MESSAGE PASSING NEURAL PROCESSES

Cătălina Cangea*, Ben Day*, Arian R. Jamasb, Pietro Liò
Department of Computer Science and Technology
University of Cambridge
{ccc53,bjd39,arj39,pl219}@cam.ac.uk

ABSTRACT

Neural Processes (NPs) are powerful and flexible models able to incorporate uncertainty when representing stochastic processes, while maintaining a linear time complexity. However, NPs produce a latent description by aggregating independent representations of context points and lack the ability to exploit relational information present in many datasets. This renders NPs ineffective in settings where the stochastic process is primarily governed by neighbourhood rules, such as cellular automata (CA), and limits performance for any task where relational information remains unused. We address this shortcoming by introducing Message Passing Neural Processes (MPNPs), the first class of NPs that explicitly makes use of relational structure within the model. Our evaluation shows that MPNPs thrive at lower sampling rates, on existing benchmarks and newly-proposed CA and Cora-Branched tasks. We further report strong generalisation over density-based CA rule-sets and significant gains in challenging arbitrary-labelling and few-shot learning setups.

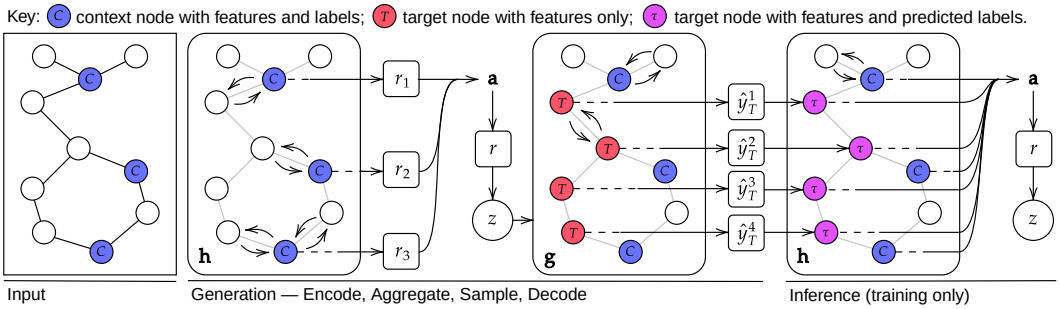

Figure 1: Computational graph of the Message Passing Neural Process. **Input:** the dataset consists of examples (nodes) and a relational structure (edges). Features, x, are observed for every node, but labels are only observed for the context set, the blue nodes labelled C. **Generation:** the encoder, $\mathbf{h}$, uses message-passing operations over the dataset to produce neighbourhood-aware representations of the context set, r_i. The aggregator, $\mathbf{a}$, combines these into a single representation, r, which parameterises the global latent variable, z. The decoder, $\mathbf{g}$, which also uses message-passing operations, is conditioned on a sample from the global latent variable and makes label predictions over the target set, $\hat{y}_T$. **Inference:** the predicted labels are added to the target examples, differentiated from the unlabelled targets by the label τ and purple nodes. The dataset is again passed through the encoder, $\mathbf{h}$, and aggregator, $\mathbf{a}$, to produce the global latent variable as conditioned on the joint target and context set, as required in the ELBO objective (Equation 5) for training.

1 MESSAGE PASSING NEURAL PROCESSES

We present Message Passing Neural Processes (MPNPs) as the synthesis of the MP and NP models. Figure 1 illustrates the operation of an MPNP and Appendix B gives more details including pseudocode for the entire computation and derivations.

Problem Statement. Given a partially-labelled set of *nodes* with features $\mathbf{X}$ *and neighbours given by* $\mathbf{A}$, sampled from $f : \mathbf{X}, \mathbf{A} \rightarrow \mathbf{Y}$, with $f \sim \mathcal{D}$, the goal is to predict labels for a subset of the unlabelled *nodes*.

Dataset Sampling. In the classification setting, the context set for a dataset (a *graph*) is defined as a set $\mathcal{C} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ of nodes and their one-hot labels. The information available to the encoder h is given by the set $\mathcal{C} \cup \{\mathbf{x}_j \mid j \in \bigcup_{i \in \text{context set}} \mathcal{N}(i)\}$, with $|\mathcal{C}| = m$, which contains the context set and the k-hop neighbourhoods of all context nodes. In this way, the MPNP uses the relational structure between the context set and other nodes to produce richer representations of the context nodes. In turn, the global latent variable $\mathbf{z}$ is able to encode relational structure present in the underlying stochastic process. The target set, $\mathcal{T} = \{\mathbf{x}_i \mid i \in \text{context set}\} \cup \{\mathbf{x}_i \mid i \notin \text{context set}\}$, with $|\mathcal{T}| = n$, is a superset of the context set (though not necessarily containing the entire graph), *without labels*. The decoder g also uses information from the k-hop neighbourhood when predicting target labels.

2 EXPERIMENTS

Recently introduced benchmark suites such as OGB (Hu et al., 2020) and those proposed by Dwivedi et al. (2020) aim to improve the quality of evaluation of graph-based models. However, we find they do not host tasks that match the problem statement in Section 1. To this end, in addition to standard variants of TUD (Kersting et al., 2020) and PPI (Zeng et al., 2019) tasks, we introduce new task formulations (Appendix C, D) of the existing ShapeNet (Chang et al., 2015; Yi et al., 2016) and full Cora (McCallum et al., 2000; Bojchevski & Günnemann, 2018) datasets and entirely novel synthetic tasks based on cellular automata (Von Neumann et al., 1951; Wolfram, 1984; Turing, 1990).

2.1 FIXED LABELLING TASKS

We first consider tasks where the same set of classes appear in every example in the same order (Tables 1,2, Figure 2). Inductive GNNs are designed for this setting and provide a useful baseline performance. Figures 3 and 6 (in the Appendix) show the superior uncertainty-modelling capabilities of the MPNP. We then introduce tasks based on modelling **Cellular Automata** to show how MPNPs extend available modelling capabilities, as baselines struggle in this setting (Figure 4). The model is provided with the states of some cells over a generation and tasked with evolving others.

Table 1: Node classification on TU datasets, reported at $\{5, 10, 30\}\%$ context points. **first / *second***.

