Technical Appendix:
Boundary Graph Neural Networks for 3D Simulations

TApp. A Discrete Element Method (DEM) simulator LIGGGHTS

The open source DEM software LIGGGHTS (Kloss et al. 2012) is based on the Molecular Dynamics code LAMMPS (Plimpton 1995) developed by Sandia National Labs. Due to the similarity of the underlying algorithms for neighbor list construction, output and parallelism this provided a stable basis for the contact models required for DEM. LIGGGHTS added support for triangular mesh walls (as e.g. such ones visualized in Fig. TApp. A.1), particle insertion and new particle shapes (multispheres and superquadrics). Several of those changes resulted in upstream contributions in LAMMPS.

Over the years LIGGGHTS has become a widely used software in both academia and industry that supports both cutting edge research and industrial applications. Support for several physical phenomena as, e.g. liquid transfer on particles, was instrumental in its success. However, it also highlighted requirements for additional research. In industrial applications there often is the need to study physical phenomena which occur on different time scales, e.g. particle collisions ($\mathcal{O}(10^{-5}s)$) vs. moisture content in particles ($\mathcal{O}(1s)$, (Mellmann et al. 2011)), which can lead to weeks of simulation time. While advances have been made to overcome such issues (e.g. Kloss et al. (2017)), they remain limited in their application, due to the fact that they rely on prior simulation of the exact setup and cannot be used for interpolation of quantities directly related to the flow behavior.

In 2019 LIGGGHTS was again forked and forms the basis of the commercial DEM software Aspherix® which expands the capabilities of LIGGGHTS with polyhedral particles, a significantly simplified input language and graphical user interface.

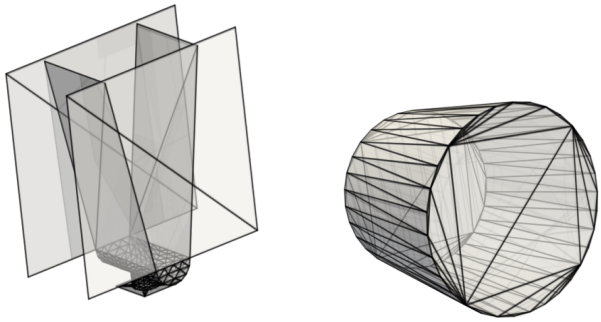

Figure TApp. A.1: Triangularized boundary surfaces of the hopper (left) and the rotating drum (right). The accurate description of rather simple, curved geometries requires a relatively large number of triangles of different shapes and sizes.

TApp. B Minimum Triangle - Point Distance

The main part of the manuscript states the optimization problem, which needs to be solved. According to Eberly (1999) seven cases have to be to distinguished: one (c0) in which $\boldsymbol{t}(u_0', u_1')$ is located within the (closed) triangle, three (c1, c3, c5) in which $\boldsymbol{t}(u_0', u_1')$ is located on one edge of the triangle (including the edge corner points as special cases), and three (c2, c4, c6) in which $\boldsymbol{t}(u_0', u_1')$ is located on one of two edges (including the triangle corner points as special cases). We visualize these cases in Fig. TApp. B.1.

TApp. C Normal Vector Representations

There is an ambiguity in the representation of planes via normal vectors due to the two possible orientations of the normal vectors, which correspond to the same geometry. In general, there are two possibilities of including normal vector information into the model: (i) encoding always that triangle plane normal vector which always points towards or away from the corresponding particle, (ii) including positively and negatively oriented versions of the normal vector, and order them. We decided for option (ii) since pathological cases where the particle is in the same plane as the triangle are avoided and training is further stabilized. In doing so, we have to deal with the fact that the network predictions should be invariant with respect to the orientation of the

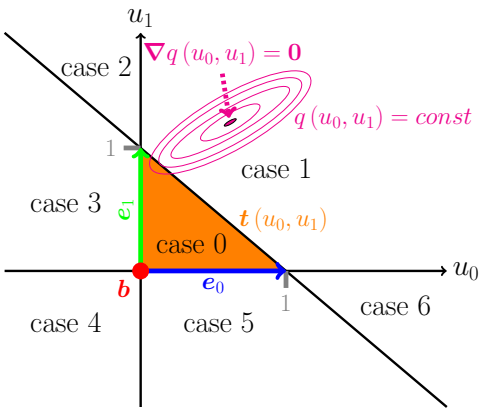

Figure TApp. B.1: Visualization of point - triangle distance calculations in 3D. The triangle is represented by a parameterized function $\boldsymbol{t}\,(u_0, u_1) = \boldsymbol{b} + u_0\ \boldsymbol{e}_0 + u_1\ \boldsymbol{e}_1$ with $u_0 \geq 0$, $u_1 \geq 0$, $u_0 + u_1 \leqslant 1$ (indicated by the orange area). Level sets of $q\,(u_0, u_1)$ are indicated by ellipses and describe the squared Euclidean distance of a triangle point $\boldsymbol{t}\,(u_0, u_1)$ to the point $\boldsymbol{p}$, for which we compute the minimum distance.

normal vectors. Therefore, we define a partial ordering which is able to sort the normal vectors with respect to their orientations. For a given normal vector $\boldsymbol{n} = (n_1, n_2, n_3) \in \mathbb{R}^3$, we use the following partial order function

$$f_o(\boldsymbol{n}) = \sum_{i=1}^{3} 3^{i-1}\ (\text{sgn}\ (n_i) + 1) \tag{TApp. C.1}$$

to retrieve the scalar values $f_o(\boldsymbol{n})$ and $f_o(-\boldsymbol{n})$ and sort the two vectors according to their corresponding mapped values. Different sign combinations of the normal vectors are shown in Fig. TApp. C.1. To test the performance of our approach, we conduct a toy experiment as well as a simulation experiment with different representations of normal vectors. We describe both experiments in TApp. C.1 and TApp. C.2.
It should be mentioned, that quite recently a new idea based on the ideas of Charles et al. (2017) and Zaheer et al. (2017) has been invented (Lim et al. 2022) in a different context than ours and which was denoted as being a solution for "sign invariances". Their idea could also serve to be useful for resolving problems with orientation independent normal vector representations. While our approach works directly on the input vectors, their approach needs larger changes in the overall network architecture, but may on the other hand have advantages if the input itself would be the result of a DNN to get end-to-end differentiable architectures.

