Tetrahedron-Tetrahedron Intersection and Volume Computation Using Neural Networks

Supplementary Materials

Erendiro Pedro[1], Hermann Meißenhelter[2] and Gabriel Zachmann[2]

[1]Institute of Engineering, Polytechnic of Porto, Portugal
[2]University of Bremen, Germany

1. Artifacts

The following artifacts are provided to ensure the reproducibility of our results and to facilitate further research:

- **ML Pipeline & Training Artifacts:** The complete source code for the *TetrahedronPairNet* architecture, including training scripts, hyperparameter configurations, and pre-trained model weights.
  [Source Code and Models]
- **Dataset Generator:** The geometric sampling engine used to generate the stratified dataset of tetrahedron pairs across all topological configurations, including the constraint-based contact samplers.
  [Dataset Generation Tools]
- **Interactive Demo:** An integrated demonstration environment showcasing the model's inference capabilities on real-time scenarios, including both isolated tetrahedron-tetrahedron pairs and full-scale mesh-mesh intersection tests.
  [Interactive Demonstration]
- **Research Documentation:** A comprehensive knowledge base containing research notes, literature reviews, and extended documentation on related tetrahedral collision topics. [Extended Research and Resources]

2. Further Related Research

State of the art point cloud processing architectures often focus on capturing local geometric information through convolutions [WSL*19], hierarchical sampling with linear transformations [QYSG17], or attention mechanisms [YYJ22], as well as other techniques. A detailed taxonomy of deep learning methods for point cloud processing can be found in [ZWT*23, DSX*23, GWH*20]. These methods excel on complex, irregular point clouds with thousands of points and rich local structure. In contrast, we adopt a simpler MLP-based architecture inspired by PointMLP [MQY*22] for tetrahedron pairs. We have chosen this approach for three reasons: (i) tetrahedron pairs possess a fixed, minimal structure (only eight vertices), rendering complex hierarchical feature extraction unnecessary; (ii) simpler models often offer superior inference speed and memory efficiency, which is critical for our throughput goals [MQY*22]; and (iii) PointNet-based architectures remain the industry standard for deployment due to their optimal balance of expressiveness and efficiency [MQY*22].

In our case, rather than treating collision detection and volume computation as separate algorithmic tasks, we view them as properties implicitly encoded within the joint coordinate manifold of two simplices. The success of our approach strongly suggests that both topology and geometric measure are accessible via a unified feature space [QA24]. In this context, volume serves not as a singular exception, but as a representative benchmark for extracting complex geometric attributes from vertex positions.

3. Generation Strategies

To provide geometric intuition, consider a tetrahedron inscribed in a unit cube. While the largest possible inscribed tetrahedron can reach a volume of $1/3 \approx 0.3333$, a randomly sampled tetrahedron is typically much smaller. Specifically, the expected volume of a tetrahedron sampled uniformly from the unit cube is approximately 0.0138 [Zin03]. Given these small volumes, two i.i.d. sampled tetrahedra are disjoint with high probability. Under this distribution, the intersection volume of overlapping pairs is typically several orders of magnitude smaller than the unit scale. With this in mind, we detail below the algorithms used to synthesize samples for the geometric cases discussed in this work.

No Intersection Algorithm 1 details the rejection sampling strategy used to generate disjoint pairs. Given the low expected volume of random tetrahedra, this approach is highly efficient.

Algorithm 1 Constructing non-intersecting tetrahedra via rejection.

1: **repeat**
2: Sample tetrahedra $\mathcal{T}_1, \mathcal{T}_2 \sim \mathcal{U}([0,1]^3)$
3: $overlap \leftarrow$ `CGAL::do_intersect`$(\mathcal{T}_1, \mathcal{T}_2)$
4: **until** $!overlap$
5: **return** $(\mathcal{T}_1, \mathcal{T}_2)$

