

---

# Supplementary Material for: Geo-Sign: Hyperbolic Contrastive Regularisation for Geometrically Aware Sign Language Translation

---

Anonymous Author(s)

Affiliation

Address

email

---

## Supplementary Material Contents

2	<b>Contents</b>	
3	<b>A Introduction</b>	<b>2</b>
4	<b>B Hyperbolic Geometry Preliminaries: A Brief Refresher</b>	<b>2</b>
5	<b>C Methodology Details</b>	<b>3</b>
6	C.1 Pose Extraction and ST-GCN Architecture . . . . .	3
7	C.2 Hyperbolic Alignment Strategies . . . . .	4
8	C.2.1 Pooled Method (Global Semantic Alignment) . . . . .	4
9	C.2.2 Token Method (Fine-Grained Part-Text Alignment) . . . . .	5
10	C.2.3 Intuition Behind the Token Method . . . . .	5
11	<b>D Mathematical Foundations (Summary)</b>	<b>6</b>
12	D.1 Fréchet Mean in the Poincaré Ball . . . . .	6
13	D.2 Gradient of the Hyperbolic Distance . . . . .	7
14	<b>E Learnable Model Parameters</b>	<b>7</b>
15	E.1 Discussion on Learnable Curvature $c$ . . . . .	7
16	E.2 Discussion on Loss Blending Factor $\alpha$ . . . . .	8
17	<b>F Experimental Setup, Analysis, and Qualitative Results</b>	<b>8</b>
18	F.1 Computational Considerations and Profiler Analysis . . . . .	8
19	F.1.1 Profiler Summary and Comparative Analysis . . . . .	9
20	F.1.2 Discussion on Data Efficiency . . . . .	11
21	F.2 Further Technical Implementation Details . . . . .	12
22	F.3 Qualitative Results . . . . .	12
23	<b>G Code Listings</b>	<b>18</b>
24	<b>References</b>	<b>33</b>

---

## 25 A Introduction

26 In this appendix, we provide comprehensive supplementary details to accompany our main paper. The  
 27 goal is to offer an in-depth understanding of our methodology, experimental setup, and the underlying  
 28 geometric principles, thereby ensuring clarity and facilitating the reproducibility of our work.

29 This document elaborates on:

- 30 • The specifics of pose feature extraction and the Spatio-Temporal Graph Convolutional  
 31 Network (ST-GCN) architecture employed (Appendix C.1).
- 32 • Detailed explanations and implementations of our proposed hyperbolic alignment strategies,  
 33 including the Pooled Method and the Token Method (Appendix C.2).
- 34 • Further mathematical derivations and discussions pertinent to hyperbolic operations, such as  
 35 Fréchet mean computation and contrastive loss gradients (Appendix D).
- 36 • Elaboration on the learnable parameters within our model, particularly the manifold curva-  
 37 ture  $c$  and the loss blending factor  $\alpha$  (Appendix E).
- 38 • A discussion of computational considerations, experimental setup, and qualitative results  
 39 (Appendix F).
- 40 • Key code snippets for essential components of Geo-Sign are provided in Appendix G to aid  
 41 in understanding and replication.

## 42 B Hyperbolic Geometry Preliminaries: A Brief Refresher

43 To ensure this supplementary material is self-contained and accessible, this section briefly recaps key  
 44 concepts from hyperbolic geometry, as introduced in Section 3.1 (“Hyperbolic Geometry Essentials”)  
 45 of the main paper.

46 We operate within the  $d_{\text{hyp}}$ -dimensional Poincaré ball model, denoted  $\mathbb{B}_c^{d_{\text{hyp}}} = \{\mathbf{x} \in \mathbb{R}^{d_{\text{hyp}}} : \|\mathbf{x}\|_2 <$   
 47  $1/\sqrt{c}\}$ . This space is characterised by a constant negative curvature  $\kappa = -c$ , where  $c > 0$  is a  
 48 learnable parameter representing the magnitude of the curvature.

49 The Poincaré ball model is chosen for its conformal nature, where angles are preserved locally, and  
 50 its intuitive representation of hyperbolic space within a Euclidean unit ball (scaled by  $1/\sqrt{c}$ ). Key  
 51 operations include:

- 52 • **Geodesic Distance**  $d_{\mathbb{B}_c}(\mathbf{u}, \mathbf{v})$ : This is the shortest path between two points  $\mathbf{u}, \mathbf{v}$  within the  
 53 curved space of the Poincaré ball. It is formally defined in Eq. (1) of the main paper. Unlike  
 54 Euclidean distance, it expands significantly as points approach the boundary of the ball.
- 55 • **Möbius Addition**  $\mathbf{u} \oplus_c \mathbf{v}$ : This operation is the hyperbolic analogue of vector addition in  
 56 Euclidean space, defined in Eq. (2) of the main paper (consistent with formulations in, e.g.,  
 57 (5)). It is essential for defining translations and other transformations in hyperbolic space  
 58 while respecting its geometry.
- 59 • **Exponential Map**  $\exp_{\mathbf{x}}^c(\mathbf{v})$ : This map takes a tangent vector  $\mathbf{v}$  residing in the tangent space  
 60  $\mathcal{T}_{\mathbf{x}}\mathbb{B}_c^{d_{\text{hyp}}}$  at a point  $\mathbf{x}$  on the manifold and maps it to another point on the manifold along a  
 61 geodesic. The map from the origin,  $\exp_0^c(\cdot)$  (Eq. (3), main paper), is particularly important  
 62 as it projects Euclidean feature vectors (which can be considered as residing in  $\mathcal{T}_0\mathbb{B}_c^{d_{\text{hyp}}}$ ) into  
 63 the Poincaré ball.
- 64 • **Logarithmic Map**  $\log_{\mathbf{x}}^c(\mathbf{y})$ : This is the inverse of the exponential map. It takes two points  
 65  $\mathbf{x}, \mathbf{y}$  on the manifold and returns the tangent vector at  $\mathbf{x}$  that points along the geodesic  
 66 towards  $\mathbf{y}$ .
- 67 • **Möbius Transformations**: These are isometries (distance-preserving transformations) of  
 68 hyperbolic space. In our work, we use learnable Möbius transformations, such as Möbius  
 69 matrix-vector products ( $\mathbf{M} \otimes_c \mathbf{v} = \exp_0^c(\mathbf{M} \log_0^c(\mathbf{v}))$ ) and Möbius bias additions, to  
 70 implement affine-like transformations within our hyperbolic attention mechanism.

71 These tools allow us to define neural network operations directly within hyperbolic space. As with all  
 72 hyperbolic operations in the paper, we utilise the geoopt library (9) in Pytorch.

## 73 C Methodology Details

### 74 C.1 Pose Extraction and ST-GCN Architecture Details

75 Our Geo-Sign framework utilizes skeletal pose data as input. This section details the extraction  
76 process and the architecture of the Spatio-Temporal Graph Convolutional Networks (ST-GCNs) used  
77 to encode this data.

78 **Pose Data Source and Preprocessing:** We use the 2D skeletal keypoints provided by the UniSign (10)  
79 framework, which were originally extracted using RTMPose-X (7) based on the COCO-WholeBody  
80 keypoint definition (8). The keypoints are organised into four distinct anatomical groups for targeted  
81 processing:

- 82 • **Body:** Includes 9 joints (COCO indices 1, 4–11).
- 83 • **Left Hand:** Includes 21 joints (COCO indices 92–112).
- 84 • **Right Hand:** Includes 21 joints (COCO indices 113–133).
- 85 • **Face:** Includes 16 keypoints from the facial region (COCO indices 24, 26, 28, 30, 32, 34,  
86 36, 38, 40, 54, 84–91).

87 For normalization, specific anchor joints are used for hand and face parts: joint 92 (left wrist) for the  
88 left hand, joint 113 (right wrist) for the right hand, and joint 54 (a central face point) for the face. The  
89 body part features are not anchor-normalised in this scheme to preserve global torso positioning.

90 **ST-GCN Architecture:** Each anatomical group is processed by a dedicated ST-GCN stream, fol-  
91 lowing the methodology of Yan et al. (17). The ST-GCN is adept at learning representations from  
92 skeletal data by explicitly modeling spatial joint relationships and temporal motion dynamics.

93 The core of the ST-GCN involves:

- 94 1. **Graph Definition:** The skeletal structure for each part is defined as a graph, where joints  
95 are nodes and natural bone connections are edges. The Graph class, detailed in Listing 1  
96 (Appendix G), handles the construction of these graphs and their corresponding adjacency  
97 matrices.
- 98 2. **Initial Projection:** Input keypoint coordinates are first linearly projected to a higher-  
99 dimensional feature space using a linear layer (referred to as `proj_linear` in our codebase).
- 100 3. **ST-GCN Blocks:** A sequence of ST-GCN blocks processes these features. Each block (see  
101 `STGCN_block` in Listing 2, Appendix G) consists of:
  - 102 • A **Spatial Graph Convolution (SGC)** layer, which aggregates information from  
103 neighboring joints. The operation for a node (joint)  $v_i$  at layer  $(l)$  can be expressed  
104 generally as:

$$\mathbf{f}_{\text{out}}(v_i)^{(l)} = \sum_{k=1}^K \left( \sigma \left( \mathbf{A}_k \mathbf{X}^{(l)} \mathbf{W}_k^{(l)} \right) \right)_i, \quad (1)$$

105 where  $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times C_{in}}$  is the matrix of input features for  $N$  nodes with  $C_{in}$  channels,  
106  $\mathbf{W}_k^{(l)} \in \mathbb{R}^{C_{in} \times C_{out}}$  are learnable weight matrices for the  $k$ -th kernel transforming  
107 node features to  $C_{out}$  channels.  $\mathbf{A}_k \in \mathbb{R}^{N \times N}$  is the adjacency matrix for the  $k$ -th  
108 spatial kernel, defining the neighborhood aggregation based on chosen strategies (we  
109 use the spatial configuration partitioning as in the original ST-GCN paper (17)).  $\sigma$  is  
110 an activation function (ReLU in our case), and  $(\cdot)_i$  denotes selection of the  $i$ -th row  
111 (features for node  $v_i$ ). The precise implementation involving tensor reshaping and  
112 `einsum` for efficient aggregation over multiple adjacency kernels is detailed in the  
113 `GCN_unit` code in Listing 2.

- 114 • A **Temporal Convolutional Network (TCN)** layer, which applies 1D convolutions  
115 across the time dimension to model motion patterns.

116 4. **Residual Connections:** To allow richer feature interaction, residual connections are intro-  
117 duced from the body stream’s ST-GCN output to the hand and face streams before their final  
118 temporal fusion layers. This allows global body posture context to inform the interpretation  
119 of fine-grained hand and face movements. Details are in Listing 3 (Appendix G). This

design choice treats body features as fixed contextual input for the parts during each forward pass, isolating the body feature extractor from direct updates via part-specific losses.

The output of each part-specific ST-GCN stream is a feature map  $\mathbf{Z}_p \in \mathbb{R}^{T \times d'_{\text{gc\_out}}}$ , where  $T$  is the sequence length and  $d'_{\text{gc\_out}}$  is the GCN output feature dimension. For the hyperbolic regularization branch, these  $\mathbf{Z}_p$  are temporally mean-pooled to produce static summary vectors  $\bar{\mathbf{f}}_p \in \mathbb{R}^{d'_{\text{gc\_out}}}$ , which encapsulate the overall kinematics of part  $p$  for subsequent hyperbolic projection.

## C.2 Hyperbolic Alignment Strategies: Detailed Implementation

This section provides a more detailed explanation of the two hyperbolic alignment strategies introduced in Section 3.3 of the main paper. These strategies are designed to regularize the mT5 model by aligning pose and text representations within the Poincaré ball.

### C.2.1 Pooled Method (Global Semantic Alignment)

This strategy aims to align the holistic semantic content of the sign language video (represented by pose features) with the corresponding text translation.

**1. Part-Specific Hyperbolic Embeddings:** The temporally mean-pooled Euclidean feature vectors  $\bar{\mathbf{f}}_p$  for each anatomical part  $p$  (body, hands, face) are projected into the Poincaré ball  $\mathbb{B}_c^{d_{\text{hyp}}}$ . This projection, yielding hyperbolic embeddings  $\mathbf{h}_p$ , is achieved using the `HyperbolicProjection` layer (Listing 4 in Appendix G), as defined in Eq. (4) of the main paper:

$$\mathbf{h}_p = \exp_0^c(s_p \mathbf{W}^p \bar{\mathbf{f}}_p). \quad (2)$$

Here,  $\mathbf{W}^p$  represents a linear layer for part  $p$ , and  $s_p$  is a learnable scalar that adaptively scales the tangent space representation before the exponential map  $\exp_0^c(\cdot)$  projects it onto the manifold.

**2. Weighted Fréchet Mean for Global Pose Representation:** The set of part-specific hyperbolic embeddings  $\{\mathbf{h}_p\}$  is aggregated into a single global pose representation  $\boldsymbol{\mu}_{\text{pose}} \in \mathbb{B}_c^{d_{\text{hyp}}}$ . This is achieved by computing their weighted Fréchet mean, which is the hyperbolic analogue of a weighted average. The Fréchet mean is defined as the point that minimizes the sum of squared weighted geodesic distances to all input points:

$$\boldsymbol{\mu}_{\text{pose}} = \underset{\boldsymbol{\mu} \in \mathbb{B}_c^{d_{\text{hyp}}}}{\operatorname{argmin}} \sum_{p=1}^P w_p d_{\mathbb{B}_c}^2(\boldsymbol{\mu}, \mathbf{h}_p). \quad (3)$$

The weights  $w_p$  are designed to give more importance to parts whose embeddings are further from the origin of the Poincaré ball (i.e., parts with more “hyperbolic energy” or distinctness), normalised via softmax:

$$w_p = \frac{\exp(d_{\mathbb{B}_c}(\mathbf{0}, \mathbf{h}_p)/\lambda_w)}{\sum_{j=1}^P \exp(d_{\mathbb{B}_c}(\mathbf{0}, \mathbf{h}_j)/\lambda_w)}. \quad (4)$$

Here,  $\lambda_w$  is a temperature parameter for the softmax (e.g., fixed to 1.0 in our experiments) controlling the sharpness of the weight distribution. The computation is performed iteratively as detailed in Algorithm 1 of the main paper and Listing 5 (Appendix G).