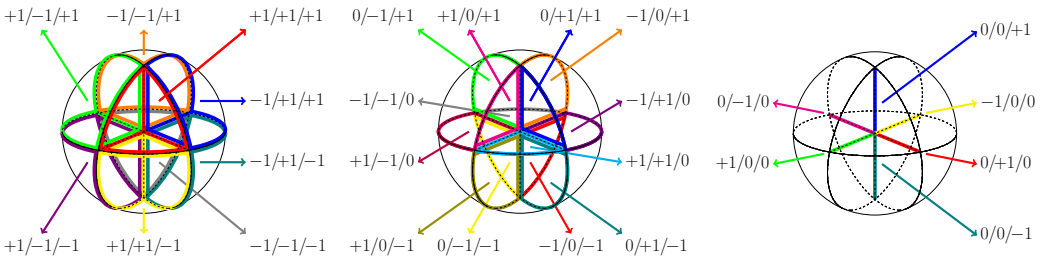

Figure TApp. C.1: Partial ordering of normal vectors. The numbers indicate the signs of normal vector components, which are used in the partial order function. The figures from left to right visualize different (ordered) sign combinations. Sets of sign combinations without zero values form volumes (left), sets with one zero value form planar areas (middle), and sets with two zero values form line sections (right). The 0/0/0 combination forms a point at the origin.

TApp. C.1 Reflection Toy Example

We conduct a toy experiment to showcase that a partial ordering of normal vectors is helpful for learning 3D simulations. In detail, we consider reflection (Ref) at a plane $\boldsymbol{n}$ as given by

$$Ref_{\boldsymbol{n}}\left(\boldsymbol{v}\right) = \boldsymbol{v} - 2\frac{\boldsymbol{v}\cdot\boldsymbol{n}}{\boldsymbol{n}\cdot\boldsymbol{n}}\boldsymbol{n}\,, \tag{TApp. C.2}$$

and try to learn the reflection formula by a simple ReLU network, which takes the 3 components of $\boldsymbol{n}$ and $\boldsymbol{v}$ as input features and predicts the 3 components of $Ref_{\boldsymbol{n}}\left(\boldsymbol{v}\right)$. The training data consists of reflections at four fixed walls: the top, the bottom, the left, and, the right side of a simple cube. We use normal vectors of these walls, that point towards the inner of the cube. When evaluating the performance of the trained models, we observe decent predictions, if the orientation of the normal vectors describing the inclination of the walls was equal to the training data (see R1-R4 in Fig. TApp. C.2). However, for inverted normal vectors in the test set, only networks which take a partial ordering of the normal vectors into account predict the reflection correctly (see R3, R4 in Fig. TApp. C.3).

TApp. C.2 Simulation Experiment

We compare three different versions of how to include normal vector information for the hopper particle flow experiments:

- not including normal vector information, and filling six node features up with zero entries instead (V1)
- including single normal vector orientation, which is given by the triangle corner point order of the mesh (V2)
- including both normal vector orientations (six features) (V3).

From an information perspective, it should be noted that (i) distance information (scalar distance and distance vectors) to the walls is present in the edge features of the graph and (ii) in most cases the used normal vectors are oriented towards the outside of relevant border walls. The different particle distribution trajectories obtained by the three versions are compared by computing the Earth Movers distances (Bonneel et al. 2011; Flamary et al. 2021, EMD) of predicted and simulated trajectories. We use Euclidean distances for the cost matrix, which we compute at time steps $2^0, 2^1, \cdots, 2^{16}$ for 5 training trajectories and 5 test trajectories. Table TApp. C.1 shows the means (μ) and standard deviations (σ) of EMD values at different time steps and from 5 different training and test trajectories. A paired Wilcoxon test on the concatenated trajectories, shows that V3 significantly outperforms V1 (p-value 2.42e-04) and V2 (p-value 1.50e-03) on the test data. Interestingly, there is less significance on the training data, which might indicate that the usage of orientation-independent features to represent walls, helps to improve generalization performance, while it might not be that helpful for optimization purposes alone.

Table TApp. C.1: Usage of different normal vector information in hopper particle flow experiment. The table summarizes means (μ) and standard deviations (σ) of the EMD for the different versions and shows the results of a paired Wilcoxon test.

Version		**Train**			**Test**		
		μ	σ	p-value Row < V3	μ	σ	p-value Row < V3
V1	No normal vector	5.06e-05	1.17e-04	2.36e-02	6.80e-05	1.59e-04	2.42e-04
V2	Single normal vector	1.15e-04	3.84e-04	3.40e-03	1.21e-04	4.33e-04	1.50e-03
V3	Both orientations	5.99e-05	1.77e-04		6.36e-05	2.06e-04	