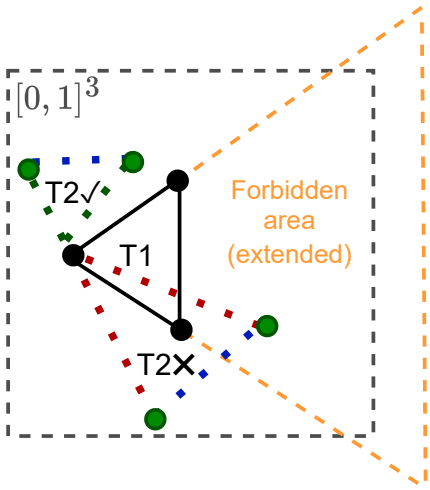

Figure 1: ***Vertex–Vertex construction (2D analogue).*** *The candidate simplex shares a single vertex with T_1. The remaining vertices are sampled in $[0,1]^3$, while configurations whose opposite face enters the forbidden region induced by extending the edges incident to the shared vertex are rejected. Sampling is repeated until a valid simplex is obtained. In the full 3D setting, this yields two tetrahedra in contact at exactly one vertex.*

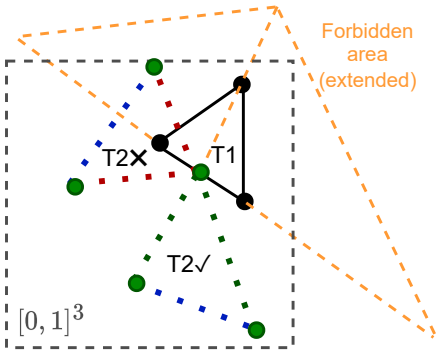

Figure 2: ***Vertex–Edge construction (2D analogue).*** *A contact point is sampled on an edge of T_1 and used as a shared vertex of the candidate simplex. The remaining vertices are sampled in $[0,1]^3$, while configurations whose opposite face intersects T_1 or enters the forbidden wedges induced by the rays from the contact point to the vertices of T_1 are rejected. Sampling is repeated until a valid simplex is obtained. In the full 3D setting, this yields a tetrahedron whose contact with T_1 is a single vertex lying on an edge of T_1.*

Vertex–Vertex To construct a singular point of contact at a shared vertex, we first sample $\mathcal{T}_1$ and select a random vertex v_{11} as the anchor. To ensure the volumes remain disjoint, we define a *forbidden region* by extending rays from v_{11} through the vertices of the opposite face, effectively bounding the cone of space already occupied by $\mathcal{T}_1$. We then sample the remaining three vertices of $\mathcal{T}_2$ such that their induced base triangle does not intersect this forbidden region. This rejection-based approach, detailed in Algorithm 2 and shown in Fig. 1, ensures that while the two tetrahedra share a vertex, their interior volumes emanate into separate angular sectors.

Algorithm 2 Constructing Vertex–Vertex contact.

repeat
  Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
  $v_{11} \leftarrow$ random vertex of $\mathcal{T}_1$
  $\{v_{12}, v_{13}, v_{14}\} \leftarrow$ vertices of face opposite v_{11}
  Construct forbidden region $\mathcal{R}$ by extending rays $(v_{11} \rightarrow v_{12,13,14})$
  Start timer t_s
  repeat
    Sample $\{v_{22}, v_{23}, v_{24}\} \sim \mathcal{U}([0,1]^3)$
    $\mathcal{T}_2 \leftarrow$ tetrahedron$(v_{11}, v_{22}, v_{23}, v_{24})$
    if $\mathcal{T}_2$ is valid **and** face $(v_{22}, v_{23}, v_{24}) \cap \mathcal{R} = \emptyset$ **then return** $(\mathcal{T}_1, \mathcal{T}_2)$
  until timeout **or** success
until success

Vertex–Edge In this configuration, a vertex of $\mathcal{T}_2$ is constrained to lie strictly within the interior of an edge of $\mathcal{T}_1$. After sampling $\mathcal{T}_1$ and selecting a target edge, we interpolate a contact point p along its length. Because the edge is an intersection of two faces, we construct a *forbidden wedge* consisting of two tetrahedral volumes that represent the occupancy of $\mathcal{T}_1$ local to that segment. The vertices of $\mathcal{T}_2$ are then sampled to ensure the resulting body does not penetrate this wedge or the primary volume of $\mathcal{T}_1$, as implemented in Algorithm 3 and visualized in Fig. 2.