	Enzymes			DHFR		
Model	5	10	30	5	10	30
NP	***79.23***	93.43	***95.75***	54.66	55.71	57.38
MPNP	79.09	***94.10***	**95.78**	***88.65***	***89.62***	***90.53***
GNN	**94.23**	**94.23**	94.23	**93.35**	**93.35**	**93.35**
LP	58.93	63.91	76.42	38.48	41.51	53.63

Table 2: Node classification on Protein-Protein Interaction Site Prediction. R-MPNP scores for $\{5, 30\}\%$ sampling rates. Results for ISIS, DeepPPISP and R-GCN are taken from Ofran & Rost (2007), Zeng et al. (2019), and Schlichtkrull et al. (2018), respectively.

Method	Accuracy %		F-measure		MCC	
ISIS	69.4		0.267		0.097	
DeepPPISP	65.5		**0.397**		0.206	
R-GCN	76.7		0.165		0.169	
	5	30	5	30	5	30
NP	77.5	79.3	0.212	0.180	0.145	0.150
R-MPNP	**79.1**	**80.7**	0.292	0.348	**0.236**	**0.284**

2.2 ARBITRARY LABELLING TASKS

As the total number of classes could be very large and test examples may include unseen classes, using fixed-classes is infeasible. Instead, arbitrary labellings $(1, \ldots, k)$ (Garnelo et al., 2018a) are assigned on a per-dataset basis, and models are required to adapt accordingly.

The Cora-ML task is a widely used community detection benchmark. Papers are represented by bag-of-words vectors with edges indicating that one of the papers cited the other. Our task, **Cora-Branched**, is derived from a more complete dataset, with 70 classes over 11 computer science

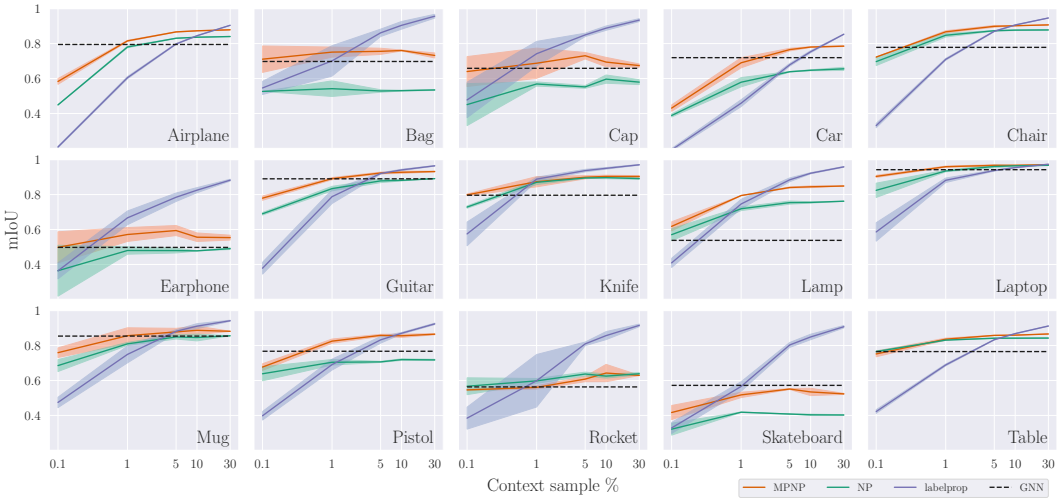

Figure 2: Linear-log plots of mIoU over context sample rates with 95% confidence interval shading for the fixed-class ShapeNet task, by category. The GNN is inductive and does not depend on context sampling. The Appendix gives numerical results (Tables 9, 10 and 11).

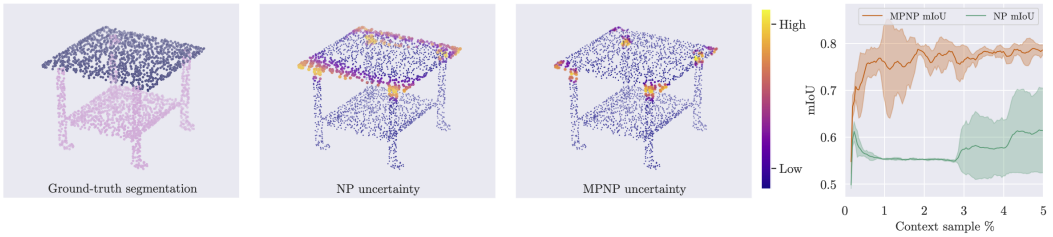

Figure 3: Segmentation uncertainty over an example from the ShapeNet fixed-class *table* category test set and active sampling. *(Left:)* Ground truth labels are shown for the table-top (purple) and table-leg (pink) parts. *(Centre:)* Uncertainty is depicted by the size and colour of the points: higher at larger, yellower points and lower at smaller, bluer points. *(Right:)* Active sampling.

disciplines (McCallum et al., 2000; Bojchevski & Günnemann, 2018). There are 10 times as many classes and the bag-of-words feature vectors are tripled in length. Given a partially labelled subgraph of the network, the task is to label the rest (Tables 3 and 4). In the **ShapeNet mixed-category** setup (Table 5), we model the process that produces n-part objects (say, $n = 4$ for *chairs* with *arms, legs, seats* and *backs*, as well as *airplanes* with *engines, bodies, tails* and *wings*.) We thus provide an arbitrary permutation of class labels for each example.

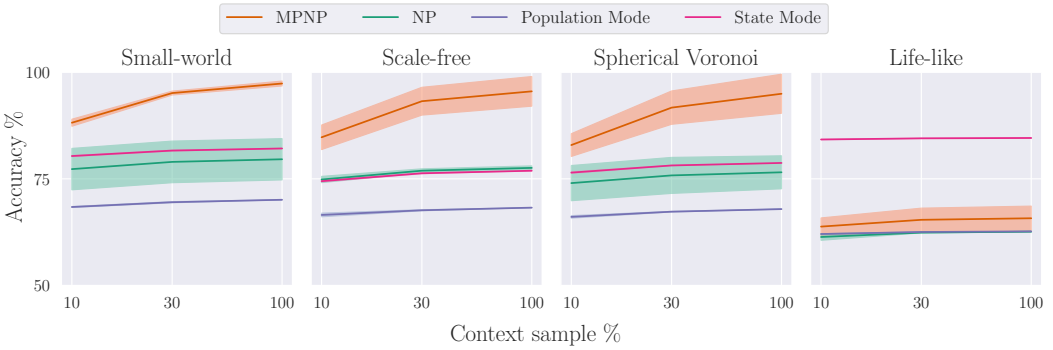

Figure 4: State evolution accuracy $\pm\sigma$ for density- and count-based cellular automata. Models are trained at 30-50% context sampling. Testing at 100% effectively judges the quality of the rule embedding under perfect information.