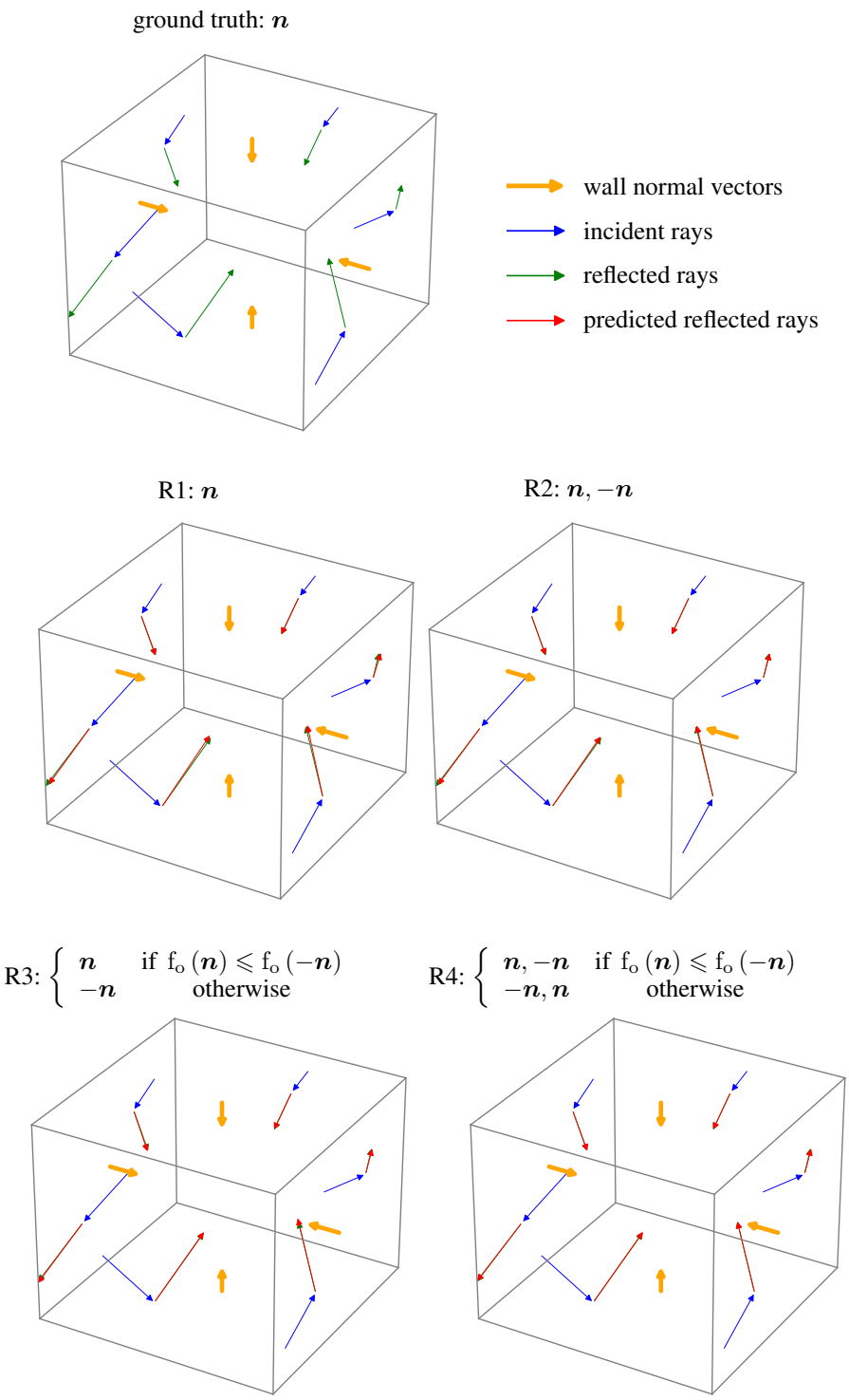

Figure TApp. C.2: Reflection of rays at four different walls (left, right, bottom, top). Wall normal vectors are visualized by bold orange arrows. The incident rays are visualized by blue arrows, reflected rays by green arrows, and neural network predictions by red arrows. Neural network predictions are based on wall representations **that are oriented the same way as in the training phase**. The caption above each plot indicates the wall input features used for training each of the networks.

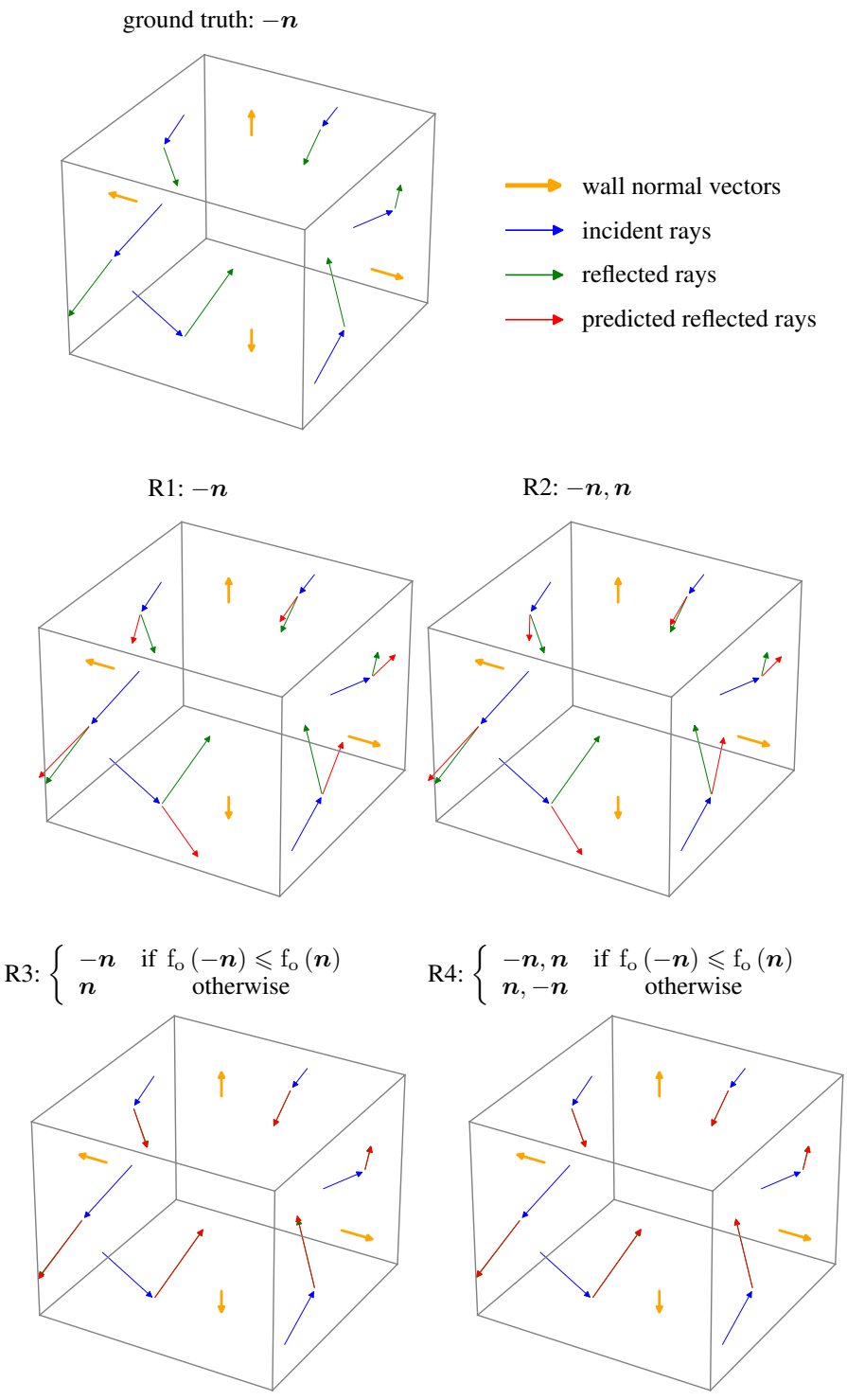

Figure TApp. C.3: Reflection of rays at four different walls (left, right, bottom, top). Wall normal vectors are visualized by bold orange arrows. The incident rays are visualized by blue arrows, reflected rays by green arrows, and neural network predictions by red arrows. Neural network predictions are based on wall representations **that are inversely oriented compared to the training phase**. The caption above each plot indicates the wall input features used for training each of the networks.