Algorithm 3 Constructing Vertex–Edge contact.

repeat
  Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
  Select edge $E(v_{11}, v_{12}) \subset \mathcal{T}_1$ and $t \sim \mathcal{U}(0.01, 0.99)$
  $v_{21} \leftarrow (1-t)v_{11} + tv_{12}$ ▷ Anchor on edge
  Construct forbidden wedges $\mathcal{W}_{1,2}$ using rays $(v_{21} \rightarrow v_{11\ldots14})$
  repeat
    Sample $\{v_{22}, v_{23}, v_{24}\} \sim \mathcal{U}([0,1]^3)$
    $F_{base} \leftarrow$ triangle(v_{22}, v_{23}, v_{24})
    if $F_{base} \cap (\mathcal{W}_1 \cup \mathcal{W}_2 \cup \mathcal{T}_1) = \emptyset$ **then**
    return $(\mathcal{T}_1, \text{tetrahedron}(v_{21}, v_{22}, v_{23}, v_{24}))$
  until timeout
until success

Vertex–Face To ensure $\mathcal{T}_2$ touches $\mathcal{T}_1$ at a single vertex without penetration, we constrain the non-contact vertices of $\mathcal{T}_2$ to the open half-space defined by the target face's outward normal $\vec{n}$. We also apply an angular buffer $\epsilon = 10^{-16}$ during spherical sampling; without this margin, floating-point drift near the boundary plane frequently degrades to other unintended configurations. Algorithm 4 shows the whole method as pseudo-code and is visualized in Fig. 3.

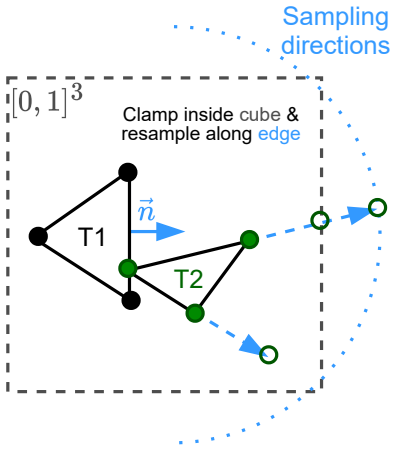

Figure 3: ***Vertex—Face construction (2D analogue).*** *A contact point p is sampled on an edge of T_1, and the remaining vertices of T_2 are generated in directions lying in the outward half-space defined by the local normal $\vec{n}$. Candidate vertices are truncated to remain inside the unit box $[0,1]^3$. In the full 3D setting, the same construction is applied to tetrahedra, with one vertex of T_2 lying on a face of T_1; the fourth vertex is omitted in this top-down schematic.*

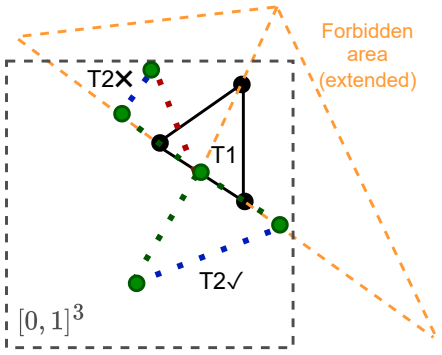

Figure 4: ***Edge–Edge construction (2D analogue).*** *A contact point is sampled in the interior of an edge of T_1 and enforced to lie on an edge of the candidate simplex by sampling one endpoint and reconstructing the second so that the edge passes through the contact point with a prescribed ratio. The remaining vertices are sampled in $[0,1]^3$, while configurations whose outer faces intersect T_1 or enter the forbidden region induced by rays from the contact point through the vertices of T_1 are rejected. Sampling is repeated until a valid simplex is obtained. In the full 3D setting, this yields two tetrahedra in edge–edge contact.*

Algorithm 4 Constructing tetrahedra with Vertex–Face contact.

$\epsilon \leftarrow 10^{-16}$ ▷ Numerical tolerance
Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
Select random face $F \subset \partial\mathcal{T}_1$ and sample point $p \in F$
Compute outward unit normal $\vec{n}$ for F
Construct local frame with origin p and z-axis aligned to $\vec{n}$
for $i \in \{1,2,3\}$ **do**
  Sample $\theta_i \sim \mathcal{U}(0,2\pi)$ and $\phi_i \sim \mathcal{U}([-\pi/2+\epsilon, \pi/2-\epsilon])$
  $\vec{d_i} \leftarrow (\sin\phi_i \cos\theta_i, \sin\phi_i \sin\theta_i, \cos\phi_i)$
  Compute $r_{\max}$ such that $p + r_{\max}\vec{d_i} \in [0,1]^3$
  Sample $r_i \sim \mathcal{U}([\epsilon, r_{\max}])$ and set $v_i \leftarrow p + r_i\vec{d_i}$
Construct $\mathcal{T}_2$ from $\{p, v_1, v_2, v_3\}$
return $(\mathcal{T}_1, \mathcal{T}_2)$