The results presented show that the richer context representations and structural bias of the MPNP are generally beneficial, outperforming the NP on Cora-Branched, PPISP, one TUD task and ShapeNet mixed (excluding 3-class @ 0.1%), while producing semantically-realistic uncertainties, as shown in Figure 3 (and Figure 6 in the Appendix.) Label propagation is more successful when

Table 3: Results on the Cora-Branched transductive learning tasks for 3, 7 and 11 classes (#). Mean accuracy and standard deviations are reported at $\{1, 5, 10, 30\}\%$ context points.

#	Model	1%	5%	10%	30%
3	NP-c	67.00 ± 1.83	76.99 ± 1.50	78.56 ± 1.19	79.61 ± 1.20
	MPNP-c	**79.71 ± 1.04**	**88.28 ± 0.59**	**89.41 ± 0.58**	**90.02 ± 0.60**
	LP	65.31 ± 0.73	75.57 ± 0.31	77.90 ± 0.16	82.04 ± 0.18
	Mode	54.35 ± 0.10	54.28 ± 0.07	54.40 ± 0.27	54.41 ± 0.18
7	NP-c	52.83 ± 0.49	63.02 ± 0.50	64.29 ± 0.43	65.23 ± 0.51
	MPNP-c	**58.40 ± 0.77**	**68.96 ± 1.08**	**70.53 ± 0.88**	71.54 ± 0.91
	LP	52.62 ± 0.31	64.85 ± 0.22	68.55 ± 0.14	**74.96 ± 0.20**
	Mode	30.48 ± 0.16	30.57 ± 0.07	30.50 ± 0.10	30.50 ± 0.10
11	NP-c	34.57 ± 2.18	37.94 ± 0.84	38.88 ± 0.80	39.42 ± 0.78
	MPNP-c	43.62 ± 1.01	50.64 ± 1.14	51.87 ± 1.23	52.67 ± 1.24
	LP	**46.84 ± 0.55**	**60.11 ± 0.12**	**64.22 ± 0.08**	**71.73 ± 0.05**
	Mode	21.60 ± 0.08	21.60 ± 0.11	21.63 ± 0.09	21.66 ± 0.10

Table 4: Performance on the Cora-Branched few-shot learning generalisation tasks for 2, 3, 5 and 11 class (#) tasks. Accuracy at $\{1, 5, 10\}\%$ context points.

#	Model	1%	5%	10%
2	NP-c	59.25	63.53	64.29
	MPNP-c	**62.91**	**67.53**	**68.57**
3	NP-c	49.82	56.93	59.03
	MPNP-c	**53.83**	**63.75**	**64.52**
5	NP-c	36.84	42.68	44.10
	MPNP-c	**41.67**	**49.99**	**51.15**
11	NP-c	19.71	21.13	21.82
	MPNP-c	**23.56**	**26.00**	**27.44**

more labels are available, but MPNP greatly improves on it at low sampling rates, showing powerful capabilities in scarce data settings. GNNs learn better when the generative process has little functional variation, but perform poorly in the opposite case (mixed-class and few-shot), being entirely unsuitable in the arbitrary labelling setting. The TUD biochemical datasets are the only fixed-class setting where GNNs do consistently better than MPNPs, though we can attribute this to the lack of functional variation of the generative process in these narrow tasks. On ShapeNet and PPISP fixed-class tasks, MPNP surpasses the GNN in most cases.

Table 5: ShapeNet mixed-category, arbitrary-labelling results for 2, 3, and 4-part shapes (#). We report the mIoU for $\{0.1, 1, 5, 10\}\%$ context points.

#	Model	0.1%	1%	5%	10%
2	NP-c	48.06	83.60	88.62	89.17
	MPNP-c	**57.18**	**86.08**	90.81	91.37
	LP	55.55	84.37	**91.90**	**93.93**
	GNN	36.14	36.14	36.14	36.14
3	NP-c	**46.87**	76.66	81.12	81.47
	MPNP-c	45.52	**78.95**	83.80	84.31
	LP	41.12	69.84	**84.40**	**87.76**
	GNN	21.68	21.68	21.68	21.68
4	NP-c	28.48	67.19	72.30	72.88
	MPNP-c	**31.52**	**74.30**	81.38	82.20
	LP	30.29	66.61	**83.61**	**87.91**
	GNN	15.82	15.82	15.82	15.82

REFERENCES

Martin Bock, Amit Kumar Tyagi, Jan-Ulrich Kreft, and Wolfgang Alt. Generalized Voronoi Tessellation as a Model of Two-dimensional Cell Tissue Dynamics. *arXiv e-prints*, art. arXiv:0901.4469, January 2009.

Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations*, pp. 1–13, 2018.

Andrew Carr and David Wingate. Graph Neural Processes: Towards Bayesian Graph Neural Networks. *arXiv e-prints*, art. arXiv:1902.10042, February 2019.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv e-prints*, art. arXiv:1512.03012, December 2015.

Wei Chu, Vikas Sindhwani, Zoubin Ghahramani, and S Sathiya Keerthi. Relational Learning with Gaussian Processes. In *Advances in Neural Information Processing Systems*, pp. 289–296, 2007.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *arXiv preprint arXiv:2003.00982*, 2020.

Mathematical Games. The Fantastic Combinations of John Conway's New Solitaire Game 'Life' by Martin Gardner. *Scientific American*, 223:120–123, 1970.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional Neural Processes. In *International Conference on Machine Learning*, pp. 1704–1713, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural Processes. *arXiv preprint arXiv:1807.01622*, 2018b.