TApp. D Experiments

TApp. D.1 Simulation details

Hopper The initialization consists of two phases. In a first step, particles are randomly inserted into a small cuboid which is positioned at a certain height above the closed hole of the hopper. This cuboid is continuously filled with particles during the initialization phase and afterwards particles freely move downwards (along the direction of gravity). In this way, the hopper is filled up to a certain height with 20,000 particles. In a second phase, we cut out particles from the filled mass of particles. We do this (i) by applying randomly selected functions and by (ii) randomly filtering out particles from the whole particle mass. The randomly selected functions are e.g. hyperplanes, where we only keep particles if they are at the same side of the hyperplane. The inserted particles have a radius of 0.002 m.

Drum For initialization we assume that the direction of gravity is different than the usual gravitation direction. We insert particles at two random fixed regions within the drum. After the particles are inserted, they can move according to the gravitation direction during the initialization phase. In this way, we obtain different initial particle distributions within the drum. The inserted particles have a radius of 0.01 m.

TApp. D.2 Implementation details

Graph Neural Network Raw inputs to our graph networks are initial particle positions and the particle positions from the 5 previous frames of the simulations. From these positions velocities are computed. Further inputs include the particle type and the coordinates of the triangle mesh of the respective time frame. We use residual connections (He et al. 2016) for both node and edge updates. For both updates, we use simple two-layer MLP networks, ReLUs (Nair and Hinton 2010) after the first layer, and layer normalization (Ba, Kiros, and Hinton 2016) without an additional activation after the second layer. For layer normalization we consider the ϵ-parameter as a hyperparameter and set it to 1.0. The networks for input embedding and read-out are similar to the message passing layers without layer normalization. The network weights are initialized similar to He et al. (2015); for the input embeddings we assume an increased number of input neurons for `fan_in`, where we consider the additional neurons as virtual copies of e.g. the wall indication feature in order to be able to upweight the influence of these features. We use the mean-squared error as an objective and train with Adam optimization (Kingma and Ba 2015). In order to facilitate learning, we provide as hyperparameter options not only $\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2, \hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j$ as features to the network, but also $\frac{1}{\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|}, \frac{\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j}{\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2}$ and $\frac{1}{\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^2}, \frac{\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j}{\|\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_j\|^3}$ reflecting the inverse distance law and the inverse-square law, which are present in many physical laws. We normalize input and target vectors and use a variant of Kahan summation (Kahan 1965; Klein 2006) in order to compute numerically stable statistics across particles of our dataset.

Hyperparameter Selection We keep 5 trajectories for each setting aside for validation. Criterions for hyperparameter selection are (i) that particles stay within the geometric object, and (ii) that the ground truth trajectory is reproduced.

TApp. D.3 Experimental Results For Cohesive Material

In the following, experimental results for a cohesive material are shown. The results in the main paper are obtained for non-cohesive granular material, i.e. material with a cohesion energy density of 0 J/m^3, which results in liquid-like behaviour. Increasing the cohesion energy density to 10^5 J/m^3 corresponds to cohesive granular material, i.e. the particles have a strong tendency to clump together. Figure TApp. D.1 shows the corresponding comparison of physical quantities for the cohesive granular material. Like in the non-cohesive case, the predictions for cohesive granular material are widely in agreement with the ground truth simulation.