Edge–Edge To achieve a configuration where two edges cross at a single point without volumetric overlap, we identify a point P on an edge of $\mathcal{T}_1$ and build a bilateral forbidden wedge. We then construct an edge for $\mathcal{T}_2$ that is mathematically constrained to pass through P at a random interpolation ratio. The remaining vertices of $\mathcal{T}_2$ are sampled such that the "outer" faces of the tetrahedron stay strictly outside the forbidden regions and the interior of $\mathcal{T}_1$. This results in an interlocking geometry, described in Algorithm 5 and visualized in Fig. 4, where the only shared coordinate is the infinitesimal crossing point of the two line segments.

Algorithm 5 Constructing Edge–Edge contact.

repeat
  Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
  Select edge $E_1 \subset \mathcal{T}_1$ and point $P \in \text{int}(E_1)$
  Construct forbidden boundary from P through vertices of $\mathcal{T}_1$
  repeat
    Sample $v_{21} \sim \mathcal{U}([0,1]^3)$ and $t_2 \sim \mathcal{U}(0.01, 0.99)$
    $v_{22} \leftarrow$ point such that P divides $E(v_{21}, v_{22})$ by ratio t_2
    Sample $\{v_{23}, v_{24}\} \sim \mathcal{U}([0,1]^3)$
    $Faces_{outer} \leftarrow \{\text{tri}(v_{21}, v_{23}, v_{24}), \text{tri}(v_{22}, v_{23}, v_{24})\}$
    if $Faces_{outer} \cap (\text{ForbiddenWedge} \cup \mathcal{T}_1) = \emptyset$ **then**
    return $(\mathcal{T}_1, \text{tetrahedron}(v_{21}, v_{22}, v_{23}, v_{24}))$
  until timeout
until success

Edge–Face We constrain *two* vertices of $\mathcal{T}_2$ to the plane Π defined by a random face of $\mathcal{T}_1$, while the remaining vertices are sampled within the outward-facing hemisphere to guarantee the bodies remain disjoint, as detailed in Algorithm 6 and shown in Fig. 5.

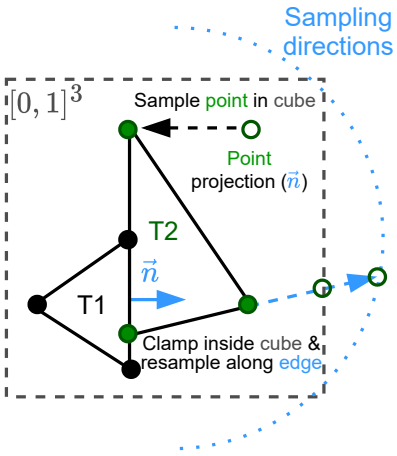

Figure 5: ***Edge—Face construction (2D analogue).*** *A first contact point is chosen on T_1, and a second point sampled in the box is projected onto the supporting plane (shown here as a line in 2D), defining the contacting edge of T_2. The remaining vertices are sampled in outward directions relative to the normal $\vec{n}$, while enforcing the box constraint $[0,1]^3$. In the full 3D tetrahedral setting, this yields an edge of T_2 lying in the plane of a face of T_1.*

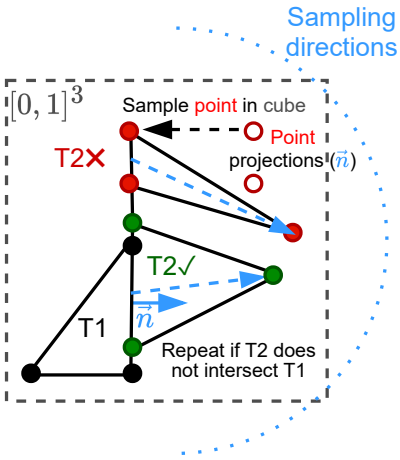

Figure 6: ***Face–Face construction (2D analogue).*** *Two vertices of the candidate simplex are obtained by sampling points in $[0,1]^3$ and projecting them onto the supporting plane of a selected face of T_1 (shown here as an edge in 2D), while the remaining vertex is sampled in the outward half-space defined by the local normal $\vec{n}$ and clipped to remain inside the box. Candidate configurations are resampled until the constructed simplex intersects T_1. In the full 3D setting, this yields a tetrahedron whose contact with T_1 is polygonal; the fourth vertex is omitted in this schematic.*

Algorithm 6 Constructing tetrahedra with Edge–Face contact.