William Gilpin. Cellular Automata as Convolutional Neural Networks. *Physical Review E*, 100(3), 9 2018. doi: 10.1103/PhysRevE.100.032402.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning-Volume 28*, pp. III–1319, 2013.

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional Conditional Neural Processes. *arXiv e-prints*, art. arXiv:1910.13556, October 2019.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=8E1-f3VhX1o.

Robert Hunt, Engin Mendi, and Coskun Bayrak. Using Cellular Automata to Model Social Networking Behavior. In *12th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2011 - Proceedings*, pp. 287–290, 2011. ISBN 9781457700453. doi: 10.1109/CINTI.2011.6108515.

Arian R. Jamasb, Pietro Lió, and Tom L. Blundell. Graphein - a python library for geometric deep learning and network analysis on protein structures. *bioRxiv*, 2020. doi: 10.1101/2020.07.15.204701. URL https://www.biorxiv.org/content/early/2020/07/15/2020.07.15.204701.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark Data Sets for Graph Kernels, 2020. URL `http://www.graphlearning.io/`.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive Neural Processes. *arXiv e-prints*, art. arXiv:1901.05761, January 2019.

Clélia Lopez, Chuan-Lin Zhao, Stéphane Magniol, Nicolas Chiabaut, and Ludovic Leclercq. Microscopic Simulation of Cruising for Parking of Trucks as a Measure to Manage Freight Loading Zone. *Sustainability*, 11(5):1276, 2 2019. ISSN 2071-1050. doi: 10.3390/su11051276.

Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The Functional Neural Process. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8746–8757. Curran Associates, Inc., 2019.

Krzysztof Malecki. Graph Cellular Automata with Relation-Based Neighbourhoods of Cells for Complex Systems Modelling: A Case of Traffic Simulation. *Symmetry*, 9(12), 12 2017. ISSN 20738994. doi: 10.3390/sym9120322.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163, 2000. ISSN 13864564. doi: 10.1023/A:1009953814988.

Jens Meiler, Anita Zeidler, Felix Schmüschke, and Michael Müller. Generation and Evaluation of Dimension-reduced Amino Acid Parameter Representations by Artificial Neural Networks. *Journal of Molecular Modeling*, 7(9):360–369, September 2001. doi: 10.1007/s008940100038.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing Neural Cellular Automata. *Distill*, 2020. doi: 10.23915/distill.00023. https://distill.pub/2020/growing-ca.

Yin Cheng Ng, Nicolò Colombo, and Ricardo Silva. Bayesian Semi-Supervised Learning with Graph Gaussian Processes. In *Advances in Neural Information Processing Systems*, pp. 1683–1694, 2018.

Y. Ofran and B. Rost. ISIS: Interaction Sites Identified from Sequence. *Bioinformatics*, 23(2): e13–e16, January 2007. doi: 10.1093/bioinformatics/btl303.

Felix L. Opolka and Pietro Liò. Graph Convolutional Gaussian Processes For Link Prediction. *arXiv e-prints*, art. arXiv:2002.04337, February 2020.

James Requeima, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner. Fast and Flexible Multi-Task Classification using Conditional Neural Adaptive Processes. In *Advances in Neural Information Processing Systems*, pp. 7957–7968, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Gautam Singh, Jaesik Yoon, Youngsung Son, and Sungjin Ahn. Sequential Neural Processes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 10254–10264. Curran Associates, Inc., 2019.

Alan Mathison Turing. The Chemical Basis of Morphogenesis. *Bulletin of Mathematical Biology*, 52(1-2):153–197, 1990.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *arXiv e-prints*, art. arXiv:1710.10903, October 2017.

John Von Neumann et al. The General and Logical Theory of Automata. *1951*, pp. 1–41, 1951.

Ian Walker and Ben Glocker. Graph Convolutional Gaussian Processes. In *International Conference on Machine Learning*, pp. 6495–6504, 2019.

Roger White. Cities and Cellular Automata. *Discrete Dynamics in Nature and Society*, 2(2):111–125, 1998. ISSN 1026-0226. doi: 10.1155/s1026022698000090.

Stephen Wolfram. Universality and Complexity in Cellular Automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35, 1984.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. *33rd International Conference on Machine Learning, ICML 2016*, 1: 86–94, 3 2016.

Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A Scalable Active Framework for Region Annotation in 3D Shape Collections. *ACM Transactions on Graphics (TOG)*, 35(6):1–12, 2016.

Min Zeng, Fuhao Zhang, Fang-Xiang Wu, Yaohang Li, Jianxin Wang, and Min Li. Protein–Protein Interaction Site Prediction through Combining Local and Global Features with Deep Neural Networks. *Bioinformatics*, September 2019. doi: 10.1093/bioinformatics/btz699.

Xiaojin Zhu and Zoubin Ghahramani. Learning from Labeled and Unlabeled Data with Label Propagation. 2002.

A Background and Related Work

We begin by reviewing the theoretical foundations of our building blocks (Neural Processes, Message Passing architectures) and related works. The next section presents MPNPs as a combination of these ideas that operates on datasets with relational structure generated by stochastic processes.

A.1 Neural Processes

Problem Statement. Given a set of points with features X, partially labelled by a function $f : X \to Y$ sampled from a distribution over functions, $\mathcal{D}$, the goal is to predict labels for a subset of the unlabelled points.