**3. Global Text Representation:** Similarly, a global hyperbolic text embedding  $\mathbf{h}_{\text{text}} \in \mathbb{B}_c^{d_{\text{hyp}}}$  is derived from the mT5 model’s output. Euclidean token embeddings from the final layer of the mT5 decoder are first mean-pooled (respecting padding masks) to obtain a single sentence-level vector  $\bar{\mathbf{e}}_{\text{text}}$ . This vector is then projected into  $\mathbb{B}_c^{d_{\text{hyp}}}$  using a dedicated hyperbolic projection layer (structurally identical to Eq. (2)):

$$\mathbf{h}_{\text{text}} = \exp_0^c(s_{\text{text}} \mathbf{W}^{\text{text}} \bar{\mathbf{e}}_{\text{text}}). \quad (5)$$

The implementation details are shown in Listing 6 (Appendix G).

**4. Contrastive Alignment:** Finally, the geometric contrastive loss (Eq. (5) in the main paper) is applied between batches of these global pose embeddings  $\{\boldsymbol{\mu}_{\text{pose},i}\}$  and global text embeddings  $\{\mathbf{h}_{\text{text},i}\}$ . This encourages semantically similar pose-text pairs to be closer in hyperbolic space.

## C.2.2 Token Method (Fine-Grained Part-Text Alignment)

This strategy facilitates a more detailed alignment by relating individual pose part embeddings  $\{\mathbf{h}_p\}$  with contextually relevant text segment embeddings  $\{\mathbf{c}_p\}$ .

**1. Hyperbolic Pose Part Embeddings  $\{\mathbf{h}_p\}$ :** These are obtained exactly as in the Pooled Method, using Eq. (2). Each  $\mathbf{h}_p$  represents a specific anatomical part’s overall kinematic signature.

**2. Hyperbolic Text Token Embeddings:** Instead of a global text embedding, each Euclidean text token embedding  $\mathbf{e}_{\text{token},j}$  (from the mT5 decoder’s final layer) is individually projected into the Poincaré ball  $\mathbb{B}_c^{d_{\text{hyp}}}$ :

$$\mathbf{h}_{\text{token},j} = \exp_0^c(s_{\text{text}} \mathbf{W}^{\text{text}} \mathbf{e}_{\text{token},j}). \quad (6)$$

This results in a sequence of hyperbolic token embeddings  $\{\mathbf{h}_{\text{token},j}\}_{j=1}^{L_t}$ , where  $L_t$  is the text sequence length.

**3. Hyperbolic Attention Mechanism:** For each hyperbolic pose part embedding  $\mathbf{h}_p$  (acting as a query), a contextual text embedding  $\mathbf{c}_p$  is generated. This is achieved using a hyperbolic attention mechanism (see Listing 7 in Appendix G) that operates as follows:

- Key Transformation:** The hyperbolic text token embeddings  $\{\mathbf{h}_{\text{token},j}\}$  serve as keys. The embeddings are first transformed using learnable Möbius transformations to enhance their representational capacity:

$$\mathbf{k}_j = (\mathbf{M}_{\text{key}} \otimes_c \mathbf{h}_{\text{token},j}) \oplus_c \mathbf{b}_{\text{key}},$$

where  $\mathbf{M}_{\text{key}}$  is a learnable Möbius matrix and  $\mathbf{b}_{\text{key}}$  is a learnable Möbius bias vector.

- Attention Scores:** Attention scores are computed based on the negative geodesic distance between each pose query  $\mathbf{h}_p$  and each transformed text key  $\mathbf{k}_j$ :

$$\text{score}_{pj} = -d_{\mathbb{B}_c}(\mathbf{h}_p, \mathbf{k}_j).$$

- Attention Weights:** These scores are normalised using a softmax function (after applying padding masks) to obtain attention weights  $\alpha_{pj}$ :

$$\alpha_{pj} = \text{softmax} \left( \frac{\text{score}_{pj}}{\tau_{\text{attn}}} \right),$$

where  $\tau_{\text{attn}}$  is a learnable temperature parameter for the attention mechanism, distinct from the temperature in the contrastive loss.

- Contextual Text Embedding  $\mathbf{c}_p$ :** The contextual text embedding  $\mathbf{c}_p$  corresponding to pose part  $\mathbf{h}_p$  is then computed as the hyperbolic weighted midpoint (a generalization of weighted Fréchet mean for two points, or an aggregation using Einstein midpoints) of the original (untransformed) hyperbolic text token embeddings  $\{\mathbf{h}_{\text{token},j}\}$  (acting as values), using the attention weights  $\{\alpha_{pj}\}$ .

**4. Contrastive Alignment:** The geometric contrastive loss (Eq. (5), main paper) is then applied for each pair  $(\mathbf{h}_{p,i}, \mathbf{c}_{p,i})$  across the batch. The total regularization loss for this strategy is the average of these individual contrastive losses over all parts  $P$ .

## C.2.3 Intuition Behind the Token Method

While the Pooled Method aligns the overall semantics of a sign sequence with its translation, it may not capture how specific signing elements (e.g., a handshape, movement, or facial expression) correspond to particular words or phrases. The Token Method aims to establish this more fine-grained understanding.

The core intuition is as follows:

- Compositional Language Understanding:** Sign languages, like spoken/written languages, are compositional. Different articulators (hands, body, face) convey distinct lexical or grammatical information. The Token Method attempts to map these compositional units from pose to corresponding textual tokens (words/sub-words).

2. **Targeted Part-to-Segment Alignment:** Instead of a single global comparison, this method learns to connect individual pose part representations (e.g., features for the dominant hand, to the most relevant segments of the textual translation.
3. **Pose Parts as Queries, Text Tokens as Sources:** Each hyperbolic pose part embedding  $\mathbf{h}_p$  acts as a “query,” effectively asking: “Which text tokens are most semantically relevant to this pose feature?” The sequence of hyperbolic text token embeddings  $\{\mathbf{h}_{\text{token},j}\}$  serves as the “information source” for these queries.
4. **Hyperbolic Attention for Geometric Relevance:**
  - Relevance between a pose part query  $\mathbf{h}_p$  and a (transformed) text token key  $\mathbf{k}_j$  is measured by their geodesic distance  $d_{\mathbb{B}_c}(\mathbf{h}_p, \mathbf{k}_j)$  in the learned hyperbolic space. A smaller distance implies higher relevance. Using hyperbolic geometry allows these comparisons to potentially leverage latent hierarchical relationships between concepts.
  - Learnable Möbius transformations on text tokens (to get keys  $\mathbf{k}_j$ ) enable the model to learn distinct tokens relevant to different pose parts (e.g., a verb token might be transformed to be closer to a body movement embedding).
  - Standard attention weights  $\alpha_{pj}$  then quantify the contribution of each text token  $j$  to the meaning conveyed by pose part  $p$ .
5. **Learning Textual Context for Each Pose Part:** The contextual text embedding  $\mathbf{c}_p$  is a hyperbolic weighted midpoint of all text token embeddings (values), using the attention weights  $\alpha_{pj}$ . Thus,  $\mathbf{c}_p$  is a summary of the sentence, but specifically customised by the interaction of pose part  $p$ .
6. **Refined Contrastive Learning:** The model is regularised to make each pose part embedding  $\mathbf{h}_p$  close to its corresponding contextual text view  $\mathbf{c}_p$  in hyperbolic space, while pushing it away from non-corresponding pairs.
7. **Overall Benefit:** This detailed, part-specific alignment encourages the mT5 model to learn more precise mappings between kinematic features of different articulators and semantic units within the text. For example, it can help distinguish visually similar signs based on subtle hand details (encoded in  $\mathbf{h}_{\text{hand}}$ ) that correlate with specific words, leading to more accurate and nuanced translations.

## D Mathematical Foundations (Summary)

This section recalls two geometric components that Geo-Sign relies on:

- the *Weighted Fréchet Mean* inside the Poincaré ball (used in Algorithm 1 of the paper);
- the Euclidean gradient of the hyperbolic distance that appears in the contrastive loss.

Readers interested only in the final formulas can jump to Eqs. (7) and (8).

### D.1 Fréchet Mean in the Poincaré Ball

Given points  $x_1, \dots, x_N$  in a metric space  $(\mathcal{M}, d)$  with normalised weights  $w_i > 0$ ,  $\sum_i w_i = 1$ , the **Fréchet mean** minimises

$$\mathcal{F}(\mu) = \sum_{i=1}^N w_i d^2(\mu, x_i), \quad \mu^* = \arg \min_{\mu \in \mathcal{M}} \mathcal{F}(\mu).$$

**Intuition.** Why not simply average the embeddings in Euclidean space? Two issues appear inside the curved Poincaré ball:

- (a) **Manifold constraint.** A Euclidean average of interior points can fall *outside* the ball, i.e. outside valid hyperbolic space, forcing an ad-hoc projection that distorts geometry.
- (b) **Metric distortion.** Euclidean distance underestimates separation near the boundary because the hyperbolic metric stretches space there. A straight average therefore over-emphasises central points and washes out fine structure carried by peripheral ones.

237 The intrinsic Fréchet mean lives on the manifold and uses the true hyperbolic distance, so it respects  
238 curvature.

239 **Why distance-based weights?** Each pose part (body, face, left hand, right hand) yields a hyperbolic  
240 embedding  $h_p$ . We set  $w_p \propto \exp(d_{\mathbb{B}_c}(0, h_p)/\lambda_w)$  so parts farther from the origin—in regions of  
241 higher curvature and greater discriminative power—receive more influence. Without this weighting  
242 the mean would drift toward the centre, diluting information contributed by the hands and face.

243 **Iterative update.** On any Riemannian manifold the mean is found by Riemannian gradient descent;  
244 the update at iteration  $k$  is

$$\mu^{(k+1)} = \exp_{\mu^{(k)}} \left( \eta_k \sum_{i=1}^N w_i \log_{\mu^{(k)}}(x_i) \right), \quad (7)$$

245 with step size  $\eta_k > 0$ .

246 [Convergence in  $\mathbb{B}_c^d$ ] The Poincaré ball  $\mathbb{B}_c^d$  is a Hadamard manifold, hence  $\mathcal{F}$  is strictly convex and  
247 has a *unique* minimiser  $\mu^*$ . Let  $L$  be the Lipschitz constant of  $\nabla \mathcal{F}$  on the geodesic convex hull of  
248  $\{x_i\}$ . If  $0 < \eta_k \leq 2/L$  for all  $k$ , the iterates (7) converge to  $\mu^*$ . In practice we observe  $L \leq 2$ , so the  
249 simple choice  $\eta_k = 1$  is usually sufficient and used in our approach.

## 250 D.2 Gradient of the Hyperbolic Distance

251 For  $u, v \in \mathbb{B}_c^d$  let  $w = (-u) \oplus_c v$  (the Möbius difference, i.e. the “vector” from  $u$  to  $v$  transported to  
252 the origin). The Poincaré distance is

$$d_{\mathbb{B}_c}(u, v) = \frac{2}{\sqrt{c}} \operatorname{artanh}(\sqrt{c} \|w\|_2).$$

253 Differentiating (12; 5) gives the Euclidean gradient required for autograd:

$$\nabla_u d_{\mathbb{B}_c}(u, v) = -\frac{2}{\lambda_u^c \lambda_v^c} \frac{w}{\|w\|_2} \frac{1}{1 - c\|w\|_2^2} \quad (8)$$

254 with conformal factor  $\lambda_x^c = \frac{2}{1 - c\|x\|_2^2}$ . The same formula (with sign reversed) holds for  $\nabla_v$ .

255 The update rule (7) and the gradient (8) provide all the geometric tools needed by Geo-Sign’s  
256 hyperbolic contrastive regulariser.

## 257 E Learnable Model Parameters: $c$ and $\alpha$

258 Our Geo-Sign model incorporates several learnable parameters beyond standard network weights.  
259 This section details two key ones: the manifold curvature  $c$  and the loss blending factor  $\alpha$ .

### 260 E.1 Discussion on Learnable Curvature $c$

261 The curvature of the Poincaré ball,  $\kappa = -c$  (where  $c > 0$ ), is a crucial hyperparameter that dictates  
262 the “shape” of the hyperbolic space. Instead of fixing  $c$  heuristically, we make it a learnable parameter  
263 of our model (see Listing 9 in Appendix G).

264 **Optimization Strategy:** The curvature magnitude  $c$  is initialised (e.g., via `args.init_c` as men-  
265 tioned in the main paper’s experiments) and then updated via standard gradient descent as part of the  
266 end-to-end training process. The `geoopt` library facilitates this by defining  $c$  as an `nn.Parameter`  
267 within its `PoincareBall` manifold object when `learnable=True`.

268 The main paper’s ablation studies (Table 2a) show that initializing  $c$  in the range of 1.0 – 2.0 (e.g.,  
269 optimal BLEU-4 at  $c = 1.5$ ) yields strong performance. Figure 1 illustrates how  $c$  adapts during  
270 training from different initializations.



