_{\mathrm{limb}} = \frac{1}{B}\sum_{b=1}^{B}\sum_{k=1}^{16}\left|\hat{\ell}_{b,k} - \ell^{\mathrm{gt}}_{b,k}\right|. \tag{4}$$

This loss is scaled by 15 in the total objective, anchoring structural plausibility without imposing rigid geometric constraints during inference.

GMCF Module (Gated Multi-Condition Fusion)

```
Input:
  global_emb ∈ ℝ^{B×17×H}        (from global branch)
  local_emb ∈ ℝ^{B×17×H}         (from local branch)
  past_emb ∈ ℝ^{B×17×H}          (from temporal branch)
  limb_node_H ∈ ℝ^{B×17×H}       (from limb branch)

Learnable Parameters:
  cond_gate ∈ ℝ^4                # learned gates for 4 branches
  cond_type_emb ∈ ℝ^{4×1×1×H}    # per-branch type embeddings
  cond_adapt_g, cond_adapt_l, cond_adapt_t, cond_adapt_m

Algorithm GMCF:

1. Branch Weighting and Adaptation
   parts = []

   if global_flag == 1 and global_emb is not None:
       g = cond_adapt_g(cond_gate[0] * global_emb) + cond_type_emb[0]
       parts.append(g)

   if local_flag == 1 and local_emb is not None:
       l = cond_adapt_l(cond_gate[1] * local_emb) + cond_type_emb[1]
       parts.append(l)

   if temp_flag == 1 and past_emb is not None:
       t = cond_adapt_t(cond_gate[2] * past_emb) + cond_type_emb[2]
       parts.append(t)

   if limb_flag in (1, 2) and limb_node_H is not None:
       m = cond_adapt_m(cond_gate[3] * limb_node_H) + cond_type_emb[3]
       parts.append(m)

2. Concatenation into Context Tensor
   if len(parts) == 0:
       cond_ctx = None
```

```
    else:
        cond_ctx = torch.cat(parts, dim=1)  # [B, L_c, H], where L_c = 17 * (number of enabled branches)

Output:
  cond_ctx ∈ ℝ^{B×L_c×H}   (concatenated condition tokens)
```

Listing 8: GMCF Module

The GMCF module fuses the available condition streams into a unified context tensor (Listing 8). To achieve this, the module utilizes a gating mechanism and lightweight adapters to align and weight each active branch. Specifically, the input of each enabled branch $\mathbf{z}_i \in \mathbb{R}^{B\times 17\times H}$ ($i \in \{0,1,2,3\}$ corresponding to the global, local, temporal, and limb branches, respectively) is first scaled by a learnable scalar gate $\alpha_i \in \mathbb{R}$. It is then passed through a lightweight linear adapter, which includes layer normalization, to balance the feature scales:

$$\mathbf{z}'_i = \mathrm{Adapter}_i(\alpha_i \cdot \mathbf{z}_i)$$

Subsequently, a branch-specific type embedding $\mathbf{e}_i \in \mathbb{R}^{1\times 1\times H}$ is added to preserve branch identity:

$$\tilde{\mathbf{z}}_i = \mathbf{z}'_i + \mathbf{e}_i$$

Finally, the adapted branch tensors are concatenated along the sequence dimension to form the final condition context tensor $\mathbf{c}_{\mathrm{ctx}}$:

$$\mathbf{c}_{\mathrm{ctx}} = \mathrm{concat}(\{\tilde{\mathbf{z}}_i \mid i \in \mathrm{enabled}\}) \in \mathbb{R}^{B\times L_c\times H}$$

Here, the sequence length L_c is not fixed; rather, $L_c = 17 \times |\mathrm{enabled\ branches}|$. This dynamic context length ensures that the model can flexibly enable or disable individual branches during experiments, facilitating comprehensive ablation studies.