A Neural Process (NP) (Garnelo et al., 2018b) learns to represent a stochastic process with an underlying distribution $\mathcal{D}$. To achieve this, the NP is trained on a set of functions $f : X \to Y$ sampled from $\mathcal{D}$ and tested on a disjoint set. For each function, f_i, a dataset contains tuples (x_j, y_j), where $y_j = f_i(x_j)$. Their joint probability distribution can be written as $p(y_{1:n}|x_{1:n}) = \int p(f_i)p(y_{1:n}|f_i, x_{1:n})df_i$. Assuming observation noise $Y_j \sim \mathcal{N}(f_i(x_j), \sigma^2)$ and a neural network γ modelling the stochastic process instance f_i (that is, $\gamma(x, z) = f_i(x)$, where z is a random vector that mimics the randomness of f_i), we obtain the generative model:

$$p(z, y_{1:n}|x_{1:n}) = p(z) \prod_{j=1}^{n} \mathcal{N}(y_j|\gamma(x_j, z), \sigma^2), \tag{1}$$

where $p(z)$ is a multivariate normal distribution. Learning the non-linear function γ requires amortised variational inference on the evidence lower bound (ELBO), using a neural-network-parameterised posterior $q(z|x_{1:n}, y_{1:n})$. Model generation starts with the NP receiving a set of m context points $\mathcal{C}_i = \{(x_j, y_j)\}_{j=1}^{m}$ sampled from f_i. The model then predicts the values $y_j = f_i(x_j)$ for n target points $\mathcal{T}_i = \{x_j\}_{j=1}^{n}$; namely, the m original context points and $(m - n)$ previously unseen target points. To match this setup, we isolate the context set $x_{1:m}, y_{1:m}$ from the target set $x_{m+1:n}, y_{m+1:n}$ in equation 1. The final ELBO is:

$$\log p(y_{m+1:n}|x_{1:n}, y_{1:m}) \geq \mathbb{E}_{q(z|x_{1:n}, y_{1:n})}\Big[\sum_{j=m+1}^{n} \log p(y_j|z, x_j) + \log \frac{q(z|x_{1:m}, y_{1:m})}{q(z|x_{1:n}, y_{1:n})} \Big]. \tag{2}$$

Framework Details. We note the differences in NP processing with respect to typical machine learning setups. An NP is trained over multiple datasets, or *sets of samples* $\mathcal{S}_i$ from functions

$f_i \sim \mathcal{D}$, with a given training episode consisting of samples from a single such function. Sampling over the distribution of functions provides information about the variability of the stochastic process being modelled to the NP. The context set $\mathcal{C}_i$ described above is a (labelled) subset of $\mathcal{S}_i$, while the target set $\mathcal{T}_i$ is an (unlabelled) superset of $\mathcal{C}_i$, with $\mathcal{C}_i \subseteq \mathcal{T}_i \subseteq \mathcal{S}_i$. Section 1 explains how our model uses the context set to compute target set predictions, while expanding on each of the main components of the framework—encoder, aggregator, decoder—that are common across NP models.

A.2 Message Passing and Graph Neural Networks

Neural networks that operate on graph-structured data process node features $\mathbf{X} \in \mathbb{R}^{n \times d}$ with relational information in the form of an adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$. The aim is to produce embeddings that are useful for downstream tasks such as node or graph classification. Graph neural networks typically use generalised convolutional layers to learn these embeddings. We describe their operation via the universal Message Passing (MP) paradigm; the next section presents the specific MP instance that our models use.

Let $\mathbf{h}_i^t \in \mathbb{R}^{d'}$ be the features of the i-th node after t message passing steps, where d' is the embedding dimensionality; optionally, we may have edge features $\mathbf{e}_{ij} \in \mathbb{R}^k$ where $A_{ij} = 1$. A message passing layer corresponds to a single message passing step, updating the node features as follows, where F and G are learnable functions, $\mathcal{N}(i) = \{j \mid A_{ij} = 1\}$ and $\square$ is a permutation-invariant aggregation function:

$$\mathbf{h}_i^{t+1} = \mathit{MP}(\mathbf{h}^t) \triangleq F(\mathbf{h}_i^t, \square_{j \in \mathcal{N}(i)}, G(\mathbf{h}_i^t, \mathbf{h}_j^t, \mathbf{e}_{ij})). \tag{3}$$

A.3 Neural Process Models

Garnelo et al. (2018b) formulated the *Neural Process* as a favourable combination of neural networks and Gaussian Processes. *Conditional Neural Processes* (CNPs) (Garnelo et al., 2018a) are NP instances without a global latent variable, which implies a deterministic dependence on the context set. *Attentive NPs* (Kim et al., 2019), *CNAPs* (Requeima et al., 2019), *Convolutional CNPs* (Gordon et al., 2019) and *Sequential NPs* (Singh et al., 2019) make modifications to reduce underfitting, better adapt in the multi-task setting, and apply inductive biases for translation and temporal sequences, respectively. Louizos et al. (2019) propose the *Functional NP* that learns a graph of dependencies between latent representations of the points, without placing a prior over the latent global variable, though their tasks do not contain explicit relational information. The *Graph NP* (Carr & Wingate, 2019) is most closely related to the MPNP, performing edge imputation using a CNP-based model and Laplacian-derived features for the context points. However, despite the naming similarity, Graph NPs and MPNPs address different tasks—the former was evaluated on link prediction, which is not in the scope of our work. Moreover, our NP-based model is more flexible, handles uncertainty and learns from neighbourhoods, rather than whole-graph features, for classifying individual dataset samples (nodes), while leveraging the structure between them (edges).

A.4 Graph Learning under Uncertainty

Graph Gaussian Processes (Ng et al., 2018) were designed as an extension to GPs, where the covariance function and prior exploit the existence of features in node neighbourhoods. Graph GPs are the only Gaussian method for node classification, but perform slightly worse than GCNs—a type of GNNs that we use as a baseline. Moreover, the complexity is somewhat higher: $\mathcal{O}(\texttt{max\_node\_degree}^2 * N)$ vs. $\mathcal{O}(N)$ for (MP)NP, where $N =$ set of observations/context nodes. The *Relational GP* (Chu et al., 2007) models pairwise undirected links between data points, thus addressing a different task. The *Graph Convolutional GP* (Walker & Glocker, 2019) is a translation-invariant model that operates similarly to convolutional layers, while generalising to non-Euclidean domains. More recently, Opolka & Liò (2020) have also proposed a *Graph Convolutional GP* model for link prediction, which uses a GP for node-level predictions, another GP that builds on the first one for edge-level predictions, and a deep GP incorporating these building blocks to produce more expressive representations.