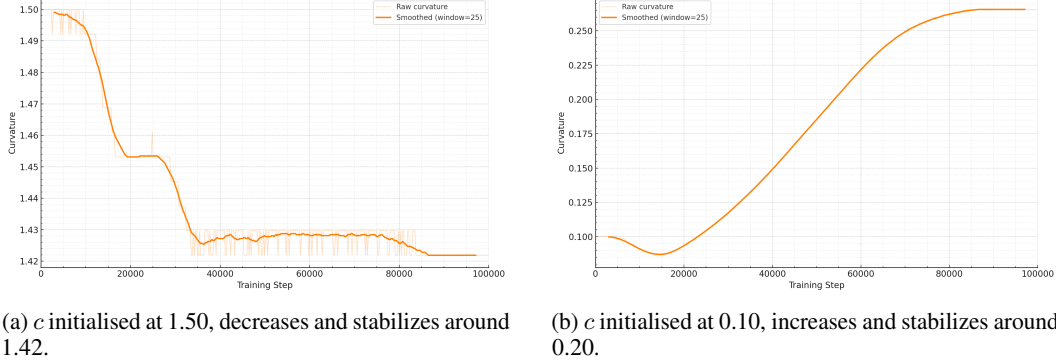


Figure 1: Evolution of the learnable manifold curvature  $c$  during training for different initializations. (a) When initialised at  $c = 1.50$ , the curvature magnitude slightly decreases, suggesting an optimal value around 1.42 for this setup. (b) When initialised at a low  $c = 0.10$ , the curvature increases, indicating the model benefits from more “hyperbolic space” initially. It stabilizes around  $c = 0.20$ , potentially influenced by the dynamic  $\alpha$  schedule that reduces regularization emphasis over time.

## 271 E.2 Discussion on Loss Blending Factor $\alpha$

The total training loss  $\mathcal{L}_{\text{total}}$  is a weighted combination of the primary cross-entropy translation loss  $\mathcal{L}_{\text{CE}}$  and our hyperbolic contrastive regularization term  $\mathcal{L}_{\text{hyp\_reg}}$ :

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{CE}} + (1 - \alpha) \cdot \mathcal{L}_{\text{hyp\_reg}}.$$

272 The blending factor  $\alpha$  is not fixed but is dynamically adjusted during training. This dynamic  
 273 scheduling allows the model to potentially benefit from different loss emphases at different training  
 274 stages. The calculation of  $\alpha$  at each training step (see Listing 10 in Appendix G) is:

$$\alpha_{\text{final}} = \text{clamp}((\alpha_{\text{init}} + 0.1 \cdot \text{progress}) + \sigma(\text{logit}_{\alpha}) \cdot 0.2, 0.1, 1.0), \quad (9)$$

275 where:

- 276 •  $\alpha_{\text{init}}$  is the initial value for the blending factor, specified as a hyperparameter (e.g.,  
 277 `args.alpha = 0.7` from the main paper’s ablations, Table 2b, which was found to be  
 278 optimal).
- 279 • `progress` is the current training progress, calculated as  $\frac{\text{current\_training\_step}}{\text{total\_training\_steps}}$ , ranging from 0 to 1.  
 280 This component introduces a linear ramp, potentially increasing  $\alpha$ ’s baseline by up to 0.1  
 281 over the course of training.
- 282 • `logit $\alpha$`  is an `nn.Parameter` (a learnable scalar, referred to as `self.loss_alpha_logit` in  
 283 the code).  $\sigma(\cdot)$  is the sigmoid function, so  $\sigma(\text{logit}_{\alpha})$  maps this learnable scalar to the range  
 284  $(0, 1)$ . This term provides a learnable adjustment to  $\alpha$  in the range of  $[0, 0.2]$ .
- 285 • `clamp( $\cdot$ , 0.1, 1.0)` ensures that the final  $\alpha_{\text{final}}$  remains within the bounds  $[0.1, 1.0]$ .

286 This dynamic  $\alpha$  allows for an initial phase where the hyperbolic regularization might have more  
 287 relative influence (if  $\alpha_{\text{init}}$  is smaller), gradually shifting emphasis or allowing the model to fine-tune  
 288 the balance via the learnable component. The ablation study in the main paper (Table 2b) indicates  
 289 that an initial  $\alpha_{\text{init}} = 0.7$  (i.e., 30% weight to  $\mathcal{L}_{\text{hyp\_reg}}$  initially) provides the best results, highlighting  
 290 the complementary role of the hyperbolic regularization.

## 291 F Experimental Setup, Analysis, and Qualitative Results

### 292 F.1 Computational Profile

293 This section discusses the computational profile of Geo-Sign, comparing it to a baseline Uni-Sign  
 294 (Pose) model without hyperbolic regularization. The analysis is based on DeepSpeed profiler outputs  
 295 for models run with a batch size of 32 on the CSL-Daily dataset for the Sign Language Translation  
 296 (SLT) task.



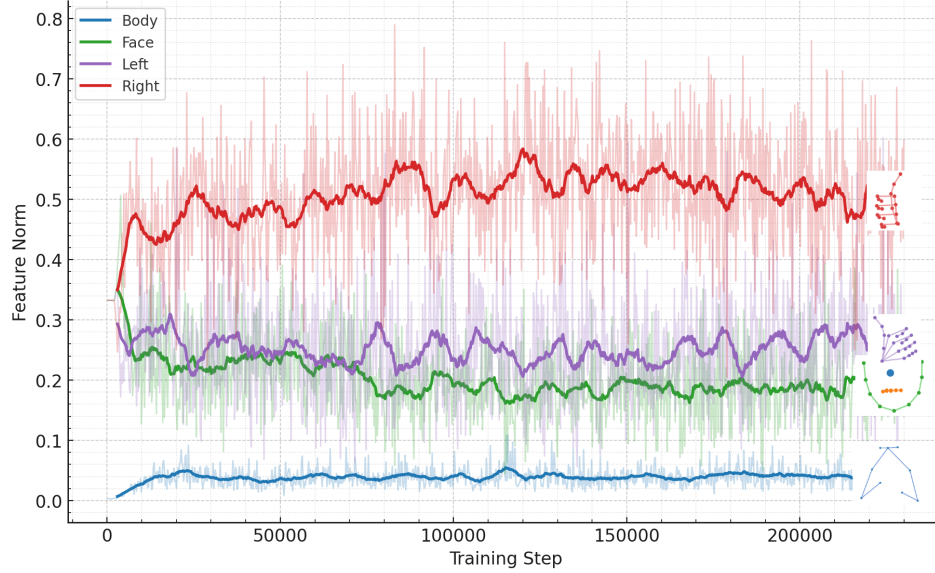


Figure 2: Plot of the geodesic distances from the origin (0) of the Poincaré disk to the hyperbolic pose embeddings ( $h_p$ ) during training, averaged per part type. This shows how features for different parts utilize the hyperbolic space. For instance, right hand features (often conveying detailed lexical information) tend to move further from the origin, leveraging more of the hyperbolic curvature for discriminability. Body and face features, which might represent broader semantics or prosody, may remain closer to the Euclidean-like central region.

297 **Experimental Context:** Key experimental conditions for fine-tuning include:

- 298 • **Hardware:** 4 NVIDIA RTX 3090 GPUs.
- 299 • **Training Time:** Approximately 10 hours for 40 epochs of fine-tuning on CSL-Daily.
- 300 • **Precision:** Mixed-precision training (bf16) is used for standard PyTorch layers, while float32 is maintained for Geoopt hyperbolic operations to ensure numerical stability.
- 301 • **Batching Strategy:** With an effective batch size of 8 per GPU, the model occupies  $\approx 20$ GB of memory. During training, we increase the total batch size to 32 and accumulate gradients over 8 steps, achieving a hypothetical batch size of 256. For the following profiler analysis, we report results for a single GPU with a batch size of 8 to provide a clear per-device profile.

### 306 F.1.1 Profiler Summary and Comparative Analysis

307 Table 1 summarizes key metrics from the profiler. Parameter counts are consistent with the main paper’s Table 1, while MACs (Multiply-Accumulate operations) and Latency are derived from DeepSpeed profiler outputs for a batch size of 8. Table 2 provides a comparison of model parameters against other gloss-free methods.

Table 1: Computational profile comparison at Batch Size 8: Baseline Uni-Sign (Pose) vs. Geo-Sign variants. Parameter counts from main paper’s Table 1. MACs and Latency from DeepSpeed profiler outputs. “Hyperbolic Proj. Layer MACs” reflects profiled contributions from the learnable linear transformations within these layers.

Model Variant (Batch Size 8)	Total Params (M)	Added Params (M)	Total Fwd MACs (GMACs)	Hyperbolic Proj. Layer MACs (MMACs)	Fwd Latency (ms)	Latency Increase (%)
Baseline Uni-Sign (Pose)	587.75	-	116.59	-	415.73	-
Geo-Sign (Hyperbolic Pooled)	588.21	0.46	116.60	3.67	1630.00	292.10
Geo-Sign (Hyperbolic Token)	589.10	1.35	116.60	$\approx 9.96$	2550.00	513.40

311 **Parameter Overhead:** The increase in parameters due to the hyperbolic components is marginal compared to the overall model size, which is dominated by the mT5 language model ( $\approx 582.4$ M parameters).

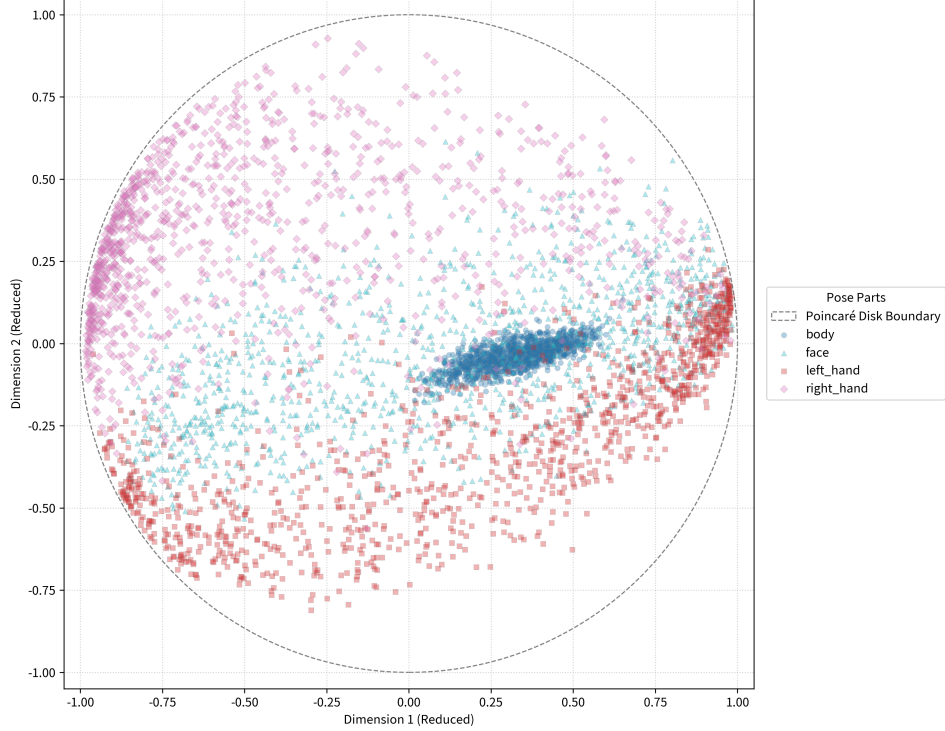


Figure 3: PCA projection of 1000 hyperbolic pose part embeddings (log-mapped to the tangent space at origin, then PCA-reduced to 2D) visualised within the Poincaré disk. Body features (blue) are tightly clustered near the origin, suggesting their discriminability is well-handled in a more Euclidean-like region. Hand features (left: red square, right: pink diamond) and face features (light blue triangle) are more dispersed, with hand features often pushed towards the periphery. This indicates these parts benefit from the increased representational capacity near the boundary of the Poincaré disk, where hyperbolic geometry provides more “space” to distinguish subtle variations crucial for sign language semantics.

- Baseline Uni-Sign (Pose):  $\approx 587.75\text{M}$  parameters.
- **Geo-Sign (Pooled)**: Adds  $\approx 0.46\text{M}$  parameters, primarily from the five hyperbolic projection layers (one for each of the four pose parts and one for the pooled text embedding).
- **Geo-Sign (Token)**: Adds  $\approx 1.35\text{M}$  parameters. This includes the  $\approx 0.46\text{M}$  for projection layers plus an additional  $\approx 0.89\text{M}$  for the learnable parameters within the hyperbolic attention mechanism (Möbius matrices and biases).

In both Geo-Sign variants, the parameter overhead from hyperbolic components is less than 0.25% of the total model size. As shown in Table 2, our Geo-Sign models achieve competitive or superior performance to recent RGB-based methods while maintaining a significantly smaller total parameter count. This highlights the efficiency of enhancing skeletal representations with geometric priors, challenging the trend that relies solely on scaling up visual encoders and language model decoders for performance gains in SLT.

**MACs Analysis:** The DeepSpeed profiler indicates that the total forward MACs are very similar across all configurations at this batch size:

- Baseline Uni-Sign (Pose):  $\approx 116.59$  GMACs.
- Geo-Sign (Hyperbolic Pooled):  $\approx 116.60$  GMACs. The profiler attributes  $\approx 3.67$  MMACs to the linear transformations within its HyperbolicProjection layers.
- Geo-Sign (Hyperbolic Token):  $\approx 116.60$  GMACs. Its HyperbolicProjection layers account for  $\approx 9.96$  MMACs from their linear components.

Table 2: Sign Language Translation performance (Test Set: BLEU-4, ROUGE-L) and model parameters on CSL-Daily. Scores are percentages (%). Higher is better. ‘Pose’ and ‘RGB’ indicate input modalities. VE/LM/Total Params are in Millions (M). Approx. values indicated by  $\approx$ . Data from CSL-Daily (Train: 18,401 sentences / 20.62 hours).