CCDM Module (Cross-Attention Conditional Diffusion Model)

```
Input:
  x_all_list ∈ ℝ^{B×17×3(past_frames+1)}   (history coarse poses + current noisy pose)
  t ∈ ℝ^B                                  (diffusion timesteps)
  cemd ∈ ℝ^{B×17×F}                        (global feature token)
  radar ∈ ℝ^{B×T×N×C}                      (radar input)

Optional Input:
  cond_ctx ∈ ℝ^{B×L_c×H}   (unified condition from GMCF)

Hyperparameters:
  hid_dim = 96  # Hidden dimension
  emd_dim = 4 * hid_dim
  num_layers = 5  # Number of graph layers
  n_head = 4  # Number of multi-head attention heads
  num_diffusion_timesteps = 51  # Number of diffusion steps
  beta_schedule = "linear"  # Beta schedule

Algorithm CCDM Denoising:

1. Timestep Embedding
   temb = get_timestep_embedding(t, hid_dim)  # sinusoidal
   temb = MLP_1(temb)
   temb = Swish(temb)
   temb = MLP_2(temb)  # [B, emd_dim]

2. Initial Node Embedding
   out = GCNConv_input(x_all_list[:, :, -3:], adj)  # [B, 17, H]

3. Layer-wise Denoising (Graph + Cross-Attention)
   for layer_idx in range(num_layers):
       # Graphformer Self-Attention
       out = GraphAttentionLayer[layer_idx](out, src_mask)
       # Conditional Cross-Attention
       if cond_ctx is not None:
           out = CrossAttnBlock[layer_idx](out, cond_ctx)
```

```
      # Time-modulated Residual Graph Convolution
      out = ResidualChebGC[layer_idx](out, temb, condition=None)  # [B, 17, H]

4. Output Projection
   out = GCNConv_output(out, adj)    # [B, 17, 3 × (past_frames + 1)]
   predicted_noise = out[:, :, -3:]  # [B, 17, 3]

5. Auxiliary Loss (used during Phase 3 training)
   loss_limb = |limb_len_pred - limb_len_gt|    # added in Phase 3
   loss_temp = InfoNCE(z_a, z_b)                # returned by temporal branch

Output:
  predicted_noise ∈ ℝ^{B×17×3}
  loss_limb
  loss_temp
```

Listing 9: CCDM Module

The CCDM acts as the core denoiser of Stage 2, integrating graph convolutions, self-attention, and conditional cross-attention (Listing 9). In the implementation, the denoising network receives the concatenated pose history and current noisy pose as $\mathbf{x}_{\text{all\_list}} \in \mathbb{R}^{B\times 17\times 3(\text{past\_frames}+1)}$. The forward denoising process is executed through the following stages:

1. **Timestep Embedding:** The diffusion timestep t is converted into a continuous vector via sinusoidal positional embeddings, followed by a two-layer MLP with Swish activation to produce the temporal condition vector $\mathbf{t}_{\text{emb}} \in \mathbb{R}^{B\times D_{\text{emb}}}$, where $D_{\text{emb}} = 4H$.
2. **Initial Node Embedding:** Rather than processing the entire sequence, only the last three channels of $\mathbf{x}_{\text{all\_list}}$ are sliced to isolate the current noisy pose. This slice is projected into the hidden space through a Chebyshev graph convolutional layer [4], resulting in $\mathbf{o} \in \mathbb{R}^{B\times 17\times H}$.
3. **Multi-layer Denoising:** The stacked denoising layers total $N_{\text{layer}} = 5$. Each layer consists of three submodules: (1) GraphFormer self-attention layer [8], which computes self-attention on the joints over the skeletal graph; (2) CrossAttnBlock, which leverages the unified condition context $\mathbf{c}_{\text{ctx}}$ as keys/-values and the pose nodes as queries to fuse conditions via multi-head cross-attention; (3) Residual Chebyshev graph convolutional layer (ResChebGC), which incorporates timestep embedding modulation to provide diffusion step information.
4. **Output Projection and Noise Prediction:** The final node embeddings pass through an output projection layer (a Chebyshev graph convolution yielding $3\times(\text{past\_frames}+1)$ channels). The last three channels correspond directly to the predicted noise for the current timestep, $\hat{\boldsymbol{\epsilon}} \in \mathbb{R}^{B\times 17\times 3}$.
5. **Auxiliary Losses:** During training, the module additionally computes limb length loss and temporal contrastive loss to enforce physical plausibility and temporal consistency of the predictions.

