Wasserstein Hypergraph Neural Network

Iulia Duta
Department of Computer Science
University of Cambridge
id366@cam.ac.uk

Pietro Liò
Department of Computer Science
University of Cambridge
pl219@cam.ac.uk

Abstract

The ability to model relational information using machine learning has driven advancements across various domains, from medicine to social science. While graph representation learning has become mainstream over the past decade, representing higher-order relationships through hypergraphs is rapidly gaining momentum. In the last few years, numerous hypergraph neural networks have emerged, most of them falling under a two-stage, set-based framework. The messages are sent from nodes to edges and then from edges to nodes. However, most of the advancement still takes inspiration from the graph counterpart, often simplifying the aggregations to basic pooling operations. In this paper, we are introducing Wasserstein Hypergraph Neural Network, a model that treats the nodes and hyperedge neighbourhood as distributions and aggregates the information using Sliced Wasserstein Pooling. Unlike conventional aggregators such as mean or sum, which only capture first-order statistics, our approach has the ability to preserve geometric properties like the shape and spread of distributions. This enables the learned embeddings to reflect how easily one hyperedge distribution can be transformed into another, following principles of optimal transport. Experimental results demonstrate that applying Wasserstein pooling in a hypergraph setting significantly benefits node classification tasks, achieving top performance on several real-world datasets.

1 Introduction

The potential to learn from relational data has substantially broadened the applicability of machine learning, extending its reach to a wide range of fields [1, 2, 3, 4, 5, 6, 7]. The flexibility of graph structures makes them well-suited for representing natural phenomena involving various types of interactions. However, while graphs are restricted to model pairwise connections, many real-world interactions involve more than two entities. To fill this gap, a generalisation of graphs called hypergraphs was introduced, allowing the representation of relationships among multiple elements.

More precisely, a hypergraph is characterised by a set of edges, where each edge connects a set of nodes, potentially of varying cardinality. The challenge of designing hypergraph networks becomes the challenge of properly modelling these sets. Many approaches [8, 9, 10] tackle this using a two-step process: first, the model aggregates information from the nodes within each hyperedge to compute a representation for that hyperedge. Then, it updates each node's representation using information from the hyperedges it belongs to. Both steps rely on methods designed to handle sets of elements.

Although set representation learning has seen significant progress in recent years [11], hypergraph networks still largely rely on sum-based aggregation methods such as Deep Sets [12] and Set Transformers [13]. Table 1 presents the update rules of several widely used hypergraph networks, emphasising that each of them utilises a form of the sum-based aggregator. Despite their strong theoretical foundation, these aggregators can struggle to effectively capture the full geometry of set-structured inputs [14].

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: New Perspectives in Advancing Graph Machine Learning.

Table 1: **Overview of the update rules used as aggregation steps** in various hypergraph neural networks from the literature. While theoretically powerful, summing can easily destroy all the geometric relationships between points. $\mathcal{N}_v(i)$ is the neighbourhood of node v_i of cardinality d_i, $\mathcal{N}_e(j)$ is the neighbourhood of edge e_j of cardinality d_j and ϵ, W_*, $\tilde{W}_*$ are learnable parameters.

Model	Hyperedge aggregation	Node aggregation
HGNN [15]	$h_e \leftarrow \sum_{i \in \mathcal{N}_e(e)} \frac{1}{\sqrt{d_i}} x_i W$	$x_i \leftarrow \frac{1}{\sqrt{d_i}} \sum_{e \in \mathcal{N}_v(i)} \frac{1}{d_e} h_e$
HCHA[1] [16]	$h_e \leftarrow \sum_{i \in \mathcal{N}_e(e)} \alpha_{e,i} x_i W$	$x_i \leftarrow \sum_{e \in \mathcal{N}_v(i)} \tilde{\alpha}_{i,e} h_e \tilde{W}$
UniGIN [10]	$h_e \leftarrow \sum_{i \in \mathcal{N}_e(e)} x_i$	$x_i \leftarrow \sum_{e \in \mathcal{N}_v(i)} h_e W + (1+\epsilon) x_i W$
ED-HNN [9]	$h_e \leftarrow \sum_{i \in \mathcal{N}_e(e)} \text{MLP}(x_i)$	$x_i \leftarrow \sum_{e \in \mathcal{N}_v(i)} \text{MLP}(x_i \| h_e)$
AllDeepSets [8]	$h_e \leftarrow \text{MLP}(\sum_{i \in \mathcal{N}_e(e)} \text{MLP}(x_i))$	$x_i \leftarrow \text{MLP}(\sum_{e \in \mathcal{N}_v(i)} \text{MLP}(h_e))$
AllSetTransformer[2] [8]	$h_e \leftarrow \sigma(\sum_{i \in \mathcal{N}_e(e)} (\alpha_i x_i W_v)$	$x_i \leftarrow \sigma(\sum_{e \in \mathcal{N}_v(i)} (\tilde{\alpha}_e h_e \tilde{W}_v)$

In this work, we are introducing Wasserstein Hypergraph Neural Networks (WHNN), a class of hypergraph models that uses Sliced Wasserstein Pooling (SWP) [14] as node and hyperedge aggregator. This pooling is based on the Wasserstein distance, an optimal transport metric which measures the distance between two distributions based on the cost of transporting mass from one to another.

We argue that this geometric information is highly relevant for hypergraph learning. Our experimental results support this claim, showing that WHNN not only outperforms traditional sum-based aggregation methods used in previous hypergraph models but also achieves superior performance compared to several strong hypergraph methods across a range of real-world datasets.

Our main contributions are summarised as follow: 1) We propose **a novel hypergraph architecture, the Wasserstein Hypergraph Neural Network (WHNN)**, which leverages Sliced Wasserstein Pooling for both node and hyperedge aggregation to more effectively capture the geometric structure of the feature space. 2) We empirically show that Wasserstein aggregation is highly effective for hypergraph representation, consistently **outperforming traditional sum-based methods** such as Deep Sets [12] and Set Transformers [13], regardless of the encoder used to process the nodes. 3) Wasserstein Hypergraph Neural Network achieves top results on multiple real-world datasets, highlighting the advantages of incorporating optimal transport into hypergraph processing.

2 Wasserstein Hypergraph Neural Network

A hypergraph is a tuple $\mathcal{H} = (V, E)$ where $V = \{v_1, v_2 \dots v_N\}$ is a set of nodes, and $E = \{e_1, e_2 \dots e_M\}$ is a set of hyperedges. Each node v_i is characterised by a feature vector $x_i \in \mathbb{R}^d$. The *neighbourhood of hyperedge* e_i is the set of nodes that are part of that hyperedge $\{v_j | v_j \in e_i\}$, while the *neighbourhood of a node* v_i is the set of all hyperedges containing that node $\mathcal{N}_{v_i} = \{e_j | v_i \in e_j\}$.

Our WHNN model follows the two-stage framework, by sending information from nodes to hyperedges and vice versa. For simplicity, this section only describes the nodes-to-hyperedges mechanism, as the hyperedge-to-node operation is entirely symmetrical. The pipeline is visually depicted in Figure 2 of the Appendix. For readability, the algorithm is presented sequentially for each hyperedge. However, our implementation processes all hyperedges in parallel.

First, we will project the node features into a more expressive representation. Each hyperedge is then associated with a probability distribution, with its constituent nodes treated as samples. These distributions are embedded using a Wasserstein-based aggregator to obtain the final hyperedge representations. Then, the hyperedge representations are fed into the hyperedges-to-nodes stage. Below, we elaborate on each of these stages.