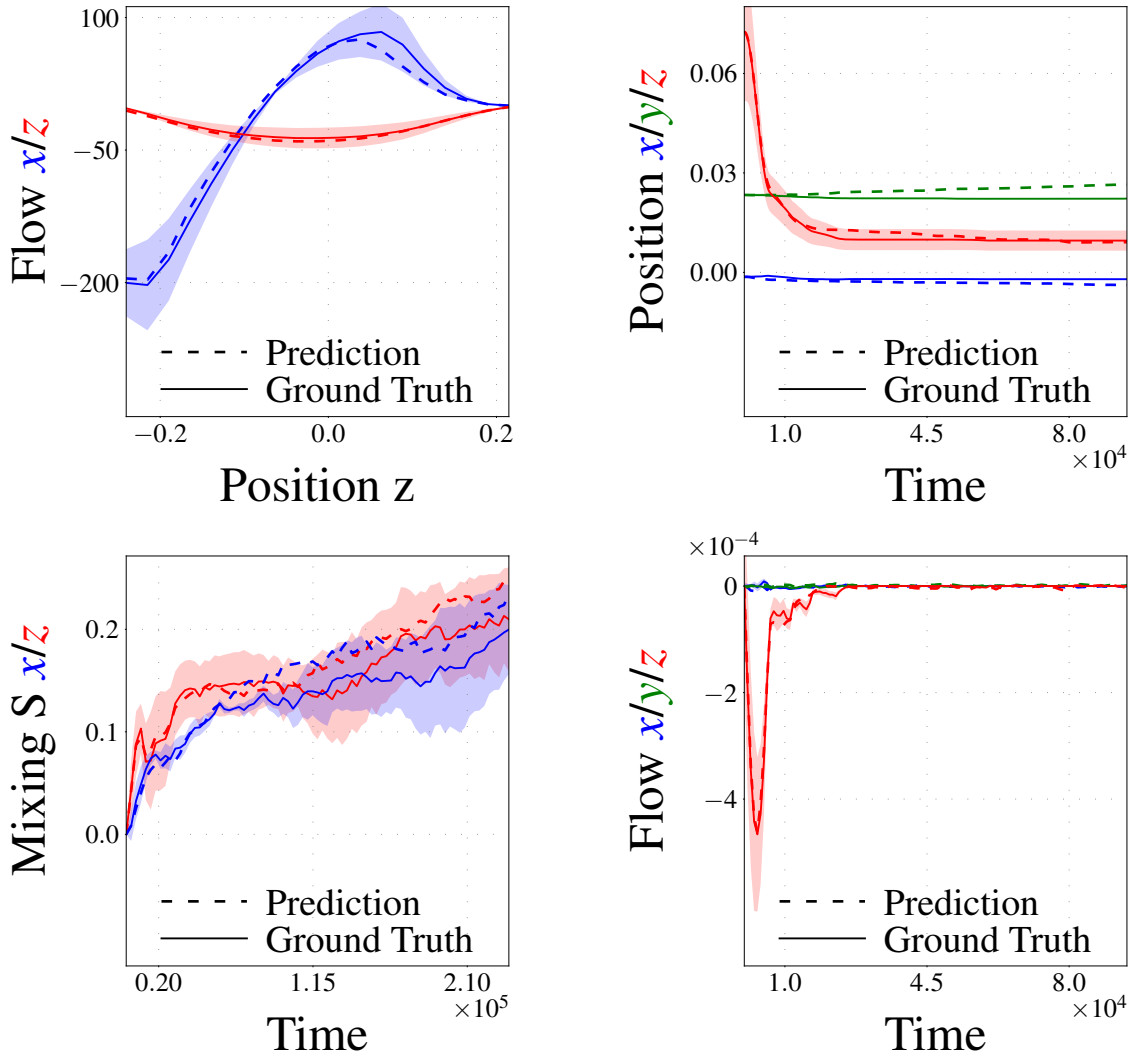

Figure TApp. D.1: Experimental results for cohesive granular meterial. Left, top: Time integrated particles flow in rotating drum in the x and z direction as a function of the position along the z axis. Left, bottom: Mixing entropies in the rotating drum as a function of time for particle class assignments according to the x (blue) and z (red) position. Right, top and bottom: Average particle position and particle flows for the hopper as a function of time.

TApp. D.4 A word on hopper OOD experiments

For hopper geometries (see e.g. Fig. 3), OOD experiments are characterized by an increase of the side wall inclination angles and by an decrease of the radii of the outlet sizes. Especially due to the latter, we expect fewer particles to hit the ground for OOD architectures if particle-particle and particle-boundary interactions are correctly modeled. In order to statistically test OOD trajectories against in-distribution trajectories, we consider the proportion of particles which have traversed through the outlet of the hopper. We therefore create 15 in-distribution and 15 OOD trajectories for both cohesive and non-cohesive materials. We then apply a Mann-Whitney U test which assesses the proportion of in-distribution against the proportion of OOD particles traversing the outlet. The null hypotheses is that the same or a higher proportion of particles traverses the outlet for the case of an OOD trajectory compared to an in-distribution trajectory.

Table TApp. D.1: Comparison of the proportion (mean μ and std σ) of particles beyond the outlet of the hopper

	cohesive		non-cohesive	
domain	μ	σ	μ	σ
in-distribution	0.34	0.09	0.89	0.03
OOD	0.14	0.11	0.73	0.15

The Mann-Whitney U test shows that the predicted proportion values are significantly lower for OOD than for in-distribution trajectories (p-value $\leqslant 1.4 * 10^{-4}$ for the cohesive material and p-value $\leqslant 1.2 * 10^{-4}$ for the non-cohesive material). We remind the reader that this is expected due to the on average reduced outlet size in OOD geometries. Furthermore, we compare the predicted proportion values of the cohesive and the non-cohesive model under the null hypothesis that the same or a higher proportion of particles traverses the outlet for the case of cohesive trajectories compared to non-cohesive trajectories. The applied Mann-Whitney U test yields a p-value $\leqslant 1.70 * 10^{-06}$ for the alternate hypothesis that the proportion is lower for the cohesive model, which is also in agreement with rational arguments (cohesive particles tend to clump together) and observations.

TApp. D.5 Ablation studies

For ablating BGNNs, we especially considered two design choices:

1. Does sampling the triangularized boundaries also work?
2. Is a unidirectional particle-wall interaction better suited to learn the corresponding particle-wall dynamics?

In Figs. TApp. D.2, TApp. D.3, TApp. D.4 and TApp. D.5 these points are assessed for cohesive and non-cohesive particles in hopper and rotating drum geometries, respectively. For the figures we kept hyperparameters consistent among the respective ablations (due to computational reasons). It should however be kept in mind, that this might not be optimal for the ablation, since individual hyperparameter optimization may lead to improved results. Nevertheless, we qualitatively tried to find an explanation what our trained ablation models might have learnt in order to avoid detected pitfalls.

A consistent result across both geometries and particle types was that models with only sparse sampling may not capture particle dynamics correctly. For including bidirectional edges, the behaviour was not always completely consistent.

In our ablation experiments with dense sampling, we tried to sample the triangle surfaces systematically, where the sampled points usually have distances that are in the range of particle diameters. For the sparser versions, we used multiples of the particle diameter (three, five). We found, that sampling might work well in the case of more homogeneous problems. For instance, in the case of cohesive particles, that tend to stick together, the model has learnt to avoid that a cluster of particles can move through boundary walls (see sparse sampling in Fig. TApp. D.2). Dense sampling might also work well for simple, homogeneous geometric settings like a drum. Although some particles for dense sampling are wrongly predicted to be at the top of the drum mesh in Fig. TApp. D.4, which may have occurred due too non-optimal hyperparameter settings, dense sampling has captured the basic dynamics well. This is not any more the case for sparser settings, where especially for the non-cohesive case, particles may freely move through the walls (see, e.g., empty drum in Fig. TApp. D.5). It is interesting, that the model relying on dense sampling in Fig. TApp. D.3 did not capture the moving dynamics at the ground very well. A reason for this could be, that the limited set of training samples, did not contain geometrically similar interactions between real and sampled virtual particles.