3.3 Three-Phase Training Strategy

```
Phase 1 (Self-supervised MMCL Pretraining)
|- Input: Radar sequences (no pose labels)
|- Trainable: MMCL encoder/decoder
|- Frozen: None
|- Loss: L_recon + λ_unif · L_unif
+- Output: Pretrained MMCL encoder

Phase 2 (Backbone / Coarse Pose Learning)
|- Input: Radar sequences + GT 3D poses
|- Trainable: SPSE, STAT, PoseRegressor
```

```
|- Frozen: MMCL encoder
|- Loss: ℒ_2(pose_pred, gt_pose)
+- Output: Coarse pose estimator + precomputed features

Phase 3 (Diffusion Refinement)
|- Input: Coarse poses + Radar + Conditions
|- Trainable: Diffusion denoiser (GCNdiff)
|- Frozen: Feature extractors + Coarse pose regressor
|- Loss: ℒ_noise + λ_limb · ℒ_limb + λ_temp · ℒ_temp
+- Output: Fine-grained pose predictions
```

Listing 10: Three-Phase Training Pipeline

Phase 1: MMCL Self-supervised Pretraining. Phase 1 pretrains the MMCL encoder self-supervisedly on raw mmWave radar sequences without pose labels. We train for 30 epochs using the AdamW optimizer (lr $= 1 \times 10^{-5}$, weight decay 1×10^{-4}). The objective minimizes the sum of mask reconstruction Eq. (1) and uniformity regularization Eq. (2) losses:

$$\mathcal{L}_{\text{phase1}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{unif}} \mathcal{L}_{\text{unif}} \tag{5}$$

where $\lambda_{\text{unif}} = 0.1$. The model with the lowest validation loss is saved. Post-training, the MMCL encoder is frozen for Phase 2 feature extraction.

Phase 2: Coarse Pose Learning. Phase 2 trains SPSE, STAT, and the pose regression head for coarse 3D pose estimation, keeping the pre-trained MMCL encoder frozen. These modules are optimized via Adam (lr $= 1 \times 10^{-3}$, weight decay 1×10^{-4}) to minimize the MSE loss against the ground truth:

$$\mathcal{L}_{\text{phase2}} = \|\hat{\mathbf{Y}}_{\text{coarse}} - \mathbf{Y}_{\text{gt}}\|_2^2 \tag{6}$$

To accelerate subsequent diffusion training, the optimal model (evaluated via MPJPE) is utilized to precompute and cache intermediate representations—including coarse poses and global features—for the entire dataset.

Phase 3: Diffusion Model Fine-tuning. Phase 3 fine-tunes the conditional diffusion model (GCNdiff) while freezing all prior feature extraction and coarse pose modules.

Model Initialization. The diffusion schedule employs a linear beta schedule ($\beta_{\min} = 10^{-4}$, $\beta_{\max} = 0.02$, $T = 51$), with the cumulative product defined as $\bar{\alpha}_t = \prod_{s=0}^{t}(1 - \beta_s)$.

Forward Process. For each batch, a timestep $t \sim \mathcal{U}[0, T-1]$ is randomly sampled, and Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ is added to the ground-truth pose $\mathbf{Y}_{\text{gt}}$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{Y}_{\text{gt}} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$$

Loss Function. The model predicts the injected noise $\hat{\boldsymbol{\epsilon}}$, minimizing the MSE alongside auxiliary limb length consistency Eq. (4) and temporal contrastive Eq. (3) losses:

$$\mathcal{L}_{\text{phase3}} = \|\hat{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}\|_2^2 + \lambda_{\text{limb}}\mathcal{L}_{\text{limb}} + \lambda_{\text{temp}}\mathcal{L}_{\text{temp}} \tag{7}$$

where $\lambda_{\text{limb}} = 15$ and $\lambda_{\text{temp}} = 5$.