[1]The coefficients $\alpha_{e,i}$ used in summations are scalars predicted as $MLP(x_i || h_e)$

[2]The function σ is a combination of residual connections and layer normalisations, while $\alpha_i = (\theta W_q)(x_i W_k)^T$ with θ, W_q and W_k as learnable parameters.

ALGORITHM 1: One Layer of Wasserstein Hypergraph Neural Network[3]

input: *node features X of hypergraph $\mathcal{H}$ and ref. distribution q*
output: *updated node features $\tilde{X}$*
procedure WHNN($X, \mathcal{H}, q$)

$X_0 \leftarrow X$
 # Sample reference sets
$Q_v, Q_e \leftarrow sample(q)$
 # Extract node and edge neighbourhood
$\mathcal{N}_v, \mathcal{N}_e \leftarrow$ neighbourhoods($\mathcal{H}$)
 # Node to hyperedge
$X \leftarrow encoder(X)$
$Z \leftarrow Wasserstein(X, \mathcal{N}_v, Q_v)$
 # Hyperedge to node
$Z \leftarrow encoder(Z)$
$X \leftarrow Wasserstein(Z, \mathcal{N}_e, Q_e)$
 # Residual connection
$\tilde{X} \leftarrow \alpha X + (1-\alpha)X_0$
return $\tilde{X}$

[3] For simplicity in handling shapes, we assume encoders that are independent of the hyperedge.

ALGORITHM 2: Wasserstein aggregator

input: *entity features X; list of neighbourhoods to aggregate $\mathcal{N}$; samples from reference distribution Q*
output: *aggregated neighbourhoods Z*
procedure WASSERSTEIN($X, \mathcal{N}, Q$)

 # Project entities into slices
$X \leftarrow X\Theta$
 # Sort the samples from the reference distr.
$Q \leftarrow$ sort(Q)
for all neighbourhoods $S \in \mathcal{N}$
 # Extract elements in the neighbourhood
$X_s \leftarrow \{x_i\}_{i \in S}$
 # If $|X_s| \neq |Q|$ interpolate X_s to match size
$X'_s \leftarrow$ interpolate(X_s)
 # Sort the elements of the neighbourhood.
$X'_s \leftarrow$ sort(X'_s)
 # Compute the dist that approx Wass dist
$Z_{s:} \leftarrow Q - X'_s$
 # Combine the slices
$Z \leftarrow ZW$
return Z

Node encoder. The goal of this module is to enhance the representation of node features by projecting them into a more informative space. We are experimenting with two types of encoders: an *edge-independent* one where the node is carrying the same representation in each hyperedge it is contained, and an *edge-dependent* one which takes into account pairwise interactions.

The edge-independent encoder is a simple MLP, which is applied in parallel for each node. This way, a node i is characterised by the same feature vector in each hyperedge e it is part of.

$$\tilde{x}_i^e = \text{MLP}(x_i)$$

On the other hand, for the edge-dependent encoder, each node has a different representation in each hyperedge it is part of. To achieve this, for each hyperedge, we are using a Set Attention Block layer (SAB) as introduced in [13], which propagates the information between each pair of two nodes contained in that hyperedge. The full version of the block acts as follows:

$$z_i^e = \sigma(x_i + \sum_{j \in e}(x_i W_q)(x_j W_k)^T(x_j W_v))$$
$$\tilde{x}_i^e = \sigma(z_i^e + \text{MLP}(z_i^e)),$$

where σ denote layer normalisation and W_k, W_q and $W_v \in \mathbb{R}^{d \times d}$ are learnable parameters.

Hyperedges as probability distributions. Unlike traditional hypergraph approaches that treat a hyperedge as a set of nodes, we model a hyperedge as a probability distribution, with its constituent nodes being samples drawn from that distribution. This way, the hyperedges are not only characterised by the combination of their elements, but by the regions of the space where their elements are situated. The nodes became prototypes of the hyperedge behaviour.

For example, a hypergraph containing two clusters of nodes suggests a bimodal underlying distribution. On the other hand, a hyperedge where nodes are close in the feature space denotes a unimodal probability distribution, suggesting a homophilic behaviour. A hyperedge in which nodes have similar representations indicates a low-variance distribution, while a hyperedge with diverse nodes suggests

a more uniform distribution. We consider these elements essential to capture; therefore, we design an aggregator with the appropriate inductive bias to do so.

Let's consider p_i the probability distribution where the elements of the hyperedge e_i are sampled from. In other words, we assume each node $v_j \in e_i$ is sampled as $\tilde{x}_j^i \in \mathbb{R}^d \sim p_i$. The goal is to obtain hyperedge embeddings that preserve the geometric information of this underlying distribution, such as spreading, shape etc. See Figure 2 in the Appendix for a visual representation of this structure.

Note that, by treating nodes as sampled from an underlying distribution, we make the assumption that other unobserved nodes drawn from the same distribution are also likely to belong to the same hyperedge. This probabilistic interpretation proved to be powerful for set representation learning [14], and our experiments demonstrate that hypergraph models can benefit from it as well.

Wasserstein aggregator. Interpreting hypergraphs as a collection of probability distributions enables us to derive more powerful similarity metrics between hyperedges. As shown in the previous section, most of the current hypergraph architectures rely on mean pooling to create hyperedge embeddings from node representations. However, from a probabilistic perspective, averaging compares distributions only based on their means. For complex data distributions, this approach fails to capture the full underlying geometry. While models relying on summation, such as Deep Sets [12] have been proven to be universal approximators, they heavily rely on the internal node encoder (an MLP) to map input features into a space where first-order statistics like the mean effectively approximate the distribution. In the hypergraph setting, where multiple sets interact in complex ways, this is hard to achieve.

This motivates us to adopt Sliced Wasserstein Pooling [14] to encode the hyperedge distributions. Concretely, for each hyperedge e, given the node embeddings of all the nodes in the hyperedge $\{\tilde{x}_i^e\}_{i \in e}$, we are aggregating them using the Sliced Wasserstein Pooling to obtain a vectorial hyperedge representation: $h_e = \mathcal{SWP}(\{\tilde{x}_i^e\}_{i \in e})$. The algorithm works as follows:

1. **Step 1**: Select a reference hyperedge distribution q and sample N points $\{y_i\}_{i=1}^N \sim q$. Choose a set of directions $\{\theta_l\}_{l=1}^L$ with $\theta_l \in \mathbb{R}^{d \times 1}$ used as projection slices in the pooling process. Note that, in order to obtain comparable embeddings across the entire hypergraph, we share the same reference distribution and the same set of slices for all hyperedges.
2. **Step 2**: Project each node representation $\tilde{x}_i^e$ into each slice θ_l as follow: $z_i^{e,\theta_l} = (\tilde{x}_i^e)^T \theta_l \in \mathbb{R}$. Since the algorithm requires the same number of sampled nodes from both the hyperedge distribution and the reference, when the cardinality of the hyperedge $|e| \neq N$, we increase/decrease the number of nodes in e using linear interpolation. $z_i^{e,\theta_l} \leftarrow \text{interp}(z_i^{e,\theta_l}, N)$
3. **Step 3**: For each hyperedge, for each slice, compute the distance between the node representations and the reference points. $h_e^{\theta_l} = ||z_{\pi(i)}^{e,\theta_l} - y_{\tilde{\pi}(i)}||$, where z_π^{e,θ_l} and $y_{\tilde{\pi}}$ represent the vectors in sorted order. The final hyperedge embedding is obtained as a weighted mean of these embeddings: $h_e = \sum_{l=1}^L (w_l h_e^{\theta_l})$, where w_l are learnable scalars combining the slices.