B MPNP DETAILS

Generation and Inference. Starting from Equation 1, with the function γ corresponding to the neural network $\mathbf{g}$ in Figure 1, and letting $\mathbf{x}_{\mathcal{N}(i)}$ denote features corresponding to an entire neighbourhood, we state the generative model for the MPNP (Appendix B.1 contains a complete derivation):

$$p(\mathbf{z}, \mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}) = p(\mathbf{z}) \prod_{i=1}^{n} \mathcal{N}\big(\mathbf{y}_i \mid F(\mathbf{x}_i \| \mathbf{z}, \square_{j \in \mathcal{N}(i)}, G(\mathbf{x}_i \| \mathbf{z}, \mathbf{x}_j \| \mathbf{z})), \sigma^2\big). \tag{4}$$

The decoder function g is a composition of learnable functions (linear projections, MP steps) and non-linearities, so it is trainable with amortised variational inference. The variational posterior $q(\mathbf{z}|\mathbf{x}_{1:n}, \mathbf{y}_{1:n})$ is also parameterised by a neural network ($\mathbf{h}$ in Figure 1) that is permutation-invariant, as each of the functions in $\mathbf{h}$ satisfies this property (full proof in Appendix B.3). Optimisation can be achieved using standard methods with the following variational approximation of the ELBO objective (fully derived in Appendix B.4), where $D = \mathbf{x}_{1:n} \cup \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)} \cup \mathbf{y}_{1:n}$:

$$\begin{aligned} \log p\big(\mathbf{y}_{m+1:n} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:m}\big) \geq \sum_{i=m+1}^{n} \mathbb{E}_{q(\mathbf{z}|D)}\big[\log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z})\big] \\ - \mathbb{KL}\Big(q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n}) \\ \Big\| \, q(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})\Big). \end{aligned} \tag{5}$$

Aggregation in Challenging Settings. The manner in which information is stored in the global latent variable $\mathbf{z}$ is crucial—at test time, the (context-conditioned) sample is processed together with the new target points, so it must reflect the behaviour of the new stochastic process in a way that is relevant to the task. Despite a simple mean over r_i being sufficient for many tasks, it is often necessary to produce a class-aware representation. Therefore, we adopt the alternative aggregation function used by Garnelo et al. (2018a) for few-shot learning tasks,

$$a'(\{\mathbf{r}_i\}) \triangleq \Big\|_{\mathbf{c} \in C} a(\mathbb{I}_{\{\mathbf{c}\}}(\mathbf{y}_i) * \mathbf{r}_i), \tag{6}$$

where C is the set of classes *in the current context*, with $|C|$ fixed, as required. This performs concatenation ($\|$) of per-class summaries aggregated with a. Intuitively, different classes in the context set are clearly delimited in this scheme, which is especially helpful in few-shot learning settings, where novel classes are seen during testing. Models using this scheme have the '-c' suffix.

B.1 GENERATIVE MODEL

Equation 1 lets us derive the MPNP generative model, where the function γ corresponds to the neural network g in Figure 1 and $\mathbf{x}_{\mathcal{N}(i)}$ denotes the features corresponding to the neighbourhood of node i:

$$\begin{aligned} p(\mathbf{z}, \mathbf{y}_{1:n} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}) &= p(\mathbf{z}) \prod_{i=1}^{n} p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}) \\ &= p(\mathbf{z}) \prod_{i=1}^{n} \mathcal{N}\big(\mathbf{y}_i \mid \gamma(\mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}), \sigma^2\big) \\ &= p(\mathbf{z}) \prod_{i=1}^{n} \mathcal{N}\big(\mathbf{y}_i \mid F(\mathbf{x}_i \| \mathbf{z}, \bigodot_{j \in \mathcal{N}(i)}, G(\mathbf{x}_i \| \mathbf{z}, \mathbf{x}_j \| \mathbf{z})), \sigma^2\big). \end{aligned} \tag{7}$$

In this derivation, line 2 assumes that $p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z})$ takes the form of a normal distribution, with mean and variance being functions of $\mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}$. Line 3 uses the fact that, in our model, $\gamma = \text{ReLU} \circ L_2 \circ MP^T \circ \text{ReLU} \circ L_1$. Let us first consider the case for $T = 1$. The function G corresponds to a linear transformation $L_1 = \mathbf{W}_{\text{MP}}$ applied to each of the (target node) neighbours' feature

vectors (here, we refer to the concatenated representations $\mathbf{x}_j \| \mathbf{z}$). This is followed by leveraging the aggregation operator $\bigodot_{j \in \mathcal{N}(i)}$ within the neighbourhood of each target node. Finally, F consists of applying the skip-connection (linear transformation) $\mathbf{W}_{\text{skip}}$ to each of the target node feature vectors, followed by the ReLU activation of the MP step and ReLU $\circ\, L_2$. The only difference for $T = 2$ lies in the aggregator and linear transformations within the MP step being performed twice. It is important to note that the variance σ^2 is output by the same network γ, as each prediction has its own associated uncertainty.

B.2 Model Pseudocode

Algorithm 1 summarises the MPNP label generation process described in the **Message Passing Neural Processes** section.

B.3 Encoder Permutation Invariance

We show that, for initial node representations $\mathbf{h}_i$, the transformation $\mathbf{r}_i = (L_2 \circ \mathit{MP}^T \circ \text{ReLU} \circ L_1)(\mathbf{h}_i)$ produced by the encoder is permutation-invariant:

$$\begin{aligned} \forall \text{ permutation } \mathbf{\Pi}. (L_2 \circ \mathit{MP}^T \circ \text{ReLU} \circ L_1)(\mathbf{X\Pi}, \mathbf{\Pi}^\top \mathbf{A\Pi}) = \\ \big((L_2 \circ \mathit{MP} \circ \text{ReLU} \circ L_1)(\mathbf{X}, \mathbf{A})\big)\mathbf{\Pi}. \end{aligned} \tag{8}$$

Proof: Assume an arbitrary set of features $\mathbf{X} \in \mathbb{R}^{n \times d}$ and an adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$, where n is the number of nodes in the context set and d is the feature dimensionality. We first show that each of the operations within the encoder is permutation-invariant:

1. The linear projections L_1, L_2 are applied to each of the node vectors $\mathbf{X}_i$ separately, so changing the order of input nodes will result in the same order in the output:

$$\begin{aligned} L_i(\mathbf{X\Pi}, \mathbf{\Pi}^\top \mathbf{A\Pi}) &= L_i(\mathbf{X\Pi}), \forall i \in \{1, 2\} \\ &= (L_i(\mathbf{X}_{\Pi_1})\ L_i(\mathbf{X}_{\Pi_2})\ \ldots\ L_i(\mathbf{X}_{\Pi_n}))^\top \\ &= (L_i(\mathbf{X}_1)\ L_i(\mathbf{X}_2)\ \ldots\ L_i(\mathbf{X}_n))^\top \mathbf{\Pi} \\ &= L_i(\mathbf{X})\mathbf{\Pi} \\ &= L_i(\mathbf{X}, \mathbf{A})\mathbf{\Pi}. \quad \square \end{aligned} \tag{9}$$

2. The same holds for the activation functions, which are applied element-wise:

$$\begin{aligned} \text{ReLU}(\mathbf{X\Pi}, \mathbf{\Pi}^\top \mathbf{A\Pi}) &= \text{ReLU}(\mathbf{X\Pi}) \\ &= \text{ReLU}(X_{\Pi_i j}), \forall i, j \\ &= \text{ReLU}(X_{ij})\mathbf{\Pi} \\ &= \text{ReLU}(\mathbf{X})\mathbf{\Pi} \\ &= \text{ReLU}(\mathbf{X}, \mathbf{A})\mathbf{\Pi}. \quad \square \end{aligned} \tag{10}$$

3. The message passing operation is also permutation-invariant, since the transformation $\mathbf{A} \to \mathbf{P^T A P}$ preserves the structure of the graph, with node neighbourhoods undergoing the transformation $\mathcal{N}(i) \triangleq \{j \mid A_{ij} = 1\} \to \mathcal{N}(i)\mathbf{\Pi} \triangleq \{\Pi_j \mid A_{\Pi_i \Pi_j} = 1\}$:

$$\begin{aligned} \mathit{MP}(\mathbf{X\Pi}, \mathbf{\Pi} \top \mathbf{A\Pi}) &= \text{ReLU}\big(\mathbf{W}_{\text{skip}}(\mathbf{X\Pi})_i + \sum_{j' \in \mathcal{N}(i)\mathbf{\Pi}} \mathbf{W}_{\text{MP}}(\mathbf{X\Pi})_j\big), \text{where } j' = \Pi_j, \\ &= \text{ReLU}\big(\mathbf{W}_{\text{skip}}(\mathbf{X}_{\Pi_i}) + \sum_{\Pi_j \in \mathcal{N}(\Pi_i)} \mathbf{W}_{\text{MP}}(\mathbf{X}_{\Pi_j})\big) \\ &= \text{ReLU}\big((\mathbf{W}_{\text{skip}}\mathbf{X}_i)\mathbf{\Pi} + \sum_{j \in \mathcal{N}(i)} (\mathbf{W}_{\text{MP}}\mathbf{X}_j)\mathbf{\Pi}\big) \\ &= \text{ReLU}\big(\mathbf{W}_{\text{skip}}\mathbf{X}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{W}_{\text{MP}}\mathbf{X}_j\big)\mathbf{\Pi} \\ &= \mathit{MP}(\mathbf{X}, \mathbf{A})\mathbf{\Pi}. \quad \square \end{aligned} \tag{11}$$

Algorithm 1: MPNP computation.

Input : Context set $\mathcal{C} = \{\mathbf{x}_i, \mathbf{y}_i\}$, with $|\mathcal{C}| = m$, features of context set node neighbours $\{\mathbf{x}_i \parallel j \in \bigcup_{i \in \text{context set}} \mathcal{N}(i)\}$, target set $\mathcal{T} = \{\mathbf{x}_i \mid i \in \text{context set}\} \cup \{\mathbf{x}_i \mid i \notin \text{context set}\}$, with $|\mathcal{T}| = n > m$, features of target set node neighbours $\{\mathbf{x}_i \parallel j \in \bigcup_{i \in \text{target set}} \mathcal{N}(i)\}$.

Output: Target label predictions $\{\hat{\mathbf{y}}_i \parallel i \in \text{target set}\}$.

// Initialise node features

foreach $i \in$ *context set* **do**
  $\mathbf{h}_i^0 \leftarrow \mathbf{x}_i \parallel \mathbf{y}_i$
  foreach $j \in \bigcup_{i \in \textit{context set}} \mathcal{N}(i)$ **do**
    $\mathbf{h}_j^0 \leftarrow \mathbf{x}_j \parallel \mathbf{0}$

// Encoding

foreach $i \in$ *context set* **do**
  $\mathbf{h}_i^0 \leftarrow \text{ReLU}(L_1(\mathbf{h}_i^0))$
  foreach $j \in \bigcup_{i \in \textit{context set}} \mathcal{N}(i)$ **do**
    $\mathbf{h}_j^0 \leftarrow \text{ReLU}(L_1(\mathbf{h}_j^0))$

foreach $t \in 1, ..., T$ **do**
  foreach $i \in$ *context set* **do**
    $\mathbf{h}_i^t \leftarrow \textit{MP}(\mathbf{h}^{t-1})$

foreach $i \in$ *context set* **do**
  $\mathbf{r}_i \leftarrow L_2(\mathbf{h}_i^T)$

// Aggregation

$\mathbf{r} \leftarrow a(\{\mathbf{r}_i \parallel i \in \text{context set}\})$

// Decoding

Sample $\mathbf{z}' \sim \mathcal{N}(\mu(\mathbf{r}), \text{diag}[\sigma(\mathbf{r})])$

foreach $i \in$ *target set* **do**
  $\mathbf{h}_i'^0 = \mathbf{x}_i \parallel \mathbf{z}'$
  foreach $j \in \bigcup_{i \in \textit{target set}} \mathcal{N}(i)$ **do**
    $\mathbf{h}_j'^0 \leftarrow \mathbf{x}_j \parallel \mathbf{z}'$

foreach $i \in$ *target set* **do**
  $\mathbf{h}_i'^0 \leftarrow \text{ReLU}(L_1(\mathbf{h}_i'^0))$
  foreach $j \in \bigcup_{i \in \textit{target set}} \mathcal{N}(i)$ **do**
    $\mathbf{h}_j'^0 \leftarrow \text{ReLU}(L_1(\mathbf{h}_j'^0))$

foreach $t \in 1, ..., T$ **do**
  foreach $i \in$ *target set* **do**
    $\mathbf{h}_i'^t \leftarrow \textit{MP}(\mathbf{h}'^{t-1})$

foreach $i \in$ *target set* **do**
  $\mathbf{r}_i' \leftarrow \text{ReLU}(L_2(\mathbf{h}_i'^T))$
  $\hat{\mathbf{y}}_i' \sim \mathcal{N}\Big(\text{softmax}(\mu(\mathbf{r}_i')),$
    $\text{diag}[\big(0.1 + 0.9 \times \text{softplus}(\sigma(\mathbf{r}_i'))\big)]\Big)$