$\epsilon \leftarrow 10^{-16}$ ▷ Numerical tolerance
Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
Select random face $F \subset \partial\mathcal{T}_1$ with plane Π and outward normal $\vec{n}$
Sample $v_1 \in F$
Sample $p \sim \mathcal{U}([0,1]^3)$ and set $v_2 \leftarrow \text{proj}_\Pi(p)$ ▷ Force v_1, v_2 coplanar with F
Construct local frame with origin v_1 and z-axis aligned to $\vec{n}$
for $i \in \{3,4\}$ **do**
    Sample $\theta_i \sim \mathcal{U}(0, 2\pi)$ and $\phi_i \sim \mathcal{U}([-\pi/2+\epsilon, \pi/2-\epsilon])$
    $\vec{d_i} \leftarrow (\sin\phi_i \cos\theta_i, \sin\phi_i \sin\theta_i, \cos\phi_i)$
    Compute $r_{\max,i}$ such that $v_1 + r_{\max,i}\vec{d_i} \in [0,1]^3$
    Sample $r_i \sim \mathcal{U}([\epsilon, r_{\max,i}])$ and set $v_i \leftarrow v_1 + r_i\vec{d_i}$
Construct $\mathcal{T}_2$ from $\{v_1, v_2, v_3, v_4\}$
return $(\mathcal{T}_1, \mathcal{T}_2)$

Face–Face We constrain three vertices of $\mathcal{T}_2$ to the plane Π containing a face of $\mathcal{T}_1$. To complete the shape without causing volumetric overlap, the fourth vertex is positioned relative to the centroid of these three points, sampled strictly within the outward-facing hemisphere defined by the face normal, as implemented in Algorithm 7 and visualized in Fig. 6.

Algorithm 7 Constructing tetrahedra with Face–Face contact.

$\epsilon \leftarrow 10^{-16}$ ▷ Numerical tolerance
Sample $\mathcal{T}_1 \sim \mathcal{U}([0,1]^3)$
repeat
    Select random face $F \subset \partial\mathcal{T}_1$ with plane Π and outward normal $\vec{n}$
    for $i \in \{1,2,3\}$ **do**
        Sample $p_i \sim \mathcal{U}([0,1]^3)$ and set $v_i \leftarrow \text{proj}_\Pi(p_i)$ ▷ Force 3 vertices coplanar with F
    Compute centroid $c \leftarrow \frac{1}{3}(v_1 + v_2 + v_3)$
    Construct local frame with origin c and z-axis aligned to $\vec{n}$
    Sample $\theta \sim \mathcal{U}(0, 2\pi)$ and $\phi \sim \mathcal{U}([-\pi/2+\epsilon, \pi/2-\epsilon])$ ▷ Constrain 4th vertex to upper hemisphere
    $\vec{d} \leftarrow (\sin\phi\cos\theta, \sin\phi\sin\theta, \cos\phi)$
    Compute $r_{\max}$ such that $c + r_{\max}\vec{d} \in [0,1]^3$
    Sample $r \sim \mathcal{U}([\epsilon, r_{\max}])$ and set $v_4 \leftarrow c + r\vec{d}$
    Construct $\mathcal{T}_2$ from $\{v_1, v_2, v_3, v_4\}$
until `CGAL::do_intersect`$(\mathcal{T}_1, \mathcal{T}_2)$
return $(\mathcal{T}_1, \mathcal{T}_2)$

Volumetric Intersection Tangential contacts form measure-zero subsets of the configuration space. As a result, a simple rejection strategy is sufficient to ensure volumetric penetration, and the sampling procedure in Algorithm 8 remains highly efficient.

Algorithm 8 Constructing volumetric intersections.