The process is also described in Algorithm 2. The directions θ_l and the reference distribution can be either fixed or learnable. For a more detailed explanation of the Wasserstein distance and the Sliced Wasserstein Pooling as applied in the set representation learning, see Section C.2 of the Appendix.

Intuitively, each hyperedge is represented by a vector which measures how difficult it is to transform the *hyperedge distribution* into the reference distribution [4]. Note that these reference distributions act only as shared anchors, similar to the origin in Euclidean space.

The true strength of Wasserstein embeddings is not observed in isolation, but lies in their ability to capture the relative distances between different entities (nodes/hyperedges). Following the theoretical properties of Sliced Wasserstein Pooling [14], the Euclidean distance between two hyperedge representations measures the cost of transforming one hyperedge distribution into another. Similarly, the Euclidean distance between two node embeddings measures how easy it is to map one node neighbourhood into the other node neighbourhood.

This type of information is particularly important in hypergraph learning, as it reflects the extent of change required to transform the characteristics of one group to resemble those of another. In the context of node classification, this means that if the neighbourhoods of two nodes are similar in

[4] As defined above, by hyperedge distribution we denote the distribution of nodes in the hyperedge.

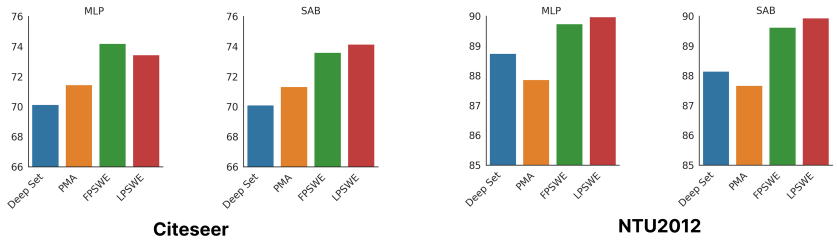

Figure 1: **Ablation on the importance of Wasserstein aggregator**. We test 2 versions of Sliced Wasserstein Pooling: with fixed (FPSWE) or learnable (LPSWE) reference distribution. Wasserstein aggregators outperform both Deep Sets and PMA, commonly used inside hypergraph models.

distribution, the nodes are likely to share the same label. In contrast, average pooling tends to assign the same label to nodes whose neighbourhoods have similar average characteristics.

Edge to node step. For simplicity, we only described in detail the first stage of the framework, which sends messages from nodes to hyperedges. The second stage of the framework, which creates node representation by aggregating the information from neighbouring hyperedges, is done in a similar way, only with different parameters. In conclusion, we not only capture the structural relationship between hyperedges, but also the structural relationship between nodes' neighbourhood.

3 Experiments

Our main goal is to understand to what extent Wasserstein aggregation is beneficial for hypergraph neural networks. Additionally, we investigate how the choice of node encoder (whether edge-dependent or edge-independent) affects overall performance. Finally, we compare our model against a range of strong baseline methods from the existing literature.

Datasets. We evaluate our model on the node-classification task. We select seven real-world datasets that vary in domain and scale. These include Cora, Citeseer, Cora-CA, DBLP-CA [17], ModelNet40 [18], NTU2012 [19] and 20News [20]. Among the datasets that are usually used for benchmarking hypergraph models [17], we omitted Pubmed due to the high percentage of isolated nodes (80.5%), which makes the relational processing unnecessary. For a fair comparison with the other methods, we follow the training procedures employed by [9], randomly splitting the data into 50% training, 25% validation and 25% test samples. In the Appendix B, we also offer additional results on Senate, Congress [21] and House [22].

Importance of Wasserstein aggregator. Our main contribution consists of adopting Sliced Wasserstein Pooling as a powerful aggregator inside the hypergraph networks. As described in the previous section, while most of the existing methods used variations of the sum pooling to aggregate the information from each node and each hyperedge neighbourhood, our Wasserstein aggregator presents a more in-depth understanding of the neighbourhood distribution, having the inductive bias to capture subtle differences, such as the difference in shape or spread.

To understand to what extent this is contributing to a better hypergraph representation for real-world scenarios, we design an ablation study in which we keep the underlying architecture fixed and only modify the aggregator. Concretely, we are using as aggregators either Deep Set (as used by AllDeepSet [8] and ED-HNN [9] models) or the PMA module (as in AllSetTransformer [8] model). For our Wasserstein aggregator, we experiment with both a fixed-reference distribution (denoted as FPSWE) or with learnable reference distribution (denoted as LPSWE). For a robust evaluation, we are comparing these aggregators using both the edge-independent encoder (MLP) and the edge-dependent one (SAB). The results on Citeseer and NTU2012 datasets are reported in Figure 1.

Regardless of the encoder and the dataset we are testing on, both Wasserstein aggregators are consistently outperforming both the Deep Sets and the PMA aggregators by a significant margin. A learnable reference seems to be beneficial however, the improvement is generally marginal. Additional experiments on other datasets show a similar trend and are provided in the Appendix B.

Importance of edge-dependent encoder. The node and hyperedge encoder transforms features into a space where their distribution within each hyperedge captures meaningful information about the group. As stated in the model description, we equipped our model with two types of encoders.

Table 2: **Performance on a collection of hypergraph datasets.** Our model using SWP as a node and hyperedge aggregator shows superior results. We test our model in both its variants: with edge-independent (MLP) and edge-dependent encoder (SAB). Both options are exhibiting competitive performance. We mark the **first**, second and *third* best performing models for each dataset.

Name	Cora	Citeseer	Cora_CA	DBLP_CA	ModelNet40	NTU2012	20News
HCHA	79.14 ± 1.02	72.42 ± 1.42	82.55 ± 0.97	90.92 ± 0.22	94.48 ± 0.28	87.48 ± 1.87	80.33 ± 0.80
HNHN	76.36 ± 1.92	72.64 ± 1.57	77.19 ± 1.49	86.78 ± 0.29	97.84 ± 0.25	89.11 ± 1.44	81.35 ± 0.61
HyperGCN	78.45 ± 1.26	71.28 ± 0.82	79.48 ± 2.08	89.38 ± 0.25	75.89 ± 5.26	56.36 ± 4.86	81.05 ± 0.59
HyperGNN	79.39 ± 1.36	72.45 ± 1.16	82.64 ± 1.65	91.03 ± 0.20	95.44 ± 0.33	87.72 ± 1.35	80.33 ± 0.42
AllDeepSets	76.88 ± 1.80	70.83 ± 1.63	81.97 ± 1.50	91.27 ± 0.27	96.98 ± 0.26	88.09 ± 1.52	81.06 ± 0.54
AllSetTransformers	78.58 ± 1.47	73.08 ± 1.20	83.63 ± 1.47	91.53 ± 0.23	*98.20 ± 0.20*	88.69 ± 1.24	*81.38 ± 0.58*
UniGCNII	78.81 ± 1.05	73.05 ± 2.21	83.60 ± 1.14	91.69 ± 0.19	98.07 ± 0.23	89.30 ± 1.33	81.12 ± 0.67
ED-HNN	80.31 ± 1.35	*73.70 ± 1.38*	*83.97 ± 1.55*	91.90 ± 0.19	97.75 ± 0.17	*89.48 ± 1.87*	81.36± 0.55
WHNN_MLP	*79.84 ± 1.56*	74.79 ± 1.19	84.12 ± 1.94	*91.73 ± 0.24*	98.47 ± 0.19	**90.87 ± 1.59**	**81.83 ± 0.68**
WHNN_(I)SAB	**80.72 ± 1.96**	**74.92 ± 1.60**	**84.62 ± 1.77**	**91.99 ± 0.33**	**98.54 ± 0.21**	90.68 ± 1.68	81.42 ± 0.60

An edge-independent module represented by an MLP, and an edge-dependent encoder represented by a self-attention block (SAB). While the MLP is processing information independently for each node/hyperedge, SAB is capturing pairwise interactions between nodes/hyperedges sharing a neighbourhood. The results in Figure 1 and Table 2 show similar results among the encoders, with the edge-dependent one being slightly more powerful. However, this comes with the cost of a more expensive model, as the edge-dependent encoder requires more memory to store the representation for all incident pairs (node, hyperedge). To alleviate that on the larger datasets (20News and DBLP), we replace the SAB block with the ISAB low-rank approximation introduced by [13].