While bidirectional particle - wall edges seemed to work for the case of the rotating drum, there were difficulties in capturing particle dynamics for the hopper. It should be mentioned, that our idea for unidirectional edges was, that a symmetry break in message passing might be helpful in learning particle - wall interactions. Since there may be more variety in particle - wall interactions of the hopper than for the rotating drum, our design choice may be supported by the ablation experiment. Related to node connectivity, we also checked, whether it would be better to insert one respective virtual triangle node per neighboring real particle or whether we could use one virtual super-node per triangle for all edges instead. In both cases, the edges carry information on the minimum triangle-point distances. There may be subtle differences, which were likely induced by different weightings of the nodes.

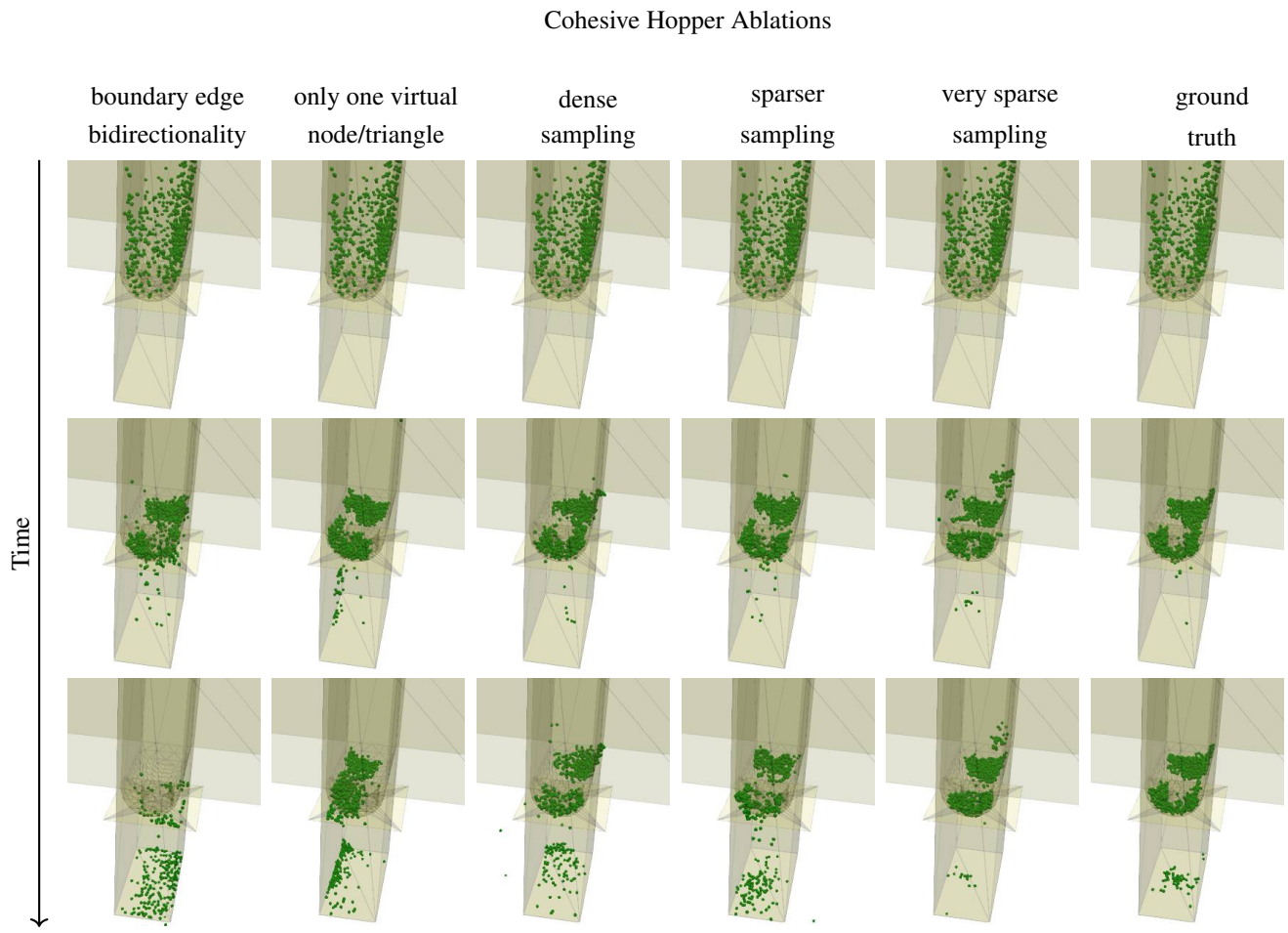

Figure TApp. D.2: Ablation experiments for cohesive particles within hopper geometries. Different ablations from our default architecture (first five columns, see text) are compared to the ground truth (last column). Particles are indicated by green spheres, triangular wall areas are yellow, the edges of these triangles are indicated by grey lines.