1: **repeat**
2: Sample $\mathcal{T}_1, \mathcal{T}_2 \sim \mathcal{U}([0,1]^3)$
3: *overlap* $\leftarrow$ `CGAL::do_intersect`$(\mathcal{T}_1, \mathcal{T}_2)$
4: **until** *overlap* ▷ Rejection sampling yields $V_{int} > 0$ with high probability
5: **return** $(\mathcal{T}_1, \mathcal{T}_2)$

Empirically, we observe that intersection volumes exceeding 2×10^{-2} are statistically harder to sample without deliberate construction, while volumes below 10^{-7} approach the noise floor for model training.

To ensure numerical stability and effective learning, we constrain the target domain to $v \in [10^{-7}, 20^{-2}]$. The lower bound provides a safety margin for gradient stability, while the upper bound captures the vast majority of naturally occurring overlaps in our probability distribution. Any samples falling outside this range are rejected during generation. Within these bounds, we employ a *log-uniform* sampling strategy, by oversampling underrepresented ranges, by doing so, we prevent the model from overfitting to the abundance of small intersections while neglecting the larger, physically critical collision events.

4. Training Data Distribution Optimization

We investigate how the composition of training data affects generalization across geometric intersection types. Using a fixed architecture (MLP, 3 × 128 neurons, 36K parameters) and dataset size (100K samples), we systematically vary the proportion of samples from each category. All models are trained for 20 epochs with AdamW (lr=0.001, batch=32) and evaluated on five separate 100K-sample test sets, one per geometric type.

Each training distribution is represented as percentages for: No Intersection, Point, Segment, Polygon, Polyhedron. Table 1 shows the iterative search results.

5. Geometric Transformations for Input Normalization

Increasing model capacity is not the only way to increase performance; we can also aim to reduce variance and align data into a canonical frame, thereby embedding inductive biases related to spatial relationships. Two primary strategies were evaluated: Unitary Tetrahedron Transformation and Principal Axis Transformation.

5.1. Unitary Tetrahedron Transformation

This approach simplifies the problem by mapping a reference tetrahedron (T_1) into a fixed unit frame, defined as a standard simplex. The task is thus reformulated as determining whether a given tetrahedron intersects this fixed simplex and, if so, computing the associated intersection volume. As outlined in Algorithm 9, the affine transformation preserves intersection relationships while introducing a linear scaling of volumes. This allows the learning process to condition solely on T_2, reducing the effective learning complexity, albeit at the cost of potentially increasing nonlinearity in the learned mapping for intersection volume prediction.

Algorithm 9 Unitary Tetrahedron Transformation

Require: Two tetrahedra T_1 (reference), T_2, each defined by 4 vertices in $\mathbb{R}^3$
Ensure: Transformed T_2' relative to unitary frame and volume correction factor
1: Select T_1 as the reference frame
2: Center T_1 by subtracting its centroid or anchor vertex
3: Compute edge vectors of T_1 relative to base vertex to construct a basis
4: Derive transformation matrix A that maps $T_1 \rightarrow$ unitary tetrahedron
5: Compute $T_2' = A \cdot (T_2 - v_0)$, where v_0 is the reference vertex of T_1
6: Use T_2' as input to the model; omit T_1 (it is fixed in this space)

5.2. Principal Axis Transformation

An alternative technique involves aligning the input tetrahedra with their intrinsic orientation using Principal Component Analysis (PCA). By rotating the objects into a canonical pose without altering scale, PCA reduces rotational variance, allowing the model to focus on other geometric features, as detailed in Algorithm 10.

Algorithm 10 Principal Axis Transformation via PCA

Require: Two tetrahedra T_1, T_2, each defined by 4 vertices in $\mathbb{R}^3$
Ensure: Rotated tetrahedra T_1', T_2' aligned to principal axes of T_1
1: Compute centroid of T_1
2: Translate vertices of T_1 and T_2 so centroid of T_1 is at origin
3: Perform PCA on T_1 vertices to extract principal directions (eigenvectors)
4: Construct rotation matrix R from sorted eigenvectors of covariance matrix
5: Rotate $T_1' = R \cdot T_1$ and $T_2' = R \cdot T_2$

6. Additional Experiments

In this section, we evaluate `TetrahedronPairNet` across: inference throughput, prediction accuracy, and their interaction with model capacity and training data size. All experiments use consumer-grade hardware—an Intel Core i5 CPU (4 physical cores, 8 threads) with 8GB RAM running Ubuntu 20.04. We employ CPU-only inference to ensure a fair comparison with CGAL, which runs sequentially on the CPU.