Comparison with baselines. In Table 2, we are comparing against a series of hypergraph networks from the literature. With respect to aggregation strategies, HNHN [23], HyperGNN [15], AllDeepSets [8], UniGCNII [10] and ED-HNN [9] use variations of Deep Sets to aggregate the information, HyperGCN [17] uses a max aggregator, while HCHA [16] and AllSetTransformer [8] use an attention-based weighted summation. Regardless of the encoder used, our model consistently obtains top results, outperforming the other methods on all datasets. This demonstrates the advantages of using Wasserstein aggregators for higher-order processing. While we integrated this aggregator into a standard two-stage framework, many existing models from the literature can be adopted to take advantage of this type of geometric-inspired aggregation.

Implementation details. In all experiments, we train our models using Adam for 500 epochs, on a single GPU NVIDIA Quadro RTX 8000 with 48GB of memory. Each model is trained 10 times with different random splits and different initialisations. We report average accuracy along with the standard deviation. The results represent the best performance obtained by each architecture using hyper-parameter optimisation with random search. Details about all the hyperparameters can be found in the Appendix. For the ablation study, the architecture is fixed to ensure a fair comparison.

We use a number of Wasserstein slices equal to the hidden dimension, and we experiment with both learning the reference set or not. In all experiments, we are using a uniform distribution as a reference and vary the number of points sampled. In the Appendix B, we offer an additional set of experiments demonstrating that, as expected, the type of reference distribution is not essential, while the number of reference points should be large enough to cover the complexity of the hyperedge set.

These experimental results show that aggregating node and hyperedge neighbourhoods using Sliced Wasserstein Pooling is highly effective for hypergraph processing, the Wasserstein aggregator consistently outperforming standard methods like Deep Sets and PMA.

4 Conclusion

In this work, we introduce Wasserstein Hypergraph Neural Networks (WHNN), a model for processing hypergraph structures. The model relies on Sliced Wasserstein Pooling to aggregate the nodes into hyperedge representations and vice versa. This design choice, inspired by optimal transport literature, enables us to capture more information about the internal structure of the neighbourhoods, preserving more geometric relations between elements. The experimental results on various datasets demonstrate that this Wasserstein aggregator is effective for modelling higher-order interactions, outperforming traditional aggregators, making WHNN a promising tool for hypergraph representation learning.

Acknowledgment. The authors would like to thank Daniel McFadyen and Alex Norcliffe for fruitful discussions and constructive suggestions during the development of the paper. I.D. additionally acknowledges the support provided by Robinson College during this period.

References

[1] Ruth Johnson, Michelle M. Li, Ayush Noori, Owen Queen, and Marinka Zitnik. Graph ai in medicine. *CoRR*, abs/2310.13767, 2023.

[2] Catherine Tong, Emma Rocheteau, Petar Veličković, Nicholas Lane, and Pietro Lio. *Predicting Patient Outcomes with Graph Representation Learning*, pages 281–293. 01 2022.

[3] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 8459–8468, 2020.

[4] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. Graphcast: Learning skillful medium-range global weather forecasting, 2023.

[5] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272, 2017.

[7] Kexin Huang, Cao Xiao, Lucas M Glass, Marinka Zitnik, and Jimeng Sun. Skipgnn: predicting molecular interactions with skip-graph networks. *Scientific reports*, 10(1):1–16, 2020.

[8] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations*, 2022.

[9] Peihao Wang, Shenghao Yang, Yunyu Liu, Zhangyang Wang, and Pan Li. Equivariant hypergraph diffusion neural operators. *arXiv preprint arXiv:2207.06680*, 2022.

[10] Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021.

[11] Jiahao Xie and Guangmo Tong. Advances in set function learning: A survey of techniques and applications, 2025.

[12] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[13] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019.

[14] Navid Naderializadeh, Joseph F Comer, Reed Andrews, Heiko Hoffmann, and Soheil Kolouri. Pooling by sliced-wasserstein embedding. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3389–3400. Curran Associates, Inc., 2021.

[15] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. *Proc. Conf. AAAI Artif. Intell.*, 33(01):3558–3565, July 2019.

[16] Song Bai, Feihu Zhang, and Philip H.S. Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021.

[17] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[18] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015.

[19] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. *Comput. Graph. Forum*, 22:223–232, 09 2003.

[20] Tom Mitchell. Twenty Newsgroups. UCI Machine Learning Repository, 1997. DOI: https://doi.org/10.24432/C5C323.

[21] James H. Fowler. Legislative cosponsorship networks in the US house and senate. *Social Networks*, 28(4):454–465, oct 2006.

[22] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances*, 7(28).

[23] Yihe Dong, Will Sawin, and Yoshua Bengio. Hnhn: Hypergraph networks with hyperedge neurons. In *Graph Representation Learning and Beyond Workshop at ICML 2020*, June 2020. Code available: https://github.com/twistedcubic/HNHN.

[24] Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021.

[25] Abihith Kothapalli, Ashkan Shahbazi, Xinran Liu, Robert Sheng, and Soheil Kolouri. Equivariant vs. invariant layers: A comparison of backbone and pooling for point cloud classification, 2024.

[26] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and radon wasserstein barycenters of measures. 51(1):22–45, January 2015.

[27] Bohan Tang, Zexi Liu, Keyue Jiang, Siheng Chen, and Xiaowen Dong. Hypergraph node classification with graph neural networks. *CoRR*, abs/2402.05569, 2024.

[28] Jiying Zhang, Yuzhao Chen, Xiong Xiao, Runiu Lu, and Shutao Xia. Learnable hypergraph laplacian for hypergraph learning. In *ICASSP*, 2022.

[29] Lev Telyatnikov, Maria Sofia Bucarelli, Guillermo Bernardez, Olga Zaghen, Simone Scardapane, and Pietro Lio. Hypergraph neural networks through the lens of message passing: A common perspective to homophily and architecture design. *Transactions on Machine Learning Research*, 2025.

[30] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Peter Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-mixer: An all-MLP architecture for vision. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[31] Yijia Zheng and Marcel Worring. Co-representation neural hypergraph diffusion for edge-dependent node classification, 2025.

[32] Minyoung Choe, Sunwoo Kim, Jaemin Yoo, and Kijung Shin. Classification of edge-dependent labels of nodes in hypergraphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 298–309. ACM, August 2023.

[33] Maolin Wang, Yaoming Zhen, Yu Pan, Yao Zhao, Chenyi Zhuang, Zenglin Xu, Ruocheng Guo, and Xiangyu Zhao. Tensorized hypergraph neural networks, 2024.

[34] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.

[35] Konstantinos Skianis, Giannis Nikolentzos, Stratis Limnios, and Michalis Vazirgiannis. Rep the set: Neural networks for learning set representations. *ArXiv*, abs/1904.01962, 2019.

[36] Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. FSPool: Learning set representations with featurewise sort pooling. 2019.

[37] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[38] Khai Nguyen, Nhat Ho, Tung Pham, and Hung Bui. Distributional sliced-wasserstein and applications to generative modeling. In *International Conference on Learning Representations*, 2021.