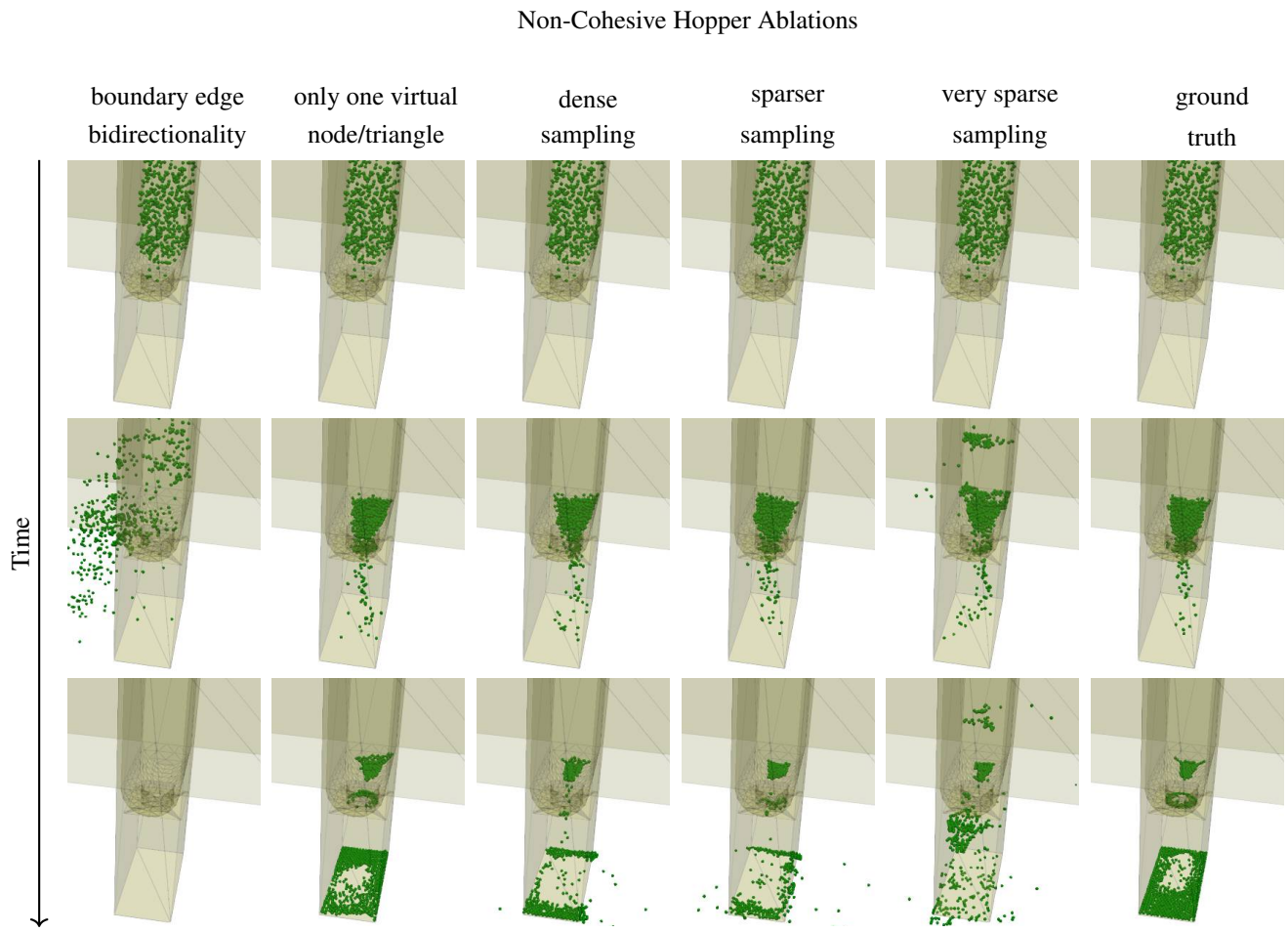

Figure TApp. D.3: Ablation experiments for non-cohesive particles within hopper geometries. Different ablations from our default architecture (first five columns, see text) are compared to the ground truth (last column). Particles are indicated by green spheres, triangular wall areas are yellow, the edges of these triangles are indicated by grey lines.

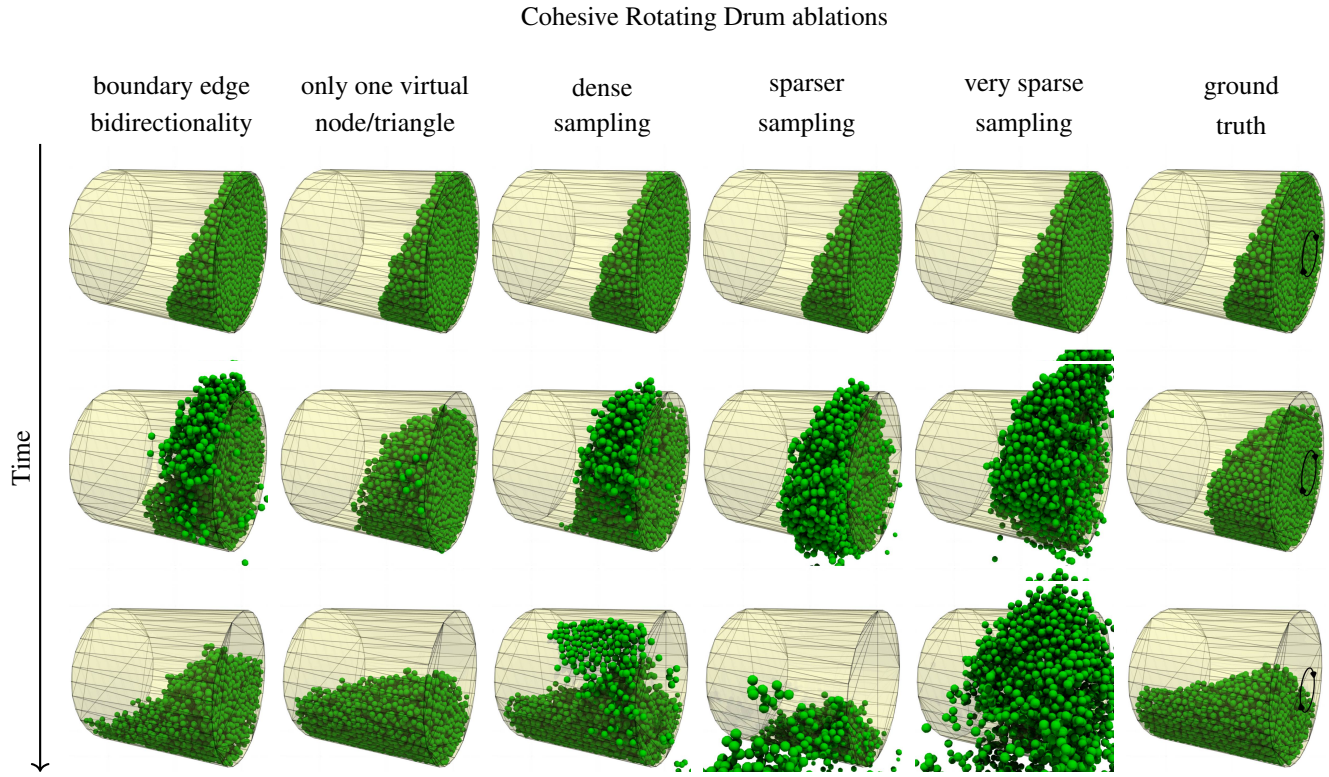

Figure TApp. D.4: Ablation experiments for cohesive particles within rotating drum geometries. Different ablations from our default architecture (first five columns, see text) are compared to the ground truth (last column). Particles are indicated by green spheres, triangular wall areas are yellow, the edges of these triangles are indicated by grey lines. The circular arrow indicates the rotation direction of the drum.