Method	VE Name	VE	LM Name	LM	Total	Modality		Test Set	
		Params (M)		Params (M)	Params (M)	Pose	RGB	B-4	R-L
Gloss-Free Methods (Prior Art)									
MSLU (20)	EffNet	5.3	mT5-Base	582.4	587.7	✓	–	11.42	33.80
SLRT (1) (G-Free)	EffNet	5.3	Transformer	≈30	≈35.3	–	✓	3.03	19.67
GASLT (18)	I3D	13	Transformer	≈30	≈43.0	–	✓	4.07	20.35
GFSLT-VLP (19)	ResNet18	11.7	mBart	680	691.7	–	✓	11.00	36.44
FLa-LLM (3)	ResNet18	11.7	mBart	680	691.7	–	✓	14.20	37.25
Sign2GPT (16)	DinoV2	21.0	XGLM	1732.9	1753.9	–	✓	15.40	42.36
SignLLM (6)	ResNet18	11.7	LLaMA-7B	6738.4	6750.1	–	✓	15.75	39.91
C <sup>2</sup> RL (2)	ResNet18	11.7	mBart	680	691.7	–	✓	21.61	48.21
Our Models and Baselines									
Uni-Sign (10) (Pose)	GCN	5.3	mT5-Base	582.4	587.7	✓	–	25.61	54.92
Uni-Sign (10) (Pose+RGB)	EffNet+GCN	9.7	mT5-Base	582.4	592.1	✓	✓	26.36	56.51
Geo-Sign (Hyperbolic Pooled)	GCN+Geo	5.8	mT5-Base	582.4	588.21	✓	–	27.17	57.75
Geo-Sign (Hyperbolic Token)	GCN+Geo+Attn	6.7	mT5-Base	582.4	589.1	✓	–	27.42	57.95

The MACs from the learnable linear transformations within the hyperbolic projection layers constitute a very small fraction ( $< 0.01\%$ ) of the total model MACs. The bulk of MACs originates from the mT5 model (profiled at  $\approx 66.29$  GMACs) and the ST-GCN modules (profiled at  $\approx 49.93$  GMACs). We should note, however, that standard profilers (like DeepSpeed’s MAC counter) primarily quantify MACs from common operations like convolutions and linear layers. The computational cost of specialised geometric functions within `geoopt` (e.g., `manifold.dist`, `expmap0`, `logmap0`, Möbius arithmetic) is not explicitly broken out as distinct hyperbolic operation MACs. These functions often involve sequences of elementary operations that are not all MAC-based (e.g., square roots, divisions, trigonometric functions like `artanh` or `tanh`). Thus, their computational load may be underestimated by MAC counters and is often better reflected in measured latency.

**Latency Analysis (Batch Size 8):** Latency figures clearly reveal the primary computational overhead introduced by the hyperbolic components during training:

- Baseline Uni-Sign (Pose):  $\approx 416$  ms forward latency per batch.
- Geo-Sign (Hyperbolic Pooled):  $\approx 1630$  ms (1.63 s), an increase of  $\approx 1214$  ms or  $\approx 292\%$  over the baseline (approx.  $3.9\times$  slowdown).
- Geo-Sign (Hyperbolic Token):  $\approx 2550$  ms (2.55 s), an increase of  $\approx 2134$  ms or  $\approx 513\%$  over the baseline (approx.  $6.1\times$  slowdown).

The substantial increase in training latency, despite modest increases in parameters and profiled MACs from learnable layers, underscores that the geometric operations themselves are the main performance consideration during the training phase. These operations (e.g., geodesic distance, exponential/logarithmic maps, Möbius transformations) are inherently more complex than their Euclidean counterparts. The Token method is notably slower than the Pooled method during training due to its per-token hyperbolic attention.

Importantly, a key advantage of our regularization approach is that these geometric operations and the hyperbolic branch are **not utilised at inference time**. Consequently, Geo-Sign models incur no additional latency increase over the baseline Uni-Sign (Pose) model during inference, preserving efficiency for deployment.

### F.1.2 Discussion on Data Efficiency

While not directly measured in this profiler analysis, it is hypothesised that skeletal data’s abstraction from visual noise (lighting, background) can enhance robustness and generalization, especially when training data is limited. Hyperbolic geometry further imposes a structural prior (hierarchy) on the representation space. This inductive bias could potentially improve data efficiency by guiding the learning process, particularly in scenarios with sparse data, although specific experiments to quantify this effect were not part of the current study. One trade-off of this approach is that we cannot directly

leverage large pre-trained visual encoders as in the case of other RGB approaches, and so pretraining on a sign-specific dataset like CSL-News (1,985 hours, used by Uni-Sign) is essential. However, this pretraining data size is comparable to that used by other SLT methods which might use datasets like How2Sign (4) (2000 hours) or YouTube-ASL (15; 14) (6000 hours). We anticipate that our method would continue to scale well with larger pre-training datasets in other sign languages, though resource constraints prevented evaluation of this aspect at the time of submission.

## F.2 Further Technical Implementation Details

This section provides additional details that are pertinent for a full understanding and potential reimplementaion of Geo-Sign.

- **Core Libraries:** Our implementation relies on PyTorch (13) as the primary deep learning framework. For Transformer models, we utilize the HuggingFace Transformers library (?). All hyperbolic geometry operations and Riemannian optimization are handled by the Geoopt library (9). For distributed training and profiling, DeepSpeed is employed.
- **Hyperparameter Tuning Strategy:** Key hyperparameters specific to the hyperbolic components, such as the initial curvature  $c$ , the initial loss blending factor  $\alpha_{\text{init}}$  (referred to as `args.alpha` in code/main paper), and the hyperbolic embedding dimension  $d_{\text{hyp}}$ , were tuned using a grid search strategy on the CSL-Daily development set.

- **Numerical Stability Measures:**

- Operations within `geoopt` are performed using `float32` precision to maintain numerical stability, while the rest of the model uses mixed precision.
- Small epsilon values (e.g.,  $10^{-5}$ ) are added in denominators and inside logarithms/arc-tanh functions where appropriate to prevent division by zeros.
- **Tangent Vector Clipping:** Before applying an exponential map  $\exp_x^c(\mathbf{v})$  from a point  $x$  with a tangent vector  $\mathbf{v}$ , especially  $\exp_0^c(\mathbf{v})$ , it’s crucial to ensure the resulting point remains strictly within the Poincaré ball and that the norm of  $\mathbf{v}$  doesn’t cause numerical issues in  $\tanh(\cdot)$ . We apply a clipping strategy as mentioned in Section 3.4 of the main paper:

$$\mathbf{v}_{\text{clipped}} \leftarrow \frac{\mathbf{v}}{\max(1, \sqrt{c}\|\mathbf{v}\|_2 + \epsilon_{\text{clip}})},$$

for a small  $\epsilon_{\text{clip}} > 0$  (e.g.,  $10^{-5}$ ). This ensures that the argument to  $\tanh$  in  $\exp_0^c$  does not become excessively large and that mapped points do not reach or exceed the boundary of the Poincaré ball. The `project=True` flag in `geoopt`’s `expmap` functions also helps enforce this by projecting points back onto the ball if they numerically fall outside.

- **Gradient Clipping:** Standard norm-based gradient clipping is applied to all model parameters during training to stabilize the optimization process.

In Table 3 we provide the full hyper-parameters for the best performing model. The full code will be released following the review process.

## F.3 Qualitative Results

**Additional Figures:** Figure 2 (similar to aspects shown in Figure 2 of the main paper, concerning learned embedding distributions) illustrates the dynamic utilization of the hyperbolic manifold by showing the average geodesic distance of different pose part embeddings from the origin during training. Notably, features corresponding to hand articulations, which often carry fine-grained lexical information, tend to migrate towards the periphery of the Poincaré disk. This suggests that the model leverages the increased representational capacity in high-curvature regions to distinguish subtle hand-based signs.

Furthermore, Figure 3 (again, related to Figure 2 of the main paper, specifically the UMAP projections) provides a PCA-reduced visualization of the learned hyperbolic pose part embeddings projected onto the 2D Poincaré disk for 1000 poses. This plot typically reveals a structured distribution where body features cluster near the origin (a more Euclidean-like region suitable for broader semantics), while

Table 3: Hyperparameter summary for Geo-Sign experiments. Values are for the best reported model configuration.

Category	Hyperparameter	Value	Description
<b>General Training Configuration</b>			
	Random Seed	42	Seed for reproducibility
	Training Epochs	40	Number of fine-tuning epochs on CSL-Daily
	Batch Size (per GPU)	8	Micro-batch size per GPU
	Gradient Accumulation Steps	8	Effective batch size becomes $8 \times \text{accum\_steps} \times \text{num\_gpus}$
	Training Precision (dtype)	bf16	Mixed precision training data type
<b>Data Handling</b>			
	Max Pose Sequence Length	256	Maximum number of frames for pose sequences
	Max Target Text Length (max_tgt_len)	100	Max new tokens for generation during evaluation
<b>Optimizer (Euclidean: ST-GCN, mT5, Linear Layers)</b>			
	Optimizer Type (opt)	AdamW	(11)
	Learning Rate (lr)	$3 \times 10^{-5}$	For Euclidean parameters (AdamW)
	AdamW $\beta_1, \beta_2$ (opt-betas)	[0.9, 0.999]	Exponential decay rates for moment estimates
	AdamW $\epsilon$ (opt-eps)	$1 \times 10^{-8}$	Term for numerical stability
	Weight Decay (weight-decay)	0.01	L2 penalty for Euclidean parameters
	LR Scheduler (sched)	Cosine Annealing	
	Warmup Epochs (warmup-epochs)	5	Number of epochs for LR warm-up
	Minimum LR (min-lr)	$1 \times 10^{-6}$	Lower bound for LR in scheduler
	Gradient Clipping Norm	1.0	Max norm for gradients
<b>Optimizer (Hyperbolic: Manifold Parameters, Projections)</b>			
	Optimizer Type	RAdam	Riemannian Adam (?)
	Learning Rate (hyp_lr)	$1 \times 10^{-3}$	For hyperbolic parameters (RAdam)
<b>Model Architecture</b>			
	ST-GCN Output Dimension (gcn_out_dim)	256	Output dimension of ST-GCN part streams
	mT5 Projection Dimension (hidden_dim)	768	Target dimension for projecting GCN features to match mT5
<b>Hyperbolic Regularization</b>			
	Hyperbolic Embedding Dimension ( $d_{\text{hyp}}, \text{hyp\_dim}$ )	256	Dimension of embeddings in Poincaré ball
	Initial Curvature ( $c_{\text{init}}, \text{init\_c}$ )	1.5	Initial value for learnable curvature $c$ (for best model)
	Loss Blend $\alpha_{\text{init}}$ (alpha)	0.70	Initial blending factor for $\mathcal{L}_{\text{CE}}$ vs $\mathcal{L}_{\text{hyp\_reg}}$ (for best model)
	Text Comparison Mode (hyp_text_cmp)	token	Strategy for aligning pose with text tokens (Token Method)
	Hyperbolic Contrastive Loss $\mathcal{L}_{\text{hyp\_reg}}$ :		
	Temperature ( $\tau$ )	Learnable	Temperature for scaling distances in contrastive loss
	Margin ( $m$ )	Learnable	Additive margin for negative pairs in contrastive loss
	Label Smoothing (label_smoothing_hyp)	0.2	Label smoothing for hyperbolic contrastive loss (InfoNCE)
<b>Loss Functions</b>			
	CE Loss Label Smoothing (label_smoothing)	0.2	Label smoothing for mT5 cross-entropy loss
<b>Distributed Training (DeepSpeed)</b>			
	ZeRO Optimization Stage (zero_stage)	2	DeepSpeed ZeRO Stage for memory efficiency
	Offload to CPU (offload)	False	Whether to offload optimizer/params to CPU

hand and face features are more dispersed, with hand features often populating regions further towards the boundary. This geometric organization, reflecting a learned kinematic hierarchy, likely contributes to the improved discriminability and, consequently, the enhanced translation quality demonstrated in the following examples. These visualizations support the hypothesis that the geometric biases induced by hyperbolic space aid in forming more effective representations for sign language translation.

**Translation Results:** In this section, we provide an overview of translation samples generated by Geo-Sign. All predictions are from our best-performing “Token” model. First, in Table 4, we show examples of prediction errors with analysis and a general measure of semantic similarity (introduced for readability, not a quantitative metric). English translations are automatically generated and then verified by a native Chinese speaker. We observe that translation quality with respect to semantics is generally high, though our method, like many SLT systems, can sometimes miss pronouns or struggle with complex tenses. In Table 5, we showcase examples where our approach generates perfect or near-perfect translations. Finally, in Table 6, we select some examples to compare our model’s output with that of the Uni-Sign (Pose) baseline. These comparisons illustrate improvements in semantic meaning and accuracy, consistent with the quantitative gains in ROUGE and BLEU-4 scores reported in the main paper.