Optimization and Inference. The denoiser is trained using Adam (lr = 1×10^{-3}) with gradients clipped to a maximum norm of 1.0. The checkpoint that produces the best test metrics (MPJPE, PA-MPJPE) is saved. During inference, fine-grained 3D poses are generated via multi-step DDIM sampling.

4 Evaluation and Inference

4.1 Metrics

We evaluate 3D pose estimation using two metrics: Mean Per-Joint Position Error (MPJPE) and Procrustes-Aligned MPJPE (PA-MPJPE), following [2]. Given a predicted pose $\hat{\mathbf{P}}$ and its ground truth $\mathbf{P}$ (with $J = 17$ joints), both are centered by translating the pelvis to the origin. MPJPE is then calculated as the average Euclidean distance across all joints:

$$\text{MPJPE} = \frac{1}{J} \sum_{j=1}^{J} \left\| \hat{\mathbf{P}}_j - \mathbf{P}j \right\|_2 . \tag{8}$$

PA-MPJPE further applies a rigid Procrustes alignment operator $\mathcal{A}(\cdot)$ to the prediction, solving for the optimal scale, rotation, and translation before computing the error, thereby measuring structural accuracy independent of global rigid transformations:

$$\text{PA-MPJPE} = \frac{1}{J} \sum_{j=1}^{J} \left\| \mathcal{A}(\hat{\mathbf{P}})_j - \mathbf{P}_j \right\|_2 . \tag{9}$$

Both metrics are averaged over the entire test set and reported in millimeters (mm). Where available, errors are also accumulated per action category for fine-grained analysis.

4.2 Comparison with Baseline mmWave Pose Models

All baseline comparisons are evaluated under identical dataset protocols, split definitions (e.g., cross-scene, cross-subject), and preprocessing pipelines to ensure fair benchmarking.

For the PointTransformer [7] baseline, we adopt the point-transformation backbone as the feature extractor, coupled with the same downstream pose regression head and test procedures. For the mmDiff [2] baseline, we reproduce its two-stage pipeline by initializing the model with protocol-specific checkpoints and configuration files provided in the codebase. This unified evaluation framework guarantees that baseline results are reproduced under the exact same data handling and metric computation as our proposed method.

In our quantitative comparisons, we report results directly reproduced by our evaluation script. Performance metrics cited directly from original papers or official implementations are included only when exact checkpoints or training provenances are unavailable in our repository.

4.3 Ablation Settings

Temporal Window Ablation. To analyze the impact of temporal context length, we evaluate the model across varying sequence lengths $T \in \{15, 25, 35\}$, with $T = 25$ serving as the default setting. This adjustment is implemented at the data construction stage by modifying the sliding-window size. Consequently, the temporal encoders are re-instantiated and trained with the corresponding temporal lengths. Thus, the sequence parameter T is deeply integrated into both the data pipeline and the model architecture, rather than acting merely as a runtime inference flag.

Module Ablations. To rigorously validate the contribution of individual components, we conduct ablation studies by disabling or replacing specific modules while strictly maintaining identical dataset protocols, data splits, and evaluation scripts. The architectural ablations are implemented as follows:

- **w/o Diff.:** Disables the final diffusion refinement stage entirely. The model directly outputs and evaluates the coarse pose generated from Phase 2.
- **w/o MMCL:** Omits the Phase 1 masked self-supervised pretraining. Instead, the motion encoder is randomly initialized and trained from scratch during the Phase 2 coarse pose learning stage.
- **w/o SPSE:** Removes the Spatial Point-wise Structure Encoder branch, relying solely on alternative feature streams.
- **w/o STAT:** Bypasses the Spatio-Temporal Alignment and Transformation fusion module, dropping the explicitly aligned feature aggregation prior to pose regression.