Our test set comprises 5×10^5 samples: 10^5 pairs from each of the five geometric categories (no intersection, point, segment, polygon, polyhedron) defined in Section 3. Unless otherwise noted, reported metrics represent averages across this stratified test set.

6.1. Baseline: CGAL Exact Methods

We benchmark CGAL's exact geometric predicates on a single CPU core. Intersection detection (`CGAL::do_intersect()`) is

Training Distribution	No Int.	V-F	E-F	F-F	Volumetric	Mean
Baseline: Pairwise splits						
(50,50,0,0,0)	97.1	97.9	93.7	35.4	3.6	65.5
(50,0,50,0,0)	97.0	97.4	97.8	42.2	7.6	68.4
(50,0,0,50,0)	94.5	89.8	89.9	87.3	33.7	79.0
(50,0,0,0,50)	98.1	7.6	18.8	27.3	86.9	47.7
H1: Balanced	95.7	94.0	94.4	79.6	65.9	85.9
(50,10,15,20,5)						*+6.9pp*
H2: Complex emphasis	93.3	92.2	93.0	82.6	84.9	89.2
(50,5,7,22,16)						*+3.3pp*
H3: Further tuning	93.8	92.0	91.9	83.1	78.8	87.9
(50,4,5,23,18)						*-1.3pp*
H4: Final	89.1	95.9	96.3	89.1	90.1	**92.1**
(50,5,7,20,18)						*+2.9pp*

Table 1: ***Generalization across intersection types.*** *While single-category baselines suffer from severe overfitting (e.g., polyhedron-only models achieve only 7.6% accuracy on point contacts), mixed strategies significantly improve robustness. The final distribution* ***H4*** *not only achieves the highest mean accuracy (92.1%) but also outperforms the category-specific baseline on its own task (90.1% vs 86.9% for polyhedron cases), suggesting the model learns fundamental geometric features rather than memorizing contact patterns.*

evaluated on the whole test set of 500k samples (100k per category). Due to high computational cost, exact volume computation is measured on a subset of 2,000 random volumetric intersecting pairs using the complete CSG pipeline: Nef polyhedron construction, boolean intersection, mesh conversion, and volume integration.

As shown in Table 2, while boolean detection is efficient, the computational cost of exact volume reconstruction (via Nef polyhedra) renders it prohibitively expensive for large-scale real-time applications.

Operation	Time / Sample	Throughput
Intersection Detection	157 μs	6,352 /s
Exact Volume Comp.	28.31 ms	35 /s

Table 2: ***Computational bottleneck of exact methods.*** *Calculating exact intersection volumes is two orders of magnitude slower than simple collision detection. At this rate, processing a batch of 100,000 pairs requires close to 47 minutes, compared to just 16 seconds for detection.*

6.2. Inference Speed vs. Model Size

We investigate the capacity-throughput trade-off by benchmarking 10 model variants (ranging from 498 to 59,778 parameters) across varying batch sizes. The models were generated by scaling layer width and depth as detailed in Table 3. The resulting throughput trends across model sizes and batch regimes are visualized in Figure 7.

Configuration Parameter	Value / Range
Parameter Range	498 – 59,778
Inference Device	GPU (Float32)
Inference Batch Sizes	128, 256, 512, 1024, 2048
Benchmark Protocol	100 runs (+10 warmup)

Table 3: *Experimental setup for throughput analysis.*

6.3. Training Data Size vs. Accuracy

We examine how prediction quality scales with dataset size and model capacity. We train three representative model configurations (Small, Medium, Large) on datasets ranging from 10k to 1M samples, evaluating on a fixed 500k-sample test set. The experimental parameters are detailed in Table 4, and the resulting accuracy trends across data regimes and model sizes are shown in Figure 8.

Configuration Parameter	Value / Range
Parameter Counts	4k / 19k / 82k
Training Set Sizes	10k, 50k, 100k, 500k, 1M
Test Set Size	500k (Fixed, Stratified)
Training Protocol	20 Epochs (Same as Sec. 6.2)

Table 4: *Experimental setup for data scaling analysis.*

The optimal choice depends on application requirements. For real-time collision detection where throughput is critical, the smaller models offer the best balance. For applications requiring higher accuracy with relaxed time constraints, the largest model may be justified despite reduced throughput.