Table 4: Examples of Prediction Errors and Analysis from Geo-Sign (Token Method)

Prediction (Unique Conceptual)	Ground Truth	Analysis of Error	Semantic Similarity
她今年50岁。(She is 50 years old.)	他今年四岁。(He is 4 years old.)	Pronoun error: 她 (she) vs. 他 (he). Number error: “50” (50) vs. 四 (four). The prediction gets the topic (age) but is wrong on subject and specific age.	Partial (topic: age)
今天星期五。(Today is Friday.)	今天星期几?(What day of the week is it today?)	Statement vs. Question: Prediction states a specific day. GT asks for the day. Character error: 五 (five) vs. 几 (how many/which).	Partial (topic: day of week)
你什么时候认识小张?(When did you meet Xiao Zhang?)	你和小张什么时候认识的?(When did you AND Xiao Zhang meet?)	Missing words: Prediction lacks “和” (and) and the particle “的”. This subtly changes the meaning from a one-way recognition to a mutual acquaintance.	High
我要去超市买椅子。(I want to go to the supermarket to buy a chair.)	我要去超市买椅子, 你去吗?(I want to go to the supermarket to buy a chair, are you going?)	Missing clause/question: Prediction omits the follow-up question “你去吗?” (are you going?).	High (core statement identical)
下午你们要去做什么?(What are you [plural] going to do in the afternoon?)	他们下午要做什么?(What are they going to do in the afternoon?)	Pronoun error: 你们 (you plural) vs. 他们 (they).	High
下午你们需要做什么?(What do you [plural] need to do this afternoon?)	他们下午要做什么?(What are they going to do this afternoon?)	Pronoun error: 你们 (you plural) vs. 他们 (they). Word choice: 需要 (need) vs. 要 (going to/want to) - subtle semantic shift, GT is more natural for general plans.	High
大家觉得什么时候去买椅子?(When does everyone think we should go buy chairs?)	他们想什么时候去买椅子?(When do they want to go buy chairs?)	Subject error: 大家 (everyone) vs. 他们 (they). Verb choice: 觉得 (feel/think) vs. 想 (want/think).	High
我手表不见了。(My watch is missing.)	这块手表是你的吗?(Is this watch yours?)	Different intent: Prediction states a loss. GT asks about ownership of a present watch. Both are about watches but different scenarios.	Medium (topic: watch)
你手表多少钱?(How much is your watch?)	这块手表多少钱买的?(How much did you buy this watch for?)	Missing context/words: Prediction is a bit abrupt. GT is more complete with “这块” (this) and “买的” (bought for).	High
我发现了他的偶像。(I discovered his idol.)	你看见我的杯子吗?(Did you see my cup?)	Completely different semantic intent and topic. Prediction is about an idol, GT is about a missing cup.	Very Low
爸爸的房间里大了。(It has become big in dad’s room / Dad’s room has become bigger.)	左边的房间是我爸爸妈妈的, 他们的房间很大。(The room on the left is my parents’, their room is very big.)	Garbled/incomplete prediction: The prediction is grammatically awkward and misses the entire context of the GT.	Low

Continued on next page

Table 4 – continued from previous page

Prediction	Ground Truth	Analysis of Error	Semantic Similarity
公司离家远,他为什么打车去公司? (The company is far from home, why does he take a taxi to the company?)	公司离家很远,她为什么不打车? (The company is very far from home, why doesn't she take a taxi?)	Pronoun error: 他 (he) vs. 她 (she). Logic error: Prediction asks why he does take a taxi, GT asks why she doesn't.	Medium
阴天说什么话?天气什么的,明天有事。 (What to say on a cloudy day? Weather something, have things to do tomorrow.)	阴天,电视上说多云,怎么了?明天有事? (Cloudy day, TV says it's overcast, what's up? Got plans tomorrow?)	Nonsensical/Garbled prediction: Prediction is very disjointed and doesn't make sense, while GT is a coherent conversation about weather and plans.	Low
桌子上有饮料,你想喝什么? (There are drinks on the table, what do you want to drink?)	桌上放着很多饮料,你喝什么? (There are many drinks on the table, what do you want to drink?)	Slight phrasing difference: “桌子上有” (On the table there are) vs. “桌上放着很多” (On the table are placed many). GT is slightly more natural. Prediction is still good.	High
我刚才在家里找了一个桌子,不是找了。 (I just looked for a table at home, not looked for.)	你去房间找找,是不是刚才放在桌子上了? (Go look in the room, was it just placed on the table?)	Different speaker and intent: Prediction is a confused statement about searching. GT is a directive and question to someone else.	Low
一个人的癌症会变得很可能。 (A person's cancer will become very possible.)	人体的许多器官都可能发生癌变。 (Many organs of the human body can become cancerous.)	Vague and unnatural prediction: “变得很可能” is awkward. GT is precise about “organs” and “癌变” (cancerous change).	Medium
老年人通过斑马线时可以走斑马线,而不走汽车。 (When elderly people cross the crosswalk, they can use the crosswalk, and not walk cars.)	一位老人正在慢慢地穿过斑马线,等待的司机却不耐烦地按起了喇叭。 (An old man was slowly crossing the crosswalk, but the waiting driver impatiently honked the horn.)	Nonsensical and irrelevant prediction: “而不走汽车” (and not walk cars) makes no sense. The GT describes a specific scenario.	Very Low

426

Table 5: Examples of Correct Predictions by Geo-Sign (Token Method)

Reference (Ground Truth)	Our Model Prediction (Perfect Match)
‘今天我想吃面条。’ (Today I want to eat noodles.)	‘今天我想吃面条。’ (Today I want to eat noodles.)
‘苹果是你买的吗?’ (Did you buy the apples?)	‘苹果是你买的吗?’ (Did you buy the apples?)
‘我昨天有点累。’ (I was a bit tired yesterday.)	‘我昨天有点累。’ (I was a bit tired yesterday.)
‘吃完午饭要多吃点水果。’ (Eat more fruit after lunch.)	‘吃完午饭要多吃点水果。’ (Eat more fruit after lunch.)

Continued on next page



Table 5 – continued from previous page

Reference (Ground Truth)	Our Model Prediction (Perfect Match)
‘我的妻子感冒了,我开车带她去医院。 (My wife has a cold, I will drive her to the hospital.)	‘我的妻子感冒了,我开车去医院。‘ (My wife has a cold, I will drive her to the hospital.)
‘我们会通过短信的方式来联系你。‘ (We will contact you via text message.)	‘我们会通过短信的方式来联系你。‘ (We will contact you via text message.)
‘我们将采用抽查的方式来进行检查。 (We will use random checks for inspection.)	‘我们将采用抽查的方式来进行检查。 (We will use random checks for inspection.)
‘你要把握好自己人生的方向。‘ (You need to grasp the direction of your own life.)	‘你要把握好自己人生的方向。‘ (You need to grasp the direction of your own life.)
‘病历是禁止涂抹、修改的。‘ (Medical records are not allowed to be smeared or altered.)	‘病历是禁止涂抹、修改的。‘ (Medical records are not allowed to be smeared or altered.)
‘他抛下家人,带着家中财物逃走了。‘ (He abandoned his family and fled with the family’s belongings.)	‘他抛下家人,带着家中财物逃走了。‘ (He abandoned his family and fled with the family’s belongings.)
‘这间玻璃作坊有一百年历史了。‘ (This glass workshop has a hundred years of history.)	‘这间玻璃作坊有一百年历史了。‘ (This glass workshop has a hundred years of history.)

427

Table 6: Comparative Analysis: Geo-Sign (Token) vs. Uni-Sign (Pose) - Selected Examples

Reference (Ground Truth)	Geo-Sign (Token) Prediction	Uni-Sign (Pose) Prediction
‘他每天回来都很累。‘ (He is very tired every day when he comes back.)	‘他每天来很累。‘ (He comes very tired every day.)	‘他每天来得及很累。‘ (He has enough time [to be/and is] very tired every day.)
‘小张,那个女生是你们公司的吗?你对她了解吗?‘ (Xiao Zhang, is that girl from your company? Do you know her?)	‘小张那个女生是你公司的吗?‘ (Xiao Zhang, is that girl from your company?)	‘那个小张是这家公司负责人,你了解吗?‘ (That Xiao Zhang is the person in charge of this company, do you understand/know?)
‘阴天,电视上说多云,怎么了?明天有事?‘ (Cloudy day, TV says it’s overcast, what’s up? Got plans tomorrow?)	‘阴天说什么话?天气什么的,明天有事。‘ (What to say on a cloudy day? Weather something, have things to do tomorrow.)	‘阴阳怪气地讲着天赋,不知不觉就发生了什么。‘ (Sarcastically talking about talent, something happened unknowingly.)
‘但是你一点也没瘦,你做什么运动了?‘ (But you haven’t lost any weight, what exercise have you been doing?)	‘但是你没有太吃饱,你去做什么运动?‘ (But you didn’t eat too full, what exercise are you going to do?)	‘但是你已经吃不消了,你能做什么呢?‘ (But you already can’t stand it, what can you do?)
‘手机选不到靠近窗户的位置。‘ (The mobile phone cannot select a seat near the window.)	‘靠近窗户的位置是不能选择的。‘ (Seats near the window cannot be selected.)	‘坐在靠近窗户的位置是不能取手机的。‘ (Sitting in a seat near the window, you cannot take out your mobile phone.)

Continued on next page

**Table 6 – continued from previous page**

<b>Reference (Ground Truth)</b>	<b>Geo-Sign (Token) Prediction</b>	<b>Uni-Sign (Pose) Prediction</b>
‘他对自己一直高标准严要求。’ (He has always had high standards and strict requirements for himself.)	‘他对自己有着严格的标准要求。’ (He has strict standard requirements for himself.)	‘他对自己最严格的标准提出了更高的要求。’ (He put forward higher requirements for his strictest standards.)
‘这位厨师制作的甜品,全部受欢迎。’ (The desserts made by this chef are all popular.)	‘厨师的作品很受欢迎。’ (The chef’s work is very popular.)	‘厨师在设计作品时非常受欢迎。’ (The chef is very popular when designing works.)

## 429 G Code Listings

430 This section provides key Python code snippets as referenced in the text.

```

431
432 15 import numpy as np
433 16 import torch # Used for self.A later
434 17
435 18 # (Helper function assumed to be defined elsewhere or inline)
436 19 def get_hop_distance(num_node, edge, max_hop=1):
437 20     # This function computes distances up to max_hop between nodes
438 21     adj = np.zeros((num_node, num_node))
439 22     for i, j in edge: # edge is a list of tuples (node_idx_1,
440 23         node_idx_2)
441 24         if i < num_node and j < num_node : # Basic check
442 25             adj[i, j] = 1
443 26             adj[j, i] = 1 # Assuming undirected graph for basic
444 27 connectivity
445 28
446 29 hop_dis = np.zeros((num_node, num_node)) + np.inf
447 30 transfer_mat = [np.linalg.matrix_power(adj, d) for d in range(
448 31 max_hop + 1)]
449 32 arrive_mat = (np.stack(transfer_mat) > 0)
450 33 for d in range(max_hop, -1, -1): # Check from furthest hop to
451 34 closest
452 35     hop_dis[arrive_mat[d]] = d
453 36 return hop_dis
454
455
456
457 35 class Graph:
458 36     # Initializes graph properties including layout (e.g., 'hand',
459 37     'body'),
460 38     # connectivity strategy, maximum hop distance, and dilation for
461 39 ST-GCNs.
462 40 def __init__(self, layout='custom', strategy='uniform', max_hop
463 41 =1, dilation=1):
464 42     self.max_hop = max_hop
465 43     self.dilation = dilation
466 44     self.get_edge(layout) # Sets self.num_node, self.edge, self
467 45 .center based on layout
468 46     # Computes hop distances between all pairs of nodes
469 47     self.hop_dis = get_hop_distance(self.num_node, self.edge,
470 48 max_hop=max_hop)
471 49     # Generates the adjacency matrix (self.A) based on the
472 50 chosen strategy
473 51     self.A = self.get_adjacency(strategy) # Store result
474
475 52 # Defines the number of nodes and edge connections for
476 53 different skeleton parts.
477 54 def get_edge(self, layout):
478 55     if layout == 'left' or layout == 'right': # Hand layout
479 56 configuration
480 57         self.num_node = 21 # 21 joints for a hand
481 58         self_link = [(i, i) for i in range(self.num_node)] #
482 59 Links of each node to itself
483 60         # Defines direct neighbor connections for hand joints
484 61         neighbor_1base = [ # Based on standard hand skeleton
485 62 topology
486 63             [0,1],[1,2],[2,3],[3,4], #
487 64 Thumb
488 65             [0,5],[5,6],[6,7],[7,8], #
489 66 Index finger
490 67             [0,9],[9,10],[10,11],[11,12], #
491 68 Middle finger

```

```

492 57         [0,13],[13,14],[14,15],[15,16], #
493     Ring finger
494 58         [0,17],[17,18],[18,19],[19,20]] #
495     Pinky finger
496 59         self.edge = self_link + neighbor_1base # All edges
497 60         self.center = 0 # Wrist joint is typically the center/
498     root of the hand graph
499 61         elif layout == 'body': # Body layout configuration
500 62             self.num_node = 9
501 63             self_link = [(i, i) for i in range(self.num_node)]
502 64             # Defines direct neighbor connections for body joints
503 65             neighbor_1base = [[0,1],[0,2],[0,3],[0,4], # Central
504     joint to limbs/head
505 66             [3,5],[5,7], # Left arm
506     chain
507 67             [4,6],[6,8]] # Right arm
508     chain
509 68             self.edge = self_link + neighbor_1base
510 69             self.center = 0 # A central body joint (e.g., neck or
511     spine base)
512 70             elif layout == 'face_all': # Combined face and head joints
513     (example mapping)
514 71             # This is an example mapping, actual joint definitions
515     from COCO-WholeBody used.
516 72             self.num_node = 16 # As per main paper's description of
517     face keypoints
518 73             self_link = [(i, i) for i in range(self.num_node)]
519 74             # Simplified example connectivity for face - actual
520     would be more complex
521 75             # and based on facial landmarks proximity or structure.
522 76             # The actual UniSign implementation uses specific
523     indices from COCO-WholeBody.
524 77             neighbor_1base = [(i, (i + 1) % self.num_node) for i in
525     range(self.num_node)] # Basic cyclic connections
526 78             self.edge = self_link + neighbor_1base
527 79             self.center = self.num_node // 2 # Example center
528 80             else:
529 81                 raise ValueError(f'Unknown graph layout: {layout}')
530 82
531 83     # Computes and normalizes the adjacency matrix based on the
532     chosen strategy.
533 84     def get_adjacency(self, strategy):
534 85         valid_hop = range(0, self.max_hop + 1, self.dilation)
535 86         adjacency = np.zeros((self.num_node, self.num_node))
536 87         for hop in valid_hop:
537 88             adjacency[self.hop_dis == hop] = 1 # Mark reachable
538     nodes at this hop
539 89
540 90             normalize_adjacency = np.zeros((self.num_node, self.
541     num_node))
542 91             if strategy == 'uniform': # Uniformly weight connections
543 92                 for i in range(self.num_node):
544 93                     sum_temp = np.sum(adjacency[i])
545 94                     if sum_temp > 0:
546 95                         normalize_adjacency[i] = adjacency[i] /
547     sum_temp
548 96             elif strategy == 'distance': # Weight by inverse distance (
549     closer nodes more important)
550 97                 for i in range(self.num_node):
551 98                     for j in range(self.num_node):
552 99                         if adjacency[i, j] > 0: # if connected
553 100                             normalize_adjacency[i,j] = 1/(self.hop_dis
554     [i,j] + 1e-6) # +1 to avoid 1/0 for self
555 101                             sum_temp = np.sum(normalize_adjacency[i])
556 102                             if sum_temp > 0:

```

```

557 103         normalize_adjacency[i] = normalize_adjacency[i]
558      / sum_temp
559 104      # Other strategies like spatial configuration partitioning
560      (ST-GCN paper)
561 105      # would result in multiple adjacency matrices here, stacked
562      into A.
563 106      # For simplicity, this example returns a single matrix.
564 107      # The actual ST-GCN uses multiple kernels (A_k).
565 108      # Here we return A as a (1, num_node, num_node) tensor for
566      compatibility
567 109      # with GCN_unit if it expects a kernel dimension.
568 110      return torch.from_numpy(normalize_adjacency).float().
569      unsqueeze(0)
570

```

gcn\_utils.py

```

572
573
574 8 import torch
575 9 import torch.nn as nn
576 10
577 11 class GCN_unit(nn.Module):
578 12     # Graph Convolutional Unit: applies graph convolution using
579     adjacency matrices.
580 13     # 'num_A_kernels' here refers to the number of different
581     adjacency matrices (A_k).
582 14     def __init__(self, in_channels, out_channels, num_A_kernels, #
583     num_A_kernels (spatial_kernel_size)
584 15         A_matrix_batch, adaptive=True, t_kernel_size=1,
585     t_stride=1,
586 16         t_padding=0, t_dilation=1, bias=True):
587 17         super().__init__()
588 18         self.num_A_kernels = num_A_kernels
589 19         # A_matrix_batch should be (num_A_kernels, num_nodes,
590     num_nodes)
591 20         assert A_matrix_batch.size(0) == self.num_A_kernels
592 21
593 22         # This conv layer learns weights for each kernel and input
594     channel
595 23         self.conv = nn.Conv2d(in_channels, out_channels * self.
596     num_A_kernels,
597 24         kernel_size=(t_kernel_size, 1), # (
598     Temporal, Spatial)
599 25         padding=(t_padding, 0),
600 26         stride=(t_stride, 1),
601 27         dilation=(t_dilation, 1), bias=bias)
602 28
603 29         if adaptive: # If A is learnable
604 30             self.A = nn.Parameter(A_matrix_batch.clone())
605 31         else: # If A is fixed
606 32             self.register_buffer('A', A_matrix_batch.clone())
607 33
608 34         self.bn = nn.BatchNorm2d(out_channels)
609 35         self.relu = nn.ReLU(inplace=True)
610 36
611 37     def forward(self, x, len_x=None): # x: (N, C_in, T, V)
612 38         x_conv = self.conv(x) # -> (N, C_out * num_A_kernels, T, V)
613 39         N, KC, T, V = x_conv.size() # N=batch, KC=C_out*
614     num_A_kernels, T=time, V=nodes
615 40         C_out = KC // self.num_A_kernels
616 41
617 42         # Reshape for einsum: (N, num_A_kernels, C_out, T, V)
618 43         x_r = x_conv.view(N, self.num_A_kernels, C_out, T, V)
619 44

```

```

620 45         # Graph convolution: Sum_k (X_k @ A_k)
621 46         # X_k is x_r[:,k,:,:,:] (N, C_out, T, V)
622 47         # A_k is self.A[k,:,:] (V, V) assuming A is (num_A_kernels,
623         V, V)
624 48         # einsum 'nkctv,kvw->nctw' means:
625 49         # for each n, c, t: out[n,c,t,w] = sum_k sum_v x_r[n,k,c,t,
626         v] * A[k,v,w]
627 50         x_agg = torch.einsum('nkctv,kvw->nctw', (x_r, self.A)).
628         contiguous() # -> (N,C_out,T,V)
629 51
630 52         return self.relu(self.bn(x_agg))
631 53
632 54 class STGCN_block(nn.Module):
633 55     # Spatio-Temporal GCN Block: Combines GCN_unit with a Temporal
634     Convolutional Network (TCN).
635 56     def __init__(self, in_channels, out_channels, kernel_size_st, #
636         (temporal_k, spatial_k_num_A)
637 57         A_matrix_batch, adaptive=True, stride=1, dropout
638         =0, residual=True):
639 58         super().__init__()
640 59         t_k, s_k_num_A = kernel_size_st[0], kernel_size_st[1] #
641         Temporal kernel, Spatial num_A_kernels
642 60
643 61         self.gcn = GCN_unit(in_channels, out_channels, s_k_num_A,
644         A_matrix_batch, adaptive=adaptive)
645 62
646 63         # Temporal Convolutional Network (TCN)
647 64         tcn_padding = ((t_k - 1) // 2, 0) # Pad temporally to keep
648         T same if stride=1
649 65         if t_k > 1: # Only apply TCN if temporal kernel size > 1
650 66             self.tcn = nn.Sequential(
651 67                 nn.BatchNorm2d(out_channels),
652 68                 nn.ReLU(inplace=True),
653 69                 nn.Conv2d(out_channels, out_channels, (t_k, 1), (
654         stride, 1), tcn_padding),
655 70                 nn.BatchNorm2d(out_channels),
656 71                 nn.Dropout(dropout, inplace=True)
657 72             )
658 73         else: # If t_k is 1, TCN is effectively an Identity
659         operation
660 74             self.tcn = nn.Identity()
661 75
662 76         # Residual connection
663 77         if not residual:
664 78             self.res_path = lambda x: 0 # No residual
665 79         elif (in_channels == out_channels) and (stride == 1):
666 80             self.res_path = nn.Identity() # Direct residual
667 81         else: # Need to project residual to match output shape
668 82             self.res_path = nn.Sequential(
669 83                 nn.Conv2d(in_channels, out_channels, kernel_size=1,
670         stride=(stride, 1)),
671 84                 nn.BatchNorm2d(out_channels)
672 85             )
673 86         self.relu_out = nn.ReLU(inplace=True)
674 87
675 88         def forward(self, x, len_x=None):
676 89             res = self.res_path(x)
677 90             x = self.gcn(x, len_x)
678 91             x = self.tcn(x)
679 92             return self.relu_out(x + res)
680 93
681 94 # Helper to build a chain of ST-GCN blocks
682 95 def get_stgcn_chain(in_dim, level_config_key, kernel_params_st, # (
683         temporal_k, spatial_k_num_A)
684 96         adj_matrix_batch, adaptive_adj):

```

```

685 97     # Defines layer configurations: [output_channels,
686     num_blocks_in_layer]
687 98     # Example configs, actual may vary based on UniSign
688 99     configs = {
689 100         'spatial': [[64,1], [128,1], [256,1]], # 3 layers, 1 block
690         each
691 101         'temporal': [[256,3]] # 1 layer, 3 blocks
692 102     }
693 103     block_config = configs.get(level_config_key)
694 104     if block_config is None:
695 105         raise NotImplementedError(f'`ST-GCN level config `{
696         level_config_key}` not found.`')
697 106
698 107     module_list = nn.Sequential()
699 108     current_channels = in_dim
700 109     for i, (out_ch, num_blocks_in_layer) in enumerate(block_config)
701     :
702 110         for j in range(num_blocks_in_layer):
703 111             # Stride might be 2 for some layers to downsample
704             spatially/temporally,
705 112             # but often 1 in SLT to preserve sequence length.
706             Assuming stride=1 here.
707 113             module_list.add_module(f'stgcn_{level_config_key}_L{i}
708             _B{j}',
709 114                 STGCN_block(current_channels, out_ch,
710                 kernel_params_st,
711 115                             adj_matrix_batch.clone(), adaptive_adj,
712                 stride=1))
713 116             current_channels = out_ch
714 117     return module_list, block_config[-1][0] # Return chain and its
715     output dimension
716

```

## STGCN\_Block.py

```

718
720 292 # Within the forward pass of a model like Uni_Sign:
721 293 # Assume 'src_input[part]' contains (Batch, Time, Num_Joints,
722     Coords)
723 294 # 'self.proj_linear[part]' projects Coords to initial feature
724     dimension
725 295 # 'self.gcn_modules[part]' is the spatial ST-GCN chain for that
726     part
727 296 # 'self.fusion_gcn_modules[part]' is the temporal ST-GCN chain
728
729 297 # body_feat_spatial_output = None # Initialize
730 298 # all_part_features = {}
731 300
732 301 # for p in ['body', 'left', 'right', 'face_all']: # Example order
733 302 #     current_part_input = src_input[p]
734 303 #     compute_dtype = self.proj_linear[p].weight.dtype # For mixed
735     precision
736 304 #     x_projected = self.proj_linear[p](current_part_input.to(dtype=
737     compute_dtype))
738 305 #     x_projected = x_projected.permute(0,3,1,2) # (N, C_in, T, V)
739 306 #     gcn_out_spatial = self.gcn_modules[p](x_projected) # Output
740     from spatial GCN layers
741 307
742 308 #     # Store spatial GCN output of 'body' part for use by subsequent
743     parts.
744 309 #     if p == 'body':
745 310 #         body_feat_spatial_output = gcn_out_spatial # (N, C_body, T,
746     V_body)
747 311

```



```

748 312 # # If current part 'p' is not 'body' and '
749      body_feat_spatial_output' is available:
750 313 # if p != 'body' and body_feat_spatial_output is not None:
751 314 # # Example: Add feature from a relevant body joint (e.g.,
752      shoulder for hand, neck for face)
753 315 # # This requires careful selection of indices from
754      body_feat_spatial_output
755 316 # # and broadcasting/repeating it to match gcn_out_spatial's
756      V dimension.
757 317 # # The original code snippet implies specific joint indices
758      for this addition.
759 318 # # For instance, if body_feat_spatial_output[... ,
760      relevant_joint_idx] is (N, C_body, T)
761 319 # # it needs to be expanded to (N, C_body, T, 1) and then
762      added.
763 320 # # The '.detach()' is used to treat body features as fixed
764      context.
765 321 # # Note: Ensure channel dimensions C_body and C_part match
766      or project one to match other.
767 322 # # This is a simplified example; actual implementation might
768      involve projections.
769 323 # if p == 'left' and body_feat_spatial_output.shape[-1] >=
770      2: # e.g. left shoulder proxy
771 324 # # Assuming C_body == C_part for simplicity in this
772      example.
773 325 # context_feat = body_feat_spatial_output[... , -2].
774      unsqueeze(-1).detach()
775 326 # if context_feat.shape[1] == gcn_out_spatial.shape[1]:
776      # Check channel match
777 327 gcn_out_spatial = gcn_out_spatial + context_feat
778 328 # elif p == 'right' and body_feat_spatial_output.shape[-1]
779      >= 1: # e.g. right shoulder proxy
780 329 # context_feat = body_feat_spatial_output[... , -1].
781      unsqueeze(-1).detach()
782 330 # if context_feat.shape[1] == gcn_out_spatial.shape[1]:
783 331 gcn_out_spatial = gcn_out_spatial + context_feat
784 332 # elif p == 'face_all' and body_feat_spatial_output.shape
785      [-1] >= 1: # e.g. neck joint proxy
786 333 # context_feat = body_feat_spatial_output[... , 0].
787      unsqueeze(-1).detach()
788 334 # if context_feat.shape[1] == gcn_out_spatial.shape[1]:
789      gcn_out_spatial = gcn_out_spatial + context_feat
790 336
791 337 # # Apply temporal fusion GCN layers.
792 338 # gcn_out_final_part = self.fusion_gcn_modules[p](gcn_out_spatial
793      )
794 339 # all_part_features[p] = gcn_out_final_part
795 340 # End of loop over parts
796

```

## models.py (Residual Connections)

```

798
799
800 53 import torch
801 54 import torch.nn as nn
802 55 from geoopt import PoincareBall # Assuming PoincareBall is imported
803 56
804 57 class HyperbolicProjection(nn.Module):
805 58     # Projects Euclidean input features to the Poincare ball
806     manifold.
807 59     def __init__(self, dim_in: int, dim_out: int, manifold:
808         PoincareBall):
809 60         super().__init__()

```

```

810 61     self.manifold = manifold # PoincareBall instance (contains
811    curvature c)
812 62     self.proj = nn.Linear(dim_in, dim_out, bias=True) #
813    Projects to tangent space dim
814 63     # Learnable log_scale for adaptive scaling in tangent space
815    .
816 64     # Initialised to 0.0, so exp(log_scale) is initially 1.0.
817 65     self.log_scale = nn.Parameter(torch.tensor(0.0, dtype=torch
818    .float32))
819 66
820 67     def forward(self, x: torch.Tensor) -> torch.Tensor: # x is
821    Euclidean input
822 68     # Ensure computations use the same dtype as linear layer
823    weights, good for mixed precision.
824 69     weight_dtype = self.proj.weight.dtype
825 70
826 71     # 1. Linearly project Euclidean input x to the desired
827    hyperbolic dimension.
828 72     tangent_vec_unscaled = self.proj(x.to(weight_dtype))
829 73
830 74     # 2. Apply learnable scale in tangent space. exp() ensures
831    scale is positive.
832 75     scale = self.log_scale.to(weight_dtype).exp()
833 76     tangent_vec_scaled = tangent_vec_unscaled * scale
834 77
835 78     # 3. Map to Poincare ball using expmap0 (exponential map at
836    the origin).
837 79     # Ensure tangent_vec is float32 for geoopt numerical
838    stability.
839 80     # 'project=True' ensures the output point is strictly
840    inside the ball boundary.
841 81     # For tangent vector clipping before expmap0 (see Sec 3.4
842    main paper & App H):
843 82     # max_norm_val = (1. / (self.manifold.c.sqrt() + 1e-5)) - 1
844    e-5 # Max norm for tangent vector
845 83     # norm = tangent_vec_scaled.norm(dim=-1, keepdim=True).
846    clamp_min(1e-8)
847 84     # clip_coeff = torch.where(norm > max_norm_val,
848    max_norm_val/norm, torch.ones_like(norm))
849 85     # tangent_vec_clipped = tangent_vec_scaled * clip_coeff
850 86     # hyperbolic_point = self.manifold.expmap0(
851    tangent_vec_clipped.float(), project=True)
852 87     # OR, if clipping is handled inside manifold.expmap0 with
853    project=True based on its internal logic
854 88     # or by the tangent vector clipping strategy mentioned in
855    Sec H.
856 89     hyperbolic_point = self.manifold.expmap0(tangent_vec_scaled
857    .float(), project=True)
858 90
859 91     # Cast back to original input dtype if necessary (e.g., for
860    mixed precision consistency)
861 92     return hyperbolic_point.to(x.dtype)
862