[39] Charlie Frogner, Farzaneh Mirzazadeh, and Justin Solomon. Learning embeddings into entropic wasserstein spaces, 2019.

[40] Trung Nguyen, Quang-Hieu Pham, Tam Le, Tung Pham, Nhat Ho, and Binh-Son Hua. Point-set distances for learning representations of 3d point clouds, 2021.

[41] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler–lehman graph kernels. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 6436–6446. Curran Associates, Inc., 2019.

[42] Soheil Kolouri, Navid Naderializadeh, Gustavo K. Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. In *International Conference on Learning Representations*, 2021.

[43] Grégoire Mialon, Dexiong Chen, Alexandre d'Aspremont, and Julien Mairal. A trainable optimal transport embedding for feature aggregation and its relationship to attention, 2021.

[44] Nicolas Courty, Rémi Flamary, and Mélanie Ducoffe. Learning wasserstein embeddings. In *International Conference on Learning Representations*, 2018.

Appendix: Wasserstein Hypergraph Neural Network

This appendix contains details related to our model, including potential limitations and future work, additional datasets for the ablation experiments, details on the hyperparameters used in our experiments and derivation of the computational complexity. The content is structured as follows:

- **Section A** highlights a series of potential limitations that can be addressed to improve the current work, together with a discussion on potential future work.
- **Section B** presents additional experiments used as ablation for our model.
- **Section C** provides a brief introduction into Wasserstein distance.
- **Section D** presents an extensive related work.
- **Section E** presents the list of hyperparameters used in our experiments.
- **Section F** derives the computational complexity of our model.

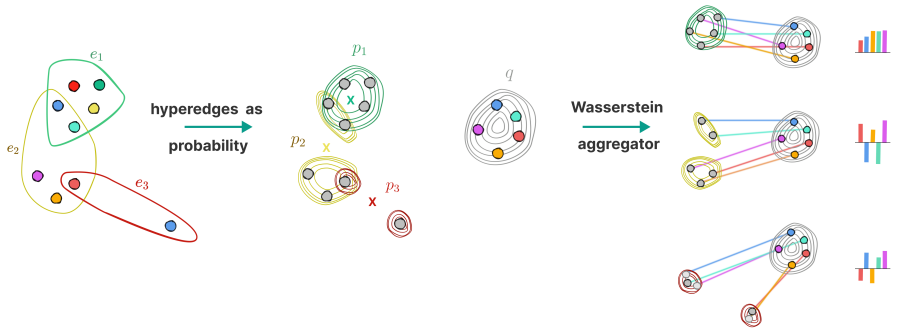

Figure 2: **One stage (node-to-hyperedge) of WHNN pipeline** designed to be more sensitive to the geometric structure of the hyperedge compared to the traditional aggregators. The hypergraphs are viewed as a collection of probability distributions $\{p_i\}$, one for each hyperedge, with the observed nodes treated as samples drawn from it. An additional distribution q is picked as a reference. Finally, the Sliced Wasserstein Pooling is adopted as an aggregation method: each hyperedge is represented by its Sliced Wasserstein distance to a reference distribution.

A Limitations and Future work

As discussed in the main paper, we treat the neighbourhood of each node as a sample from an underlying probability distribution. This approach assumes that any additional nodes drawn from this distribution should belong to the same neighbourhood as the observed ones. This aligns with the intuition that elements within a group should share common characteristics. While the datasets we used support this assumption, there may be real-world scenarios where it does not hold. Our model relies solely on the node encoder to project features into a space where the assumption is approximately valid.

Moreover, due to this continuous view of the neighbourhood (as a distribution of probability) together with the interpolation step, the current model may lose information about the exact cardinality of the neighbourhoods. In situations where neighbourhood size is important, we recommend encoding it as an explicit feature. However, we mention that this is an issue we share with the mean-based pooling algorithms.

The main goal of this paper is to highlight the benefits of using geometrically-inspired poolings for aggregating neighbourhood information in hypergraphs. While we focused entirely on hypergraphs, a similar idea can be applied on graph neural networks or other topological structures to aggregate

messages coming from each node's neighbourhood. As future work, it would be interesting to see to what extent these models can benefit from Wasserstein aggregators.

Moreover, while the proposed model integrates the Wasserstein aggregator into a standard two-stage pipeline, several other architectures, such as ED-HNN which uses summation as an aggregator, might benefit from adopting it. We are leaving this investigation as future work.

B Additional experiments

Importance of Wasserstein aggregator. Due to space constraints, in the main paper, we only included ablation studies on Citeseer and NTU datasets. Here we report additional results for Cora_CA and ModelNet40 datasets (Figure 3) together with the numerical results (Table 3).

For each experiment, we kept the architecture fixed and modified the aggregator used in the two stages to be either Deep Set, PMA, and the learnable (LPSWE) or fixed (FPSWE) Wasserstein aggregator. The results are similar across the datasets, with Wasserstein Pooling proving to be beneficial compared to Deep Sets and PMA. In terms of encoder type, we noticed that, in some cases, for a fixed architecture, SAB tends to model the distribution better than MLPs.

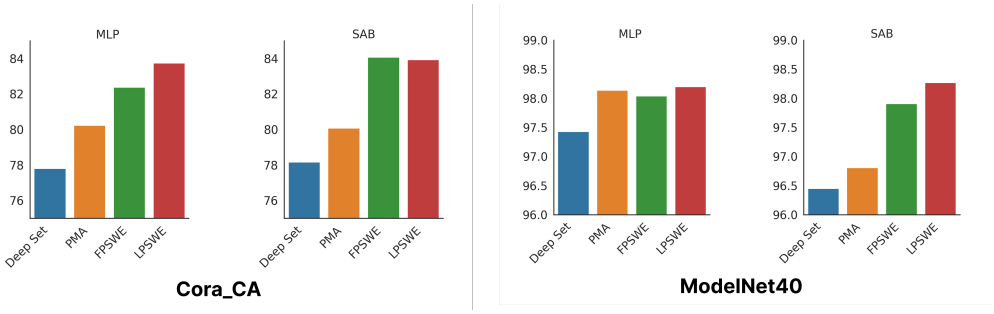

Figure 3: **Additional results for the ablation study on the importance of Wasserstein aggregator for hypergraph representation learning** Cora_CA and ModelNet datasets. FPSWE denotes the Wasserstein aggregator with fixed reference while LPSWE denotes the Wasserstein aggergator with learnable reference distribution. Regardless of the encoder used to project the nodes and hyperedges, the Wasserstein aggregators outperform both the Deep Sets and PMA commonly used inside hypergraph models.

Table 3: **Numerical results for the ablation study** comparing DeepSet, PMA and our Wasserstein aggregator with fixed (FPSWE) or learnable (LPSWE) references.