All architectural variants are retrained under their respective modified configurations to ensure fair comparison.

4.4 Inference Procedure and Runtime

At inference, the pipeline operates in two modes. In standard evaluation, Stage 1 spatial and temporal outputs are pre-computed and cached. The Stage 2 diffusion module directly consumes these cached features for refinement. For online end-to-end inference, raw mmWave inputs are first preprocessed during data-loading via sliding-window aggregation and point normalization, and the Stage 1 backbone extracts coarse 3D poses and context features on the fly.

Using the coarse pose and temporal context, the Stage 2 diffusion module performs reverse sampling via the `generalized_steps` routine. This sampler follows a DDIM-style deterministic update rule with $\eta = 0$ and is configured for 25 steps (optimized from 51 training steps to balance accuracy and latency). While the architecture supports multiple hypotheses, our default evaluation sets `test_times` to 1 to maximize throughput.

Computational overhead is profiled solely on the diffusion submodel, excluding data loading and Stage 1 backbone latency. We measure the parameter

count via `fvcore`'s `parameter_count_table`, compute theoretical FLOPs using `FlopCountAnalysis`, and record latency/FPS via CUDA events on the target GPU, averaged over 200 iterations after a 50-iteration warm-up phase.

5 Conclusion

This companion paper provides comprehensive implementation and reproducibility notes for our ICPR 2026 spatio-temporal mmWave 3D human pose estimation framework. To facilitate seamless reproduction and extension, we have detailed: (1) the dataset preprocessing pipeline, highlighting sliding-window aggregation and point normalization; (2) the structural configurations of our two-stage core modules (SPSE, MMCL, STAT, TJA, SJA, GMCF, and CCDM); and (3) the progressive three-phase training strategy. Finally, we clarified the evaluation metrics, ablation setups, and inference procedures. These practical details are intended to resolve implementation ambiguities and promote future research in mmWave radar-based human sensing.

References

1. Cao, K.M., Lee, M.H., Hsu, W.C., Wu, K.R., Lin, H.D., Xie, R.D., Chen, B.Y., Tseng, Y.C.: Robust 3D Human Pose Estimation from mmWave Radar via Spatio-Temporal Representation Learning. In: International Conference on Pattern Recognition. Springer (2026)
2. Fan, J., Yang, J., Xu, Y., Xie, L.: Diffusion Model is a Good Pose Estimator from 3D RF-Vision. In: Proc. European Conference on Computer Vision (ECCV). pp. 1–18 (2024)
3. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked Autoencoders Are Scalable Vision Learners. In: Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 16000–16009 (2022)
4. He, M., Wei, Z., Wen, J.R.: Convolutional neural networks on graphs with chebyshev approximation, revisited. Advances in neural information processing systems **35**, 7264–7276 (2022)
5. Li, Z., Rao, Z., Pan, L., Wang, P., Xu, Z.: TI-MAE: Self-Supervised Masked Time Series Autoencoders. arXiv preprint arXiv:2301.08871 (2023)
6. Yang, J., Huang, H., Zhou, Y., Chen, X., Xu, Y., Yuan, S., Zou, H., Lu, C.X., Xie, L.: MM-FI: Multi-Modal Non-Intrusive 4D Human Dataset for Versatile Wireless Sensing. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 36, pp. 18756–18768 (2023)
7. Zhao, H., Jiang, L., Jia, J., Torr, P.H.S., Koltun, V.: Point Transformer. In: Proc. IEEE/CVF International Conference on Computer Vision (ICCV). pp. 16259–16268 (2021)
8. Zhao, W., Wang, W., Tian, Y.: Graformer: Graph-oriented transformer for 3d pose estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 20438–20447 (2022)