```

models.py (HyperbolicProjection)

```

864
865 117 import torch
866 118 from geoopt import PoincareBall # Assuming PoincareBall is imported
867
868 119
869 120 def weighted_frechet_mean(points: torch.Tensor, # (NumPoints,
870    BatchSize, HyperbolicDim)
871    weights: torch.Tensor, # (NumPoints,
872    BatchSize) or (NumPoints,)

```

```

873 122 manifold: PoincareBall,
874 123 max_iter: int = 50, # Max iterations for
875 convergence
876 124 tol: float = 1e-5, # Tolerance for
877 convergence
878 125 eta: float = 1.0): # Step size for update
879 (Algorithm 1 uses 1.0)
880 126 # Ensure weights are correctly broadcastable for (points *
881 weights)
882 127 if weights.ndim == 1: # (NumPoints,) -> (NumPoints, 1, 1) for
883 broadcasting over B, H
884 128 norm_weights = weights.unsqueeze(-1).unsqueeze(-1)
885 129 elif weights.ndim == 2: # (NumPoints, BatchSize) -> (NumPoints,
886 BatchSize, 1) for H
887 130 norm_weights = weights.unsqueeze(-1)
888 131 else: # Assumed (NumPoints, BatchSize, 1) or similar, ensure it
889 broadcasts with points
890 132 norm_weights = weights
891 133
892 134 # Normalize weights for each batch item (if weights are per
893 batch item)
894 135 # If weights are global (NumPoints,), sum over dim 0.
895 136 # If per batch (NumPoints, BatchSize), sum over dim 0 for each
896 batch.
897 137 if weights.ndim == 2: # (NumPoints, BatchSize)
898 138 sum_weights = norm_weights.sum(dim=0, keepdim=True)
899 139 else: # Global weights or already shaped (NumPoints, B, 1) from
900 (N,B) input
901 140 sum_weights = norm_weights.sum(dim=0, keepdim=True) # sum
902 over N_pts
903 141 norm_weights = norm_weights / (sum_weights + 1e-8)
904 142
905 143
906 144 # Initialize mean estimate (e.g., with the first point, or
907 average in tangent space at origin)
908 145 # Using first point as init: mu (BatchSize, HyperbolicDim)
909 146 mu = points[0].clone().detach() # Detach to avoid in-place
910 modification issues if points[0] is used elsewhere
911 147
912 148 for iteration in range(max_iter):
913 149 mu_old = mu.clone()
914 150
915 151 # 1. LogMap: Project all points to tangent space at the
916 current mean 'mu'.
917 152 # 'mu' is (B,H), 'points' is (N_pts, B, H). Need 'mu' to be
918 (1, B, H) for logmap.
919 153 tangent_vecs_at_mu = manifold.logmap(mu.unsqueeze(0),
920 points) # (N_pts, B, H)
921 154
922 155 # 2. Weighted Average of tangent vectors (this is a
923 Euclidean operation).
924 156 # 'norm_weights' is (N_pts, B, 1), 'tangent_vecs_at_mu' is
925 (N_pts, B, H)
926 157 avg_tangent_vec_at_mu = (norm_weights * tangent_vecs_at_mu)
927 .sum(dim=0) # (B,H)
928 158
929 159 # 3. ExpMap: Map the average tangent vector back to the
930 manifold from 'mu'.
931 160 # The step size 'eta' scales the update in the tangent
932 space.
933 161 mu_next = manifold.expmap(mu, eta * avg_tangent_vec_at_mu,
934 project=True)
935 162
936 163 # 4. Check Convergence: if distance between new and old
937 mean is small enough.

```

```

938 164     # This should be checked per batch element if mu is batched
939 .
940 165     dist_change = manifold.dist(mu_next, mu_old) # (B,) if
941 batched
942 166     if (dist_change < tol).all():
943 167         mu = mu_next # One final update
944 168         break
945 169     mu = mu_next
946 170     if iteration == max_iter - 1:
947 171         # print(f'Frechet mean did not converge in {max_iter}
948 iterations.')
949 172         pass # Or log a warning
950 173
951 174     return mu
952

```

models.py (Weighted Fréchet Mean)

```

954
955 380 # Inside a model's forward pass, e.g., Uni_Sign.forward
956 381 # Assumes 'self.hyp_proj_text' is an instance of
957 382 HyperbolicProjection.
958 383 # 'txt_token_embeddings_euclidean': Euclidean token embeddings (
959 Batch, SeqLen, EmbDim).
960 384 # 'txt_padding_mask_bool': Boolean padding mask (Batch, SeqLen),
961 True for valid tokens.
962 385
963 386 # if self.text_cmp_mode == 'pooled': # Or similar conditional
964 logic
965 387 # 1. Masked mean pooling to get a single Euclidean sentence
966 vector.
967 388 # Expand mask for broadcasting: (Batch, SeqLen) -> (Batch,
968 SeqLen, 1)
969 389 mask_expanded = txt_padding_mask_bool.unsqueeze(-1).float()
970 390
971 391 # Apply mask (zero out padded tokens)
972 392 masked_txt_embeddings = txt_token_embeddings_euclidean *
973 mask_expanded
974 393
975 394 # Sum valid token embeddings
976 395 summed_txt_embeddings = masked_txt_embeddings.sum(dim=1) # (Batch,
977 EmbDim)
978 396
979 397 # Count valid tokens per sentence, ensure at least 1 to avoid
980 div by zero.
981 398 num_valid_tokens = mask_expanded.sum(dim=1).clamp_min(1.0) # (Batch
982 , 1)
983 399
984 400 # Compute mean
985 401 euclidean_sentence_embedding = summed_txt_embeddings /
986 num_valid_tokens # (Batch, EmbDim)
987 402
988 403 # 2. Project the mean Euclidean sentence embedding to
989 hyperbolic space.
990 404 # Ensure input to projection is float32 for geoopt stability.
991 405 hyperbolic_text_embedding_pooled = self.hyp_proj_text(
992 euclidean_sentence_embedding.float())
993 406
994 407 # This 'hyperbolic_text_embedding_pooled' is then contrasted
995 with
996 # the global pose embedding 'mu_pose'.
997
998

```

models.py (Global Text Embedding - Pooled)

```

1000
1002 407 import torch
1003 408 import torch.nn as nn
1004 409 import torch.nn.functional as F
1005 410 from geoopt import PoincareBall # Assuming PoincareBall is imported
1006 411
1007 412 # Inside a model's forward pass, e.g., Uni_Sign.forward
1008 413 # Assumes 'self.manifold' is a PoincareBall instance.
1009 414 # Assumes 'self.hyp_proj_text' projects Euclidean text tokens to
1010      hyperbolic.
1011 415 # Assumes 'self.hyp_attn_W_key', 'self.hyp_attn_b_key' are nn.
1012      Parameters for Mobius transform.
1013 416 # 'hyperbolic_pose_part_queries': (Batch, NumParts, HyperbolicDim)
1014      - e.g., from h_p
1015 417 # 'euclidean_text_tokens': (Batch, SeqLen_Text, EmbDim_Euclidean)
1016 418 # 'text_padding_mask_bool': (Batch, SeqLen_Text) - True for valid
1017      tokens
1018 419
1019 420 # if self.text_cmp_mode == 'token': # Or similar conditional
1020      logic
1021 421
1022 422 # 1. Project Euclidean text tokens to hyperbolic space (to
1023      serve as Keys & Values).
1024 423 # Ensure input is float32 for geoopt.
1025 424 hyperbolic_text_tokens_kv = self.hyp_proj_text(
1026      euclidean_text_tokens.float())
1027 425 # Output: (Batch, SeqLen_Text, HyperbolicDim)
1028 426
1029 427 B, T_text, H_hyp = hyperbolic_text_tokens_kv.shape
1030 428 NumParts = hyperbolic_pose_part_queries.shape[1]
1031 429
1032 430 # 2. Hyperbolic Cross-Attention:
1033 431 # Queries: 'hyperbolic_pose_part_queries' (B, NumParts, H_hyp)
1034 432 # Keys/Values: 'hyperbolic_text_tokens_kv' (B, T_text, H_hyp)
1035 433
1036 434 # Expand queries and keys/values for pairwise operations.
1037 435 # Queries: (B, NumParts, 1, H_hyp) -> for broadcasting with
1038      T_text
1039 436 expanded_queries = hyperbolic_pose_part_queries.unsqueeze(2)
1040 437 # Keys/Values: (B, 1, T_text, H_hyp) -> for broadcasting with
1041      NumParts
1042 438 expanded_keys_values = hyperbolic_text_tokens_kv.unsqueeze(1)
1043 439
1044 440 # Key Transformation: Apply learnable Mobius transformations to
1045      text token keys.
1046 441 # self.hyp_attn_W_key should be (H_hyp, H_hyp), self.
1047      hyp_attn_b_key (H_hyp,)
1048 442 # Geoopt's mobius_matvec and mobius_add handle batching and
1049      manifold operations.
1050 443 transformed_text_keys = self.manifold.mobius_matvec(
1051      self.hyp_attn_W_key.float().to(expanded_keys_values.device),
1052      expanded_keys_values
1053 444 )
1054 445 transformed_text_keys = self.manifold.mobius_add(
1055      transformed_text_keys, self.hyp_attn_b_key.float().to(
1056      expanded_keys_values.device)
1057 446 )
1058 447 # 'transformed_text_keys' is (B, NumParts_or_1, T_text, H_hyp)
1059 448 # depending on broadcasting, ensure it's (B, NumParts, T_text,
1060      H_hyp) if W,b are shared
1061 449 # or (B, 1, T_text, H_hyp) if W,b are not part-specific. Assume
1062      shared for now.
1063 450 # If W,b are global, expanded_keys_values is fine: (B,1,T_text,
1064      H_hyp)
1065 451 # transformed_text_keys will be (B,1,T_text,H_hyp)

```

```

1066 454
1067 455     # Attention Scores: Negative geodesic distance between pose
1068     queries and transformed keys.
1069 456     # 'expanded_queries' (B, NumParts, 1, H_hyp)
1070 457     # 'transformed_text_keys' (B, 1, T_text, H_hyp)
1071 458     # Resulting 'attn_logits' will be (B, NumParts, T_text)
1072 459 attn_logits = -self.manifold.dist(expanded_queries,
1073     transformed_text_keys)
1074 460
1075 461     # Mask scores for padded text tokens.
1076 462     # 'text_padding_mask_bool' (B, T_text) -> (B, 1, T_text) for
1077     broadcasting
1078 463 attn_mask = text_padding_mask_bool.unsqueeze(1).expand(-1, NumParts
1079     , -1) # Ensure mask aligns with attn_logits
1080 464 attn_logits = attn_logits.masked_fill(~attn_mask, -torch.inf) #
1081     Fill padded with -inf
1082 465
1083 466     # Attention Weights from softmax (e.g. self.attention_temp is a
1084     learnable scalar for tau_attn)
1085 467     # tau_attn_param = torch.sigmoid(self.attention_temp_logit) *
1086     1.99 + 0.01 # Or fixed
1087 468 tau_attn_param = 0.1 # Example fixed temperature for attention
1088     softmax
1089 469 attn_weights = F.softmax(attn_logits / tau_attn_param, dim=-1) # (B
1090     , NumParts, T_text)
1091 470
1092 471     # Contextual Text Embedding: Hyperbolic weighted midpoint of
1093     original hyperbolic text tokens (Values),
1094 472     # using the computed attention weights.
1095 473     # 'expanded_keys_values' (Values): (B, 1, T_text, H_hyp)
1096 474     # 'attn_weights': (B, NumParts, T_text)
1097 475     # 'weighted_midpoint' needs weights shaped (B, NumParts, T_text
1098     ) and points (B, NumParts, T_text, H_hyp)
1099 476     # So, expand 'expanded_keys_values' to align with NumParts
1100     dimension of weights
1101 477 aligned_values = expanded_keys_values.expand(-1, NumParts, -1, -1)
1102     # (B, NumParts, T_text, H_hyp)
1103 478
1104 479 contextual_text_embeddings_for_parts = self.manifold.
1105     weighted_midpoint(
1106 480         points=aligned_values,
1107 481         weights=attn_weights,
1108 482         reducedim=[2], # Aggregate over the T_text dimension
1109 483         keepdim=False
1110 484     ) # Output: (B, NumParts, H_hyp)
1111 485
1112 486     # These 'contextual_text_embeddings_for_parts' are then used in
1113     the contrastive loss
1114 487     # against the 'hyperbolic_pose_part_queries'.
1115

```

models.py (Hyperbolic Attention - Token)

```

1117
1118 75 import torch
1119 76 import torch.nn as nn
1120 77 from geoopt import PoincareBall # Assuming PoincareBall is imported
1121 78 from typing import Dict # For type hinting
1122 79
1123 80 class HyperbolicContrastiveLoss(nn.Module):
1124 81     # Computes InfoNCE-style contrastive loss in hyperbolic space.
1125 82     def __init__(self, manifold: PoincareBall,
1126 83         initial_temp_logit: float = 0.0, # Logit for
1127         temperature, so temp ~ sigmoid(0) = 0.5
1128