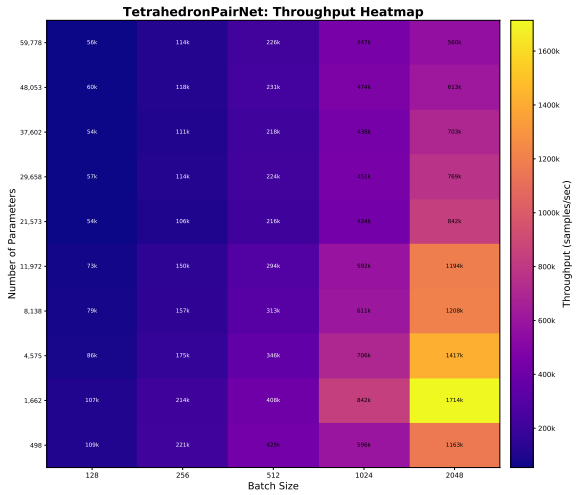

Figure 7: ***Throughput analysis (samples/sec).*** *The heatmap illustrates the trade-off between model capacity and inference speed. Note that the largest model (59,778 parameters) maintains a robust throughput of 560k samples/sec at a batch size of 2048, demonstrating the architecture's suitability for high-load environments even at higher capacities*

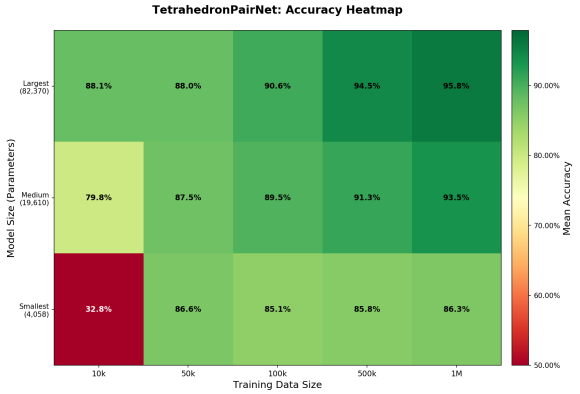

Figure 8: ***Accuracy scaling behavior.*** *The Smallest model saturates early (≈ 86% accuracy at 50k samples), unable to leverage additional data. In contrast, the Largest model improves steadily, though gains beyond 500k samples are marginal (1–2pp).*

References

[DSX*23] DING Z., SUN Y., XU S., PAN Y., PENG Y., MAO Z.: Recent advances and perspectives in deep learning techniques for 3d point cloud data processing. *Robotics 12*, 4 (2023). 1

[GWH*20] GUO Y., WANG H., HU Q., LIU H., LIU L., BENNAMOUN M.: Deep learning for 3d point clouds: A survey, 2020. 1

[MQY*22] MA X., QIN C., YOU H., RAN H., FU Y.: Rethinking network design and local geometry in point cloud: A simple residual mlp framework, 2022. arXiv:2202.07123. 1

[QA24] QUACKENBUSH B., ATZBERGER P. J.: Geometric neural operators (gnps) for data-driven deep learning of non-euclidean operators, 2024. URL: https://arxiv.org/abs/2404.10843, arXiv:2404.10843. 1

[QYSG17] QI C. R., YI L., SU H., GUIBAS L. J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. 1

[WSL*19] WANG Y., SUN Y., LIU Z., SARMA S. E., BRONSTEIN M. M., SOLOMON J. M.: Dynamic graph cnn for learning on point clouds, 2019. arXiv:1801.07829. 1

[YYJ22] YE Y., YANG X., JI S.: Apsnet: Attention based point cloud sampling, 2022. arXiv:2210.05638. 1

[Zin03] ZINANI A.: The expected volume of a tetrahedron whose vertices are chosen at random in the interior of a cube. *Monatshefte für Mathematik 139*, 4 (aug 2003), 341–348. doi:10.1007/s00605-002-0531-y. 1

[ZWT*23] ZHANG H., WANG C., TIAN S., LU B., ZHANG L., NING X., BAI X.: Deep learning-based 3d point cloud classification: A systematic survey and outlook. *Displays 79* (2023), 102456. 1