Model	MLP encoder				SAB encoder			
	Citeseer	NTU	Cora_ca	ModelNet	Citeseer	NTU	Cora_ca	ModelNet
DeepSet	70.14 ± 0.69	88.75 ± 1.88	77.81 ± 2.03	97.43 ± 0.29	70.10 ± 0.57	88.15 ± 1.32	78.17 ± 1.02	96.45 ± 0.42
PMA	71.45 ± 0.48	87.87 ± 1.79	80.23 ± 0.81	98.14 ± 0.26	71.33 ± 0.86	87.67 ± 1.53	80.08 ± 1.35	96.81 ± 0.33
FPSWE	74.20 ± 0.66	89.74 ± 1.65	82.39 ± 1.27	98.04 ± 0.32	73.60 ± 0.99	89.62 ± 1.61	84.07 ± 1.23	97.91 ± 0.24
LPSWE	73.45 ± 0.81	89.98 ± 1.62	83.75 ± 1.74	98.20 ± 0.27	74.15 ± 0.99	89.94 ± 1.59	83.93 ± 1.68	98.27 ± 0.31

Influence of the reference distribution and the number of samples. For our nodes and hyperedge embeddings, the reference distribution only acts as a common anchor, similar to the origin in an Euclidean space. On its own, each set embedding (node/hyperedge representation) contains information about how different the underlying distribution is compared to the same reference distribution. However, when computing the relative distance between two sets (two nodes, two hyperedges), the reference distribution cancels out, and we obtain information about how one hyperedge can be transformed into another (regardless of the shape and characteristics of the reference distribution). Because of that, we expect the choice of reference distribution to have minimal impact on the performance, mostly attributed to numerical stability.

To validate this, we design a series of experiments in which we modify the shape of the reference distribution and the number of samples used to represent it.

Table 4: **Ablation study comparing performance when varying the type of reference distributions and the number of points sampled from them.** Since the reference distribution serves only as a shared anchor for all sets, it has little impact on final accuracy. The empirical results on three datasets confirm this intuition.

(a) Ablation study on Citeseer dataset.

Distr.	2 ref	5 ref	10 ref	25 ref	50 ref
Uniform	74.55 ± 1.68	74.88 ± 1.59	74.72 ± 1.63	74.69 ± 1.77	74.84 ± 1.73
Normal	74.51 ± 1.73	74.83 ± 1.61	74.71 ± 1.65	74.71 ± 1.79	74.81 ± 1.72
Poisson	74.63 ± 1.54	74.81 ± 1.53	74.70 ± 1.66	74.66 ± 1.79	74.83 ± 1.73
Learnable	74.62 ± 1.55	74.60 ± 1.41	74.85 ± 1.54	74.68 ± 1.75	74.85 ± 1.69

(b) Ablation study on Cora_CA dataset

Distr.	2 ref	5 ref	10 ref	25 ref	50 ref
Uniform	84.38 ± 1.51	84.74 ± 1.60	84.63 ± 1.63	84.56 ± 1.71	85.03 ± 1.82
Normal	84.19 ± 1.79	84.69 ± 1.57	84.53 ± 1.65	84.50 ± 1.80	84.75 ± 1.79
Poisson	84.12 ± 1.80	84.51 ± 1.61	84.63 ± 1.66	84.49 ± 1.73	84.70 ± 1.71
Learnable	84.29 ± 1.94	84.51 ± 1.49	84.50 ± 1.35	84.62 ± 1.63	84.62 ± 1.75

(c) Ablation study on NTU dataset.

Distr.	2 ref	5 ref	10 ref	25 ref	50 ref
Uniform	90.19 ± 1.38	90.21 ± 1.60	90.67 ± 1.17	90.47 ± 1.31	90.80 ± 1.21
Normal	90.18 ± 1.37	90.07 ± 1.57	90.59 ± 1.50	91.01 ± 1.46	90.41 ± 1.47
Poisson	90.19 ± 1.38	90.25 ± 1.61	90.50 ± 1.49	90.39 ± 1.02	90.41 ± 1.87
Learnable	90.51 ± 1.39	90.50 ± 1.38	90.52 ± 1.26	90.62 ± 1.37	90.58 ± 1.06

In the first experiment, we pick the reference distribution to be either uniform, Gaussian, Poisson or a learned distribution. For the learnable distribution, we consider the samples from the distribution to be learnable parameters.

The results in Table 4 suggest that the choice of distribution has little effect, with only slight variations observed in the NTU datasets and for the learnable distribution, differences likely due to computational stability issues.

In the second set of experiments, we run the same setup as before, but modify the number of sampled points from the reference distribution. If all sets have the same cardinality, the standard approach is to select a number of reference points equal to this cardinality (thus avoiding the need for linear interpolation). However, this is not possible in the hypergraph domain, where hyperedges tend to have various cardinalities. Thus, we expect the model to perform well as long as we pick the number of reference points to be comparable to most of the cardinalities. The experiments in Table 4 show a small drop in performance for very few reference points (when $M = 2$ for Citeeser and Cora_CA datasets, and $M \in \{2, 5\}$ for NTU), with comparable performance otherwise.

Experiments on additional datasets. In the main paper, we tested our model on seven benchmarks usually used in the hypergraph literature. Here we provide additional results on three more datasets, Senate, Congress [21] and House [22]. Compared to the previous ones, for these datasets, the nodes are not equipped with features, so we adopt the usual setup in which synthetic features are generated using Gaussian noise [9]. However, this limitation of the benchmarks makes it harder to interpret or understand the input space we want to model.

The results in Table 5 show a consistent trend with the other benchmarks: even when the feature space is synthetically generated, the Wasserstein aggregator enhances the representations and yields improved performance.

Table 5: **Performance comparison on Congress, Senate, and House datasets.** The WHNN model, which improves the message passing hypergraph architecture with a Wasserstein aggregator, leads to better results, clearly overcoming the DeepSet-based models.

Model	Congress	Senate	House
HCHA	90.43 ± 1.20	48.62 ± 4.41	61.36 ± 2.53
HNHN	53.35 ± 1.45	50.93 ± 6.33	67.80 ± 2.59
HyperGCN	55.12 ± 1.96	42.45 ± 3.67	48.32 ± 2.93
HyperGNN	91.26 ± 1.15	48.59 ± 4.52	61.39 ± 2.96
AllDeepSets	91.80 ± 1.53	48.17 ± 5.67	67.82 ± 2.40
AllSetTransformer	92.16 ± 1.05	51.83 ± 5.22	69.33 ± 2.20
UniGCNII	94.81 ± 0.81	49.30 ± 4.25	67.25 ± 2.57
ED-HNN	95.00 ± 0.99	64.79 ± 5.14	72.45 ± 2.28
WHNN (ours)	$\mathbf{95.67 \pm 0.90}$	$\mathbf{67.04 \pm 4.80}$	$\mathbf{72.66 \pm 1.26}$

C Background

C.1 Hypergraph Representation Learning

A hypergraph is a tuple $\mathcal{H} = (V, E)$ where $V = \{v_1, v_2 \dots v_N\}$ is a set of nodes, and $E = \{e_1, e_2 \dots e_M\}$ is a set of hyperedges. Different from the graph structure, where each edge contains exactly two nodes, in a hypergraph, each hyperedge contains a set of nodes, which can vary in cardinality. Each node v_i is characterised by a feature vector $x_i \in \mathbb{R}^d$. We denote by *neighbourhood of hyperedge* e_i the set of nodes that are part of that hyperedge $\{v_j | v_j \in e_i\}$. Similarly, the *neighbourhood of a node* v_i is the set of all hyperedges containing that node $\mathcal{N}_{v_i} = \{e_j | v_i \in e_j\}$.

Several architectures were developed for hypergraph-structured input [15, 9, 24, 8]. However, the most general pipeline follows a two-stage framework, inspired by the bipartite representation of the hypergraphs. First, the information is sent from nodes to the hyperedges using a permutation-invariant operator $z_j = f_{V \to E}(\{x_i | v_i \in e_j\})$. Secondly, the messages are sent back from hyperedge to nodes $\tilde{x}_i = f_{E \to V}(\{z_j | v_i \in e_j\})$.

While aggregators like Deep Sets [12] were theoretically capable of approximating any permutation-invariant function on sets, they rely on the initial encoder (such as MLPs) to reshape the feature space in a way in which the sum pooling does not lose important information. In other words, it moves the complexity of the representation from the pooling to the initial encoding. This is in line with the empirical results shown in [25] where, in order to preserve good performance, mean pooling requires more complex encoders compared to more sophisticated pooling methods.