```

```

1129 84         initial_margin: float = 0.1, # Additive margin for
1130      negatives
1131 85         label_smoothing: float = 0.0): # Label smoothing
1132  for CE loss part
1133 86         super().__init__()
1134 87         self.manifold = manifold
1135 88         # Learnable temperature (tau). Sigmoid maps logit to (0,1),
1136      then scale.
1137 89         # init to 0.0 => temp = (sigmoid(0)*1.99+0.01) =
1138      0.5*1.99+0.01 ~ 1.0
1139 90         self.temp_logit = nn.Parameter(torch.tensor(
1140      initial_temp_logit, dtype=torch.float32))
1141 91         # Learnable additive margin (m) for negative pairs.
1142 92         # Use softplus to ensure margin >= 0, or just a direct
1143      parameter if careful with init.
1144 93         self.margin_val = nn.Parameter(torch.tensor(initial_margin,
1145      dtype=torch.float32))
1146 94
1147 95         self.loss_fct = nn.CrossEntropyLoss(label_smoothing=
1148      label_smoothing,
1149 96                                         ignore_index=-100) #
1150      For compatibility
1151 97
1152 98         def forward(self, anchors: torch.Tensor, # (BatchSize,
1153      HyperbolicDim), e.g., pose embeddings
1154 99             targets: torch.Tensor, # (BatchSize, HyperbolicDim)
1155      , e.g., text embeddings
1156 100             other_negatives: torch.Tensor = None # (BatchSize,
1157      NumNeg, HyperbolicDim), optional
1158 101             ) -> Dict[str, torch.Tensor]:
1159 102             # Assumes anchors[i] and targets[i] form a positive pair.
1160 103             # All other targets[j] (j!=i) are in-batch negatives for
1161      anchors[i].
1162 104
1163 105             bsz = anchors.shape[0]
1164 106             if bsz == 0: # Handle empty batch gracefully
1165 107                 current_temp_val = torch.sigmoid(self.temp_logit.detach
1166      ()) * 1.99 + 0.01
1167 108                 current_margin_val = self.margin_val.detach().clamp_min
1168      (0.0)
1169 109             return {
1170 110                 'loss': torch.tensor(0.0, device=anchors.device,
1171      requires_grad=True),
1172 111                 'sim_mean_pos': torch.tensor(0.0, device=anchors.
1173      device),
1174 112                 'margin_used': current_margin_val,
1175 113                 'temp_used': current_temp_val
1176 114             }
1177 115
1178 116             # 1. Compute all-pairs geodesic distances d(anchors_i,
1179      targets_j).
1180 117             # anchors.unsqueeze(1): (B, 1, H_dim)
1181 118             # targets.unsqueeze(0): (1, B, H_dim)
1182 119             dist_matrix = self.manifold.dist(anchors.unsqueeze(1),
1183      targets.unsqueeze(0)) # (B, B)
1184 120
1185 121             # Similarity = negative distance. Higher is better.
1186 122             sim_matrix = -dist_matrix # (B, B)
1187 123
1188 124             # 2. Calculate learnable temperature (tau).
1189 125             tau = torch.sigmoid(self.temp_logit) * 1.99 + 0.01 # Temp
1190      in [0.01, 2.0]
1191 126             logits = sim_matrix / tau # Scale similarities to get
1192      logits
1193 127

```



```

1194 128     # 3. Apply additive margin to negative pairs (off-diagonal
1195     elements).
1196 129     # Margin should make negatives less likely (i.e., decrease
1197     their logit).
1198 130     # Ensure margin is non-negative for its intended effect.
1199 131     current_margin = self.margin_val.clamp_min(0.0) # m >= 0
1200 132
1201 133     # Create a mask for positive pairs (diagonal)
1202 134     positive_mask = torch.eye(bsz, device=logits.device, dtype=
1203     torch.bool)
1204 135     # Subtract margin from negative pairs (logits for j != i)
1205 136     # This is equivalent to adding 'm * I(i!=j)' to -d/tau if m
1206     was defined for distances.
1207 137     # For similarities, we subtract: sim_neg' = sim_neg -
1208     margin
1209 138     logits_with_margin = torch.where(positive_mask, logits,
1210     logits - current_margin)
1211 139
1212 140     # (Optional: if 'other_negatives' are provided, compute
1213     their logits too and concat)
1214 141     # This part is not in Eq. (5) of main paper, so we'll stick
1215     to in-batch negatives.
1216 142
1217 143     # 4. Define targets for CrossEntropyLoss: target for
1218     anchors_i is targets_i (index i).
1219 144     # These are the ground truth labels for the rows of the
1220     logit matrix.
1221 145     ce_targets = torch.arange(bsz, device=anchors.device) # [0,
1222     1, ..., B-1]
1223 146
1224 147     # 5. Compute InfoNCE loss using CrossEntropyLoss on the
1225     logits.
1226 148     loss = self.loss_fct(logits_with_margin, ce_targets)
1227 149
1228 150     # For logging purposes:
1229 151     sim_mean_pos = sim_matrix.diag().mean().detach()
1230 152     return {
1231 153         'loss': loss,
1232 154         'sim_mean_pos': sim_mean_pos,
1233 155         'margin_used': current_margin.detach(),
1234 156         'temp_used': tau.detach()
1235 157     }
1236

```

models.py (HyperbolicContrastiveLoss)

```

1238
1240 209 import torch
1241 210 import torch.nn as nn
1242 211 from geoopt import PoincareBall
1243 212
1244 213 # Inside a model's __init__ method (e.g., Uni_Sign.__init__):
1245 214 # 'args.hyp_dim' would be the desired hyperbolic embedding
1246     dimension.
1247 215 # 'args.init_c' would be the initial value for the curvature
1248     magnitude c.
1249 216
1250 217 # Assume args object is available, e.g. passed to __init__
1251 218 # self.hyp_dim = args.hyp_dim # Hyperbolic embedding dimension.
1252 219
1253 220 # Initialize the PoincareBall manifold from geoopt.
1254 221 # 'c': initial curvature magnitude. Must be positive.
1255 222 # 'learnable=True': This makes 'manifold.c' an nn.Parameter,
1256     allowing it to be

```

```

1257 223 #             updated by the optimizer based on the loss
1258         gradient w.r.t. c.
1259 224 # 'geoopt' internally handles ensuring c remains positive during
1260         optimization,
1261 225 # often by parameterizing c = softplus(c_raw) or similar if c is
1262         directly optimised.
1263 226 # The 'learnable=True' flag in geoopt typically handles this more
1264         directly.
1265 227
1266 228 # config_init_c = args.init_c # Example initial curvature from args
1267 229 # self.manifold = PoincareBall(c=config_init_c, learnable=True)
1268 230
1269 231 # All subsequent hyperbolic layers (e.g., HyperbolicProjection,
1270         HyperbolicContrastiveLoss)
1271 232 # will receive this 'self.manifold' instance and use its current (
1272         possibly learned)
1273 233 # curvature 'self.manifold.c'.
1274

```

#### models.py (Manifold Initialization)

```

1276
1277 488 import torch
1278 489 import torch.nn as nn
1279 490
1280 491 # Inside a model's forward pass (e.g., Uni_Sign.forward):
1281 492 # Assume 'ce_loss' (CrossEntropy loss) and 'hyperbolic_reg_loss'
1282         are computed.
1283 493 # Assume 'self.args.alpha_init' is the initial base value for alpha
1284         (e.g., 0.7).
1285 494 # Assume 'self.global_step' (current training step) and 'self.
1286         total_training_steps' are available.
1287 495 # Assume 'self.loss_alpha_logit' is an nn.Parameter (a learnable
1288         scalar).
1289 496
1290 497 # 1. Calculate training progress (ratio from 0.0 to 1.0).
1291 498 # Ensure total_training_steps is not zero to prevent division by
1292         zero.
1293 499 # if self.trainer.max_steps > 0: # Example if using PyTorch
1294         Lightning
1295 500 #     progress_ratio = self.global_step / self.trainer.max_steps
1296 501 # else:
1297 502 #     progress_ratio = 0.0
1298 503 # Placeholder for actual progress calculation logic:
1299 504 progress_ratio = 0.5 # Example: self.global_step / self.
1300         total_training_steps
1301 505
1302 506 # 2. Base alpha component: starts at 'self.args.alpha_init' and
1303         linearly increases by up to 0.1.
1304 507 # This gives more weight to CE loss as training progresses.
1305 508 alpha_init_val = 0.7 # Example: self.args.alpha_init
1306 509 alpha_base_scheduled = alpha_init_val + (0.1 * progress_ratio)
1307 510
1308 511 # 3. Learnable alpha adjustment: 'self.loss_alpha_logit' is an nn.
1309         Parameter.
1310 512 # Sigmoid maps the logit to (0,1), then scaled by 0.2, providing an
1311         adjustment in [0, 0.2].
1312 513 # This allows the model to learn a small deviation from the
1313         scheduled base alpha.
1314 514 # loss_alpha_logit = self.loss_alpha_logit # Assuming it's an nn.
1315         Parameter
1316 515 loss_alpha_logit = torch.tensor(0.0) # Placeholder if not defined
1317         in this snippet context
1318 516 alpha_learned_adjustment = torch.sigmoid(loss_alpha_logit) * 0.2
1319

```

```

1320 517
1321 518 # 4. Combine components and clamp to ensure alpha is within a
1322     sensible range [0.1, 1.0].
1323 519 # An alpha of 1.0 means only CE loss, 0.0 means only hyperbolic
1324     loss.
1325 520 # Clamping prevents extreme values.
1326 521 current_alpha_scalar = (alpha_base_scheduled +
1327     alpha_learned_adjustment).clamp(0.1, 1.0)
1328 522
1329 523 # 5. Compute the final weighted total loss.
1330 524 # Ensure losses are float for stable combination, especially under
1331     mixed precision.
1332 525 # ce_loss and hyperbolic_reg_loss are assumed to be defined torch
1333     tensors.
1334 526 # total_combined_loss = current_alpha_scalar * ce_loss.float() + \
1335 527 #     (1.0 - current_alpha_scalar) *
1336     hyperbolic_reg_loss.float()
1337 528
1338 529 # Log current_alpha_scalar for monitoring if desired.
1339 530 # self.log('alpha_loss_blend', current_alpha_scalar)
1340

```

---

models.py (Dynamic Alpha Calculation)

## References

- [1] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, and Richard Bowden. Sign language transformers: Joint end-to-end sign language recognition and translation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10023–10033, 2020.
- [2] Zhigang Chen, Benjia Zhou, Yiqing Huang, Jun Wan, Yibo Hu, Hailin Shi, Yanyan Liang, Zhen Lei, and Du Zhang. C<sup>2</sup>rl: Content and context representation learning for gloss-free sign language translation and retrieval. 2024.
- [3] Zhigang Chen, Benjia Zhou, Jun Li, Jun Wan, Zhen Lei, Ning Jiang, Quan Lu, and Guoqing Zhao. Factorized learning assisted with large language model for gloss-free sign language translation. pages 7071–7081, 2024.
- [4] Amanda Duarte, Shruti Palaskar, Lucas Ventura, Deepti Ghadiyaram, Kenneth DeHaan, Florian Metze, Jordi Torres, and Xavier Giro-i Nieto. How2sign: a large-scale multimodal dataset for continuous american sign language. In *CVPR*, pages 2735–2744, 2021.
- [5] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. *Advances in neural information processing systems*, 31, 2018.
- [6] Jia Gong, Lin Geng Foo, Yixuan He, Hossein Rahmani, and Jun Liu. Llms are good sign language translators. In *CVPR*, pages 18362–18372, 2024.
- [7] Tao Jiang, Peng Lu, Li Zhang, Ning Ma, Rui Han, Chengqi Lyu, Yining Li, and Kai Chen. RTMPose: Real-time multi-person pose estimation based on mmpose. 2023.
- [8] Sheng Jin, Lumin Xu, Jin Xu, Can Wang, Wentao Liu, Chen Qian, Wanli Ouyang, and Ping Luo. Whole-body human pose estimation in the wild. In *European Conference on Computer Vision*, pages 196–214. Springer, 2020.
- [9] Max Kochurov, Rasul Karimov, and Serge Kozlukov. Geoopt: Riemannian optimization in pytorch, 2020.
- [10] Zecheng Li, Wengang Zhou, Weichao Zhao, Kepeng Wu, Hezhen Hu, and Houqiang Li. Uni-sign: Toward unified sign language understanding at scale. *arXiv preprint arXiv:2501.15187*, 2025.
- [11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [12] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.
- [13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. 2019.
- [14] Garrett Tanzer and Biao Zhang. Youtube-sl-25: A large-scale, open-domain multilingual sign language parallel corpus. 2024.
- [15] Dave Uthus, Garrett Tanzer, and Manfred Georg. Youtube-asl: A large-scale, open-domain american sign language-english parallel corpus. 2024.
- [16] Ryan Wong, Necati Cihan Camgoz, and Richard Bowden. Sign2GPT: Leveraging large language models for gloss-free sign language translation. In *ICLR*, 2024.
- [17] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.
- [18] Aoxiong Yin, Tianyun Zhong, Li Tang, Weike Jin, Tao Jin, and Zhou Zhao. Gloss attention for gloss-free sign language translation. In *ICCV*, pages 2551–2562, 2023.
- [19] Benjia Zhou, Zhigang Chen, Albert Clapés, Jun Wan, Yanyan Liang, Sergio Escalera, Zhen Lei, and Du Zhang. Gloss-free sign language translation: Improving from visual-language pretraining. In *ICCV*, pages 20871–20881, 2023.
- [20] Wengang Zhou, Weichao Zhao, Hezhen Hu, Zecheng Li, and Houqiang Li. Scaling up multi-modal pre-training for sign language understanding. 2024.