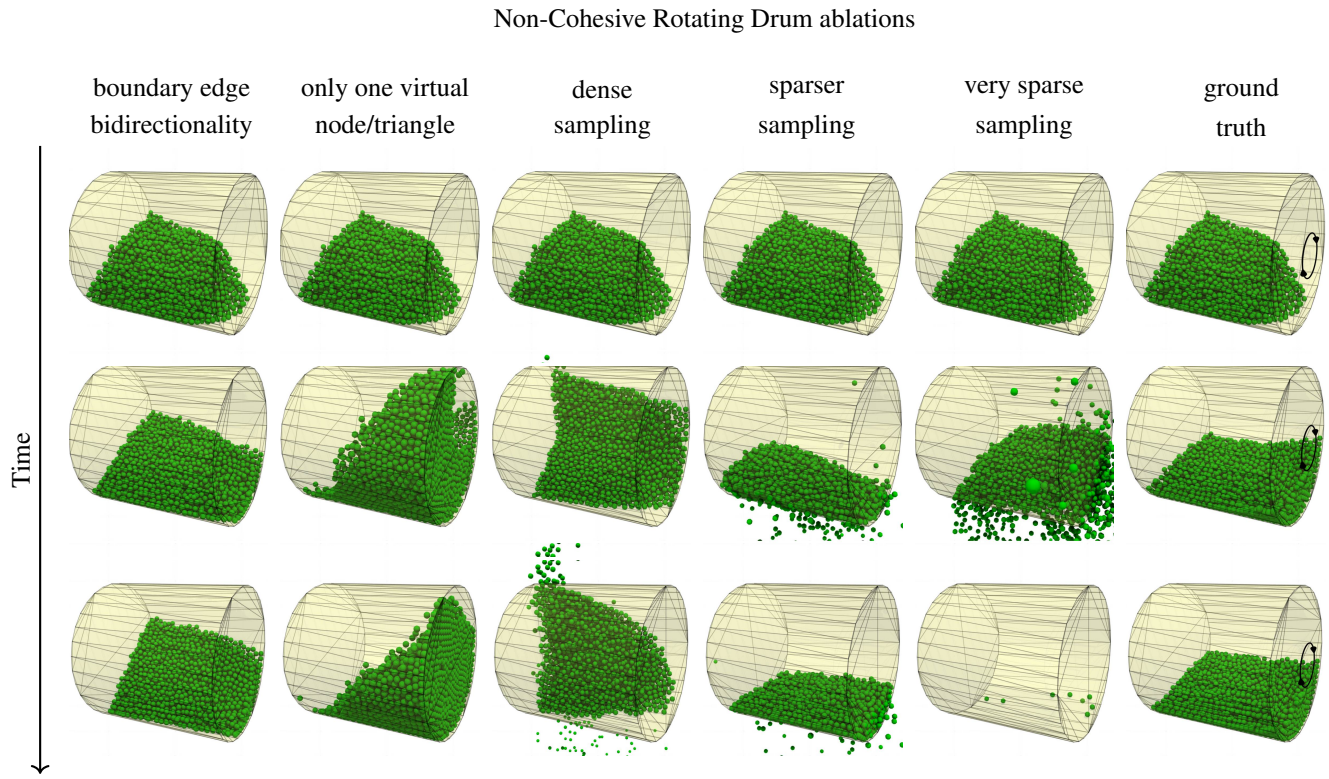

Figure TApp. D.5: Ablation experiments for non-cohesive particles withing rotating drum geometries. Different ablations from our default architecture (first five columns, see text) are compared to the ground truth (last column). Particles are indicated by green spheres, triangular wall areas are yellow, the edges of these triangles are indicated by grey lines. The circular arrow indicates the rotation direction of the drum.

TApp. References

Ba, J.; Kiros, J.; and Hinton, G. 2016. Layer normalization. arXiv:1607.06450.

Bonneel, N.; van de Panne, M.; Paris, S.; and Heidrich, W. 2011. Displacement Interpolation Using Lagrangian Mass Transport. *ACM Transactions on Graphics*, 30(6): 1–12.

Charles, R.; Su, H.; Kaichun, M.; and Guibas, L. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE national telecommunications conference*, 77–85.

Eberly, D. 1999. Distance Between Point and Triangle in 3D. Technical report, Geometric Tools.

Flamary, R.; Courty, N.; Gramfort, A.; Alaya, M.; Boisbunon, A.; Chambon, S.; Chapel, L.; Corenflos, A.; Fatras, K.; Fournier, N.; Gautheron, L.; Gayraud, N.; Janati, H.; Rakotomamonjy, A.; Redko, I.; Rolet, A.; Schutz, A.; Seguy, V.; Sutherland, D.; Tavenard, R.; Tong, A.; and Vayer, T. 2021. POT: Python Optimal Transport. *Journal of Machine Learning Research*, 22(78): 1–8.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Kahan, W. 1965. Pracniques: further remarks on reducing truncation errors. *Communications of the ACM*, 8(1): 40.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*. arXiv.

Klein, A. 2006. A Generalized Kahan-Babuška-Summation-Algorithm. *Computing*, 76(3-4): 279–293.

Kloss, C.; Aigner, A.; Mayrhofer, A.; and Goniva, C. 2017. fastDEM: A method for faster DEM simulations of granular media. In *Particles 2017*.

Kloss, C.; Goniva, C.; Hager, A.; Amberger, S.; and Pirker, S. 2012. Models, algorithms and validation for opensource DEM and CFD–DEM. *Progress in Computational Fluid Dynamics, an International Journal*, 12: 140–152.

Lim, D.; Robinson, J.; Zhao, L.; Smidt, T.; Sra, S.; Maron, H.; and Jegelka, S. 2022. Sign and Basis Invariant Networks for Spectral Graph Representation Learning. arXiv:2202.13013.

Mellmann, J.; Iroba, K.; Metzger, T.; Tsotsas, E.; Mészáros, C.; and Farkas, I. 2011. Moisture content and residence time distributions in mixed-flow grain dryers. *Biosystems Engineering*, 109(4): 297–307.

Nair, V.; and Hinton, G. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 807–814. Omnipress.

Plimpton, S. 1995. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1): 1–19.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R.; and Smola, A. 2017. Deep Sets. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.