In this work, we are following the standard two-stage framework. Compared to existing methods, we take advantage of the success demonstrated by Sliced Wasserstein Pooling in capturing and retaining the geometric structure of sets and propose the first hypergraph model that uses optimal transport techniques to perform the node and hyperedge aggregation.

C.2 Sliced Wasserstein Pooling (SWP)

To ensure the method's readability, this section introduces all the key concepts underlying our Wasserstein Hypergraph Neural Network. First, we will define the 2-Wasserstein metric, approximate it using the tractable Sliced-Wasserstein distance and finally present the algorithm to compute the SWP used as an aggregator in our model.

Definition 1. The **2-Wasserstein distance** between two distributions p_i and p_j over $\mathbb{R}^d$ is defined as:

$$\mathcal{W}_2(p_i, p_j) = \Big(\inf_{\gamma \in \Gamma(p_i, p_j)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x - y||^2 d\gamma(x, y) \Big)^{\frac{1}{2}}, \tag{1}$$

where $\Gamma(p_i, p_j)$ represent the collection of all the transport plans with marginals p_i and p_j.

In simpler terms, the 2-Wasserstein distance quantifies the cost of transforming one distribution into another.

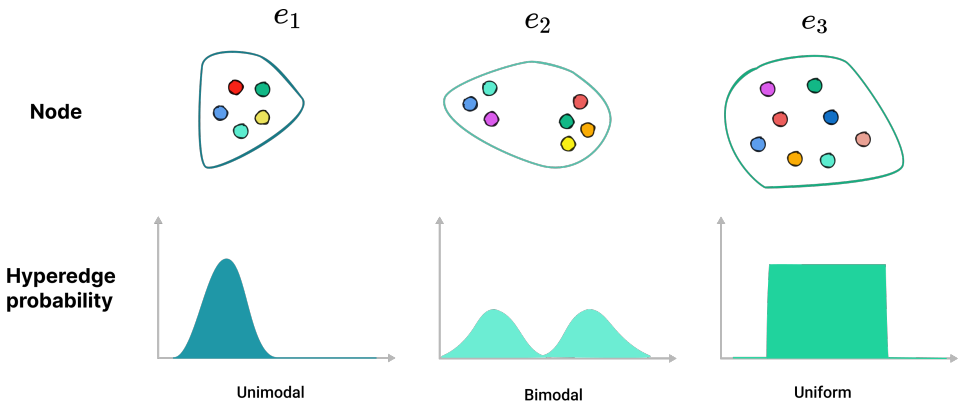

Figure 4: Diagram showing how different geometric arrangements of node embeddings correspond to distributions with varying shapes and spreads.

Unfortunately, computing the infimum over all possible transport maps is generally untractable. However, **in the one-dimensional case (when $d = 1$)), a closed-form solution exists** that avoids expensive optimisation. Specifically, when p_i and p_j are probability distributions over $\mathbb{R}$, the 2-Wasserstein distance is given by $\mathcal{W}_2(p_i, p_j) = \left(\int_0^1 |F_{p_i}^{-1}(t) - F_{p_j}^{-1}(t)|^2 dt \right)^{\frac{1}{2}}$, where $F_{p_i}^{-1}$ and $F_{p_j}^{-1}$ denote the inverse cumulative distribution functions of p_i and p_j. A key practical benefit of this formulation is that this inner integral can be empirically estimated using a discrete sum over sorted samples from the distribution.

Building on this observation, Sliced Wasserstein distance [26] was introduced to approximate the Wasserstein distance, by projecting the high-dimensional probabilities into 1D lines using all possible directions on the unit sphere.

Definition 2. The **Sliced Wasserstein distance** between two distributions p_i and p_j over $\mathbb{R}^d$ is defined as:

$$\mathcal{SW}_2(p_i, p_j) = \left(\int_{S^{d-1}} \mathcal{W}_2(P_\theta p_i, P_\theta p_j) d\theta \right)^{\frac{1}{2}} \approx \left(\frac{1}{L} \sum_{l=1}^{L} \underbrace{\mathcal{W}_2(P_{\theta_l} p_i, P_{\theta_l} p_j)}_{\text{1D Wasserstein distance}} \right)^{\frac{1}{2}}, \tag{2}$$

where S^{d-1} is the unit sphere in $\mathbb{R}^d$, $P_\theta p_i$ represent the projection (pushforward) of p_i onto the line direction θ and $\{\theta_l\}_{l=1}^{L}$ represents the set of L directions used to empirically approximate the expectation.

To avoid the computational cost of calculating distances between every pair of probability distributions, the **Sliced Wasserstein embedding** [14] was introduced. It maps a probability distributions p_i to a vector $\phi(p_i)$ in such a way that the Euclidean distance between the vectors (which is inexpensive to compute) approximates the Sliced Wasserstein distance between the original distributions $||\phi(p_i) - \phi(p_j)||_2 \approx \mathcal{SW}_2(p_i, p_j)$. In other words, it provides a vectorial representation that captures the geometric structure of distributions, preserving information about how costly it is to transform one distribution into another. This geometric encoding reflects characteristics such as shape, spread, and density. This proves useful in our context, as it allows us to quantify the cost of transforming one hyperedge into another, a measure we argue effectively captures the similarity between group interactions (hyperedges). Figure 4 illustrates how different node features exhibit distinct underlying distribution types.

Since our nodes and hyperedges are sets rather than distributions, we use a variant of this embedding called **Sliced Wasserstein Pooling** [14], which is designed not as an embedding of probability distributions themselves, but rather as an embedding of sets sampled from those distributions. In short, Sliced Wasserstein Pooling encodes a set of points by measuring, in an efficient way, how different they are positioned compared to a set of reference points. The complete algorithm as used in our model is described in the main paper.

D Related Work

Hypergraph representation learning. Hypergraphs represent a versatile structure for modelling group-wise interactions, which allows us to capture interactions between various numbers of elements. This flexibility, combined with the widespread presence of higher-order interactions in real-world scenarios, has led to a growing interest in developing machine learning architectures for modelling hypergraph data. Some methods [15, 27] reduce the hypergraph to a clique-expansion graph that can be further processed with standard graph neural networks. A more popular approach is based on a two-stage framework [8, 10], which sends the information from node to hyperedges and then from hyperedges back to nodes. Depending on how these stages are instantiated, several architectures emerged. HCHA and HERALD [16, 28] use an attention mechanism to combine the information. AllDeepSets [8] uses Deep Set model, while AllSetTransformer [8] uses a PMA-like [13] pooling.

In all of these methods, the information sent from the node is independent of the target hyperedge. Recently, models that create edge-dependent node representations have gained traction. ED-HNN [9] uses as messages a concatenation of node and hyperedge information, while MultiSetMixer [29] uses MLP-Mixer [30] to combine the information. Similar to our node encoder, CoNHD[31] incorporates pairwise propagation at the hyperedge-level using self-attention blocks (SAB [13]) to create edge-dependent representations. However, similar to [32], the model is only tested on hyperedge-dependent node classification tasks, where each node is assigned multiple labels corresponding to the number of hyperedges it participates in. A complementary line of work [33] represents uniform hypergraphs as high-dimensional tensors and applies tensorial operators to propagate the information.

In contrast, we are interpreting the hyperedges as samples from a set of probability distributions, and use Sliced Wasserstein Pooling to aggregate the information such that we preserve geometric information. In terms of node encoders, we are experimenting with both edge-dependent and edge-independent modules.

Set representation learning. The core operation in set representation learning is the permutation-invariant operator that aggregates the information without imposing an order among elements. Popular examples of such operators include summation, mean or maximum. More recently, a learnable version of permutation-invariant poolings was introduced. Among these, Deep Sets [12] uses element-wise encoding of the elements followed by summation and is proven to be a universal approximator for permutation-invariant functions. Janossy Pooling [34] extends this model by explicitly aggregating pairs of elements. On the other hand, Set Transformer [13] and [35] use an anchor set as a reference and compute the similarity against this set as a representation, while FSPool [36] sorts the elements feature-wise to create a canonical order. Recently, [25] shows empirically that combining an equivariant backbone with an invariant pooling layer creates powerful set representation learning. Inspired by optimal transport literature, Sliced Wasserstein Pooling was introduced in [14] as a geometrically-interpretable set representation technique.

Wasserstein embeddings. In recent years, Wasserstein distance has attracted significant attention in deep learning, demonstrating success in areas such as generative modeling [37, 38], natural language processing [39] and point cloud processing [40]. In graph representation learning, Wasserstein distance was used to define a similarity kernel between pairs of graphs [41]. While recognised as a powerful tool, computing this distance for each pair of compared graphs is extremely inefficient. More recent works [42, 43, 44] try to reduce this cost by introducing Wasserstein embeddings. The purpose of a Wasserstein embedding is to infer a vector representation such that the L_2 distance in the vector space approximates the Wasserstein distance in the input space. Particularly important for us is the work of [14] which produces set representations using efficient Wasserstein embeddings.

To more effectively capture the internal structure of node and hyperedge neighbourhoods, we employ Sliced Wasserstein Pooling as the aggregation operator in hypergraph message passing, demonstrating its advantages for hypergraph representation learning.

E Implementation details

The results reported in Table 2 of the main paper are obtained using random hyperparameter tuning. We report here the range of parameters that we searched for. Table 6 and Table 7 contain the best hyperparameter configuration for the WHNN_MLP model and WHNN_SAB. We depict in bold

the parameters specific to the Wasserstein aggregator, in italic the parameters specific to the SAB encoder, while the rest of them are the standard parameters used in the two-stage hypergraph models. In our experiment, we search for the following hyperparameters:

- num_ref: number of elements sampled from the reference distribution $\{5, 10, 25, 50\}$
- learnable_W: choose between learning or not the reference distribution $\{\text{True}, \text{False}\}$
- heads: number of heads used by the SAB block $\{1, 2, 4\}$
- MLP_layers: number of layers in all MLPs used $\{0, 1, 2\}$
- MLP_hid: number of hidden units in all MLPs used. This is also the number of slices used by Wasserstein aggregator. $\{128, 256, 512\}$
- MLP2_layers: using or not an additional linear projection after the residual connection of each stage $\{0, 1\}$
- Cls_layers: number of layers in the final classifier MLP $\{1, 2\}$
- Cls_hid: number of hidden units in the final classifier MLP $\{96, 128, 256\}$
- self_loops: using or not self loops $\{\text{True}, \text{False}\}$
- dropout: dropout used inside the model $\{0.5, 0.6, 0.7\}$
- in_dropout: dropout used in the begining of the model $\{0.2, 0.5, 0.6, 0.7\}$
- fixed hyperparameters: All models use 1 layer of WHNN, LayerNorm normalisation, the residual coefficient α fixed to 0.5, and they are trained for 500 epochs with a learning rate of 0.001.

Table 6: The best configuration of hyperparameters used by our model WHNN_MLP on all tested datasets. We mark with bold the parameters that are specific to the Wasserstein aggregator.

Parameter	Cora	Citeseer	Cora_CA	DBLP_CA	ModelNet40	NTU2012	20News
num_ref	25	10	25	5	50	25	25
learnable_W	True	False	True	True	False	False	False
MLP_layers	1	2	2	2	1	1	0
MLP2_layers	0	0	1	0	0	1	0
MLP_hid	128	256	256	512	256	512	512
Cls_layers	1	1	1	2	2	2	2
Cls_hid	256	128	96	96	96	96	96
self_loops	True	True	True	True	True	False	False
dropout	0.7	0.5	0.6	0.7	0.5	0.5	0.5
in_dropout	0.7	0.5	0.6	0.7	0.2	0.2	0.2

Table 7: The best configuration of hyperparameters used by our model WHNN_SAB on all tested datasets. We mark with bold the parameters that are specific to the Wasserstein aggregator and with italic the parameters that are specific to the SAB encoder.

Parameter	Cora	Citeseer	Cora_CA	DBLP_CA	ModelNet40	NTU2012	20News
num_ref	10	5	50	5	25	25	5
learnable_W	True	False	False	False	False	False	True
heads	2	4	1	4	1	2	2
MLP_layers	2	2	2	1	1	2	2
MLP2_layers	0	0	1	1	0	0	0
MLP_hid	128	256	128	256	256	512	512
Cls_layers	1	1	1	2	2	2	2
Cls_hid	128	256	128	96	96	96	96
self_loops	True	False	True	True	True	True	False
dropout	0.7	0.7	0.5	0.7	0.5	0.5	0.5
in_dropout	0.7	0.7	0.5	0.7	0.2	0.2	0.2

F Computational complexity

In this section, we first derive the theoretical complexity for both versions of our Wasserstein Hypergraph Neural Network: using the edge-independent encoder (WHNN_MPN) and using the edge-dependent encoder (WHNN_SAB).

For the theoretical analysis, we present the complexity for a hypergraph with N nodes, M hyperedges, the maximum cardinality of a hyperedge K_e, the maximum number of hyperedges a node is part of K_v and R number of reference points sampled from the reference distribution.

Regarding the encoders, the edge-independent one (MLP) has a complexity of $O(N)$ while the edge-dependent one (SAB) has complexity $O(M \times K^2)$ due to the pairwise exchange of messages (K^2) inside each hyperedge (M).

For the Wasserstein aggregator, the complexity for the nodes to hyperedges stage consists of the complexity of the linear interpolation applied for each hyperedge to obtain R points from the set of K_e points representing the hyperedge. After that, all we need to do is an elementwise difference between the interpolated points and the reference points. To sort each hyperedge, the complexity is $O(K_e log K_e)$, and the complexity for interpolating on each sorted hyperedge is $(R \times log K_e)$ for a total of $M \times (R \times log K_e + K_e log K_e)$. Similarly, for the hyperedge to node stage, the complexity is $N \times (R \times log K_v + K_v log K_v)$.

For comparison, the complexity of a Deep Set pooling is $O(M \times K_e + N \times K_v)$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: To our knowledge, this is the first paper that uses Sliced Wasserstein Pooling as an aggregator inside hypergraph neural networks, and our experiments demonstrate its effectiveness.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalise to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We included a section on Limitations and Future Work on the Supplementary Material.

   Guidelines:

   - The answer NA means that the paper has no limitation, while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or when images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognise that individual actions in favour of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed not to penalise honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA] .

Justification: The paper does not contain theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We did our best to include all the necessary information needed to reproduce the results, including hyperparameters used in the experiments. We also plan to publicly release the code upon acceptance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We plan to release the code upon acceptance. The workshop platform does not provide an option to include the code.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes] ,

   Justification: We included the implementation details in the main paper and hyperparameters used in the Supp material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Experiments were run with multiple runs and std is reported in the Table.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We reported in the paper the GPU used to run all the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work respects the NeurIPS Code of Ethics

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The paper is performing fundamental research and thus it has no direct societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No risk associated.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We will mention them in the ReadMe of the code repositories used to create the code.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.