Each type of operation performed within the encoder is thus permutation-invariant. Composing permutation-invariant functions yields a function which has this property itself, so it follows that the overall transformation is permutation-invariant. □

B.4 ELBO

We derive the variational approximation to the ELBO objective stated under **Generation and Inference**. In the derivation, we assume m context nodes and n target nodes (that is, $n - m$ additional targets). The aim is to maximise the log-likelihood of target labels $\mathbf{y}_{m+1:n}$, given the target node features $\mathbf{x}_{1:n}$, context node features $\mathbf{x}_{1:m}$, context labels $\mathbf{y}_{1:m}$ and neighbourhoods of context nodes. We denote by $\mathbf{x}_{\mathcal{N}(i)}$ the features corresponding to an entire neighbourhood and let $D = \mathbf{x}_{1:n} \cup \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)} \cup \mathbf{y}_{1:n}$.

$$
\begin{aligned}
&\log p\big(\mathbf{y}_{m+1:n} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:m}\big) = \\
&\log p\big(\mathbf{y}_{m+1:n}, \mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:m}\big) - \log p\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big) = \\
&\Big[\log p\big(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{i=1}^{m} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:m}\big) + \sum_{i=m+1}^{n} \log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z})\Big] - \log p\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big) = \\
&\log \frac{p\big(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{i=1}^{m} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:m}\big)}{q\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big)} + \sum_{i=m+1}^{n} \log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}) \; - \log \frac{p\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big)}{q\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big)} = \\
&\mathbb{E}_{q(\mathbf{z}|D)}\left[\sum_{i=1}^{n} \log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}) + \log \frac{p(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})}{q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n})}\right] + \\
&\mathbb{KL}\Big(q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n}) \Big\| p(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n})\Big) \geq \\
&\mathbb{E}_{q(\mathbf{z}|D)}\left[\sum_{i=1}^{n} \log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z}) + \log \frac{p(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})}{q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n})}\right] = \\
&\sum_{i=m+1}^{n} \mathbb{E}_{q(\mathbf{z}|D)}\big[\log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z})\big] - \mathbb{E}_{q(\mathbf{z}|D)} \log \frac{q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n})}{p(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})} = \\
&\sum_{i=m+1}^{n} \mathbb{E}_{q(\mathbf{z}|D)}\big[\log p(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{x}_{\mathcal{N}(i)}, \mathbf{z})\big] - \mathbb{KL}\Big(q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{j=1}^{n} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:n}) \Big\| q(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})\Big).
\end{aligned}
$$

In the order given above, the (in)equalities use the following: rewriting the log-likelihood via the posterior distribution, substituting the first term via the generative model, introducing a variational distribution $q(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n})$ (in our case, the encoder h and the aggregation a) to approximate the posterior $p(\mathbf{z} \mid \mathbf{x}_{1:n}, \mathbf{y}_{1:n})$, multiplying by $q\big(\mathbf{z} \mid \mathbf{x}_{1:n}, \bigcup_{i=1}^{n} \mathbf{x}_{\mathcal{N}(i)}, \mathbf{y}_{1:n}\big)$ and integrating over $\mathbf{z}$, the result that $\forall p, q.\ \mathbb{KL}(p\|q) \geq 0$, separating terms, approximating $p(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})$ with $q(\mathbf{z} \mid \mathbf{x}_{1:m}, \bigcup_{j=1}^{m} \mathbf{x}_{\mathcal{N}(j)}, \mathbf{y}_{1:m})$ and applying the $\mathbb{KL}$ definition.

C Task Descriptions

Cellular Automata For the Life-like family of cellular automata we sample $\sim\ \%$ of the possible 2^{18} rule sets at random (Bernoulli $p = 0.01$). For each selected rule set, we generate a random state on a 30×30 toroidal lattice (top connects to bottom, left connects to right) and check that every possible state is present (i.e. there are live cells with each of 0, 1, 2, ..., 8 neighbours and similarly a dead cell), then step forward one generation by applying the rule set to form the input-label pair.

Table 6: Dataset statistics by tasks. For transductive Cora there is a single citation network (i.e. 1 graph) from which subgraphs are sampled to produce training examples (of which the total possible number depends on the number of classes being used in the split e.g. for the 2-class task there are $\binom{11}{2} = 55$.) In the few-shot case, the train and test subgraphs are disjoint and neither features, labels, nor edges are observed from the test set during training. In all Cora tasks we use PCA to reduce the number of input features from 8710 to 100. For Proteins, we remove 6 graphs with more than one component or non-physical features (negative length). The density CA tasks (Voronoi, spherical Voronoi, small-world, scale-free) use generated graphs with the number of nodes being drawn from $[100, 200]$, we report the observed mean as generated by our seed.

Dataset	Task	Graphs	Mean-Nodes	Features	Classes
ShapeNet	Bag	76	2749.46	3	2
	Cap	55	2631.53	3	2
	Knife	392	2156.57	3	2
	Laptop	451	2758.13	3	2
	Mug	184	2816.97	3	2
	2-parts	1158	2,557.26	3	2
	Earphone	69	2496.70	3	3
	Guitar	787	2353.91	3	3
	Pistol	283	2654.22	3	3
	Rocket	66	2358.59	3	3
	Skateboard	152	2529.55	3	3
	Table	5271	2722.40	3	3
	3-parts	6628	2,665.34	3	3
	Airplane	2690	2577.92	3	4
	Car	898	2763.81	3	4
	Chair	3758	2705.34	3	4
	Lamp	1547	2198.46	3	4
	4-parts	8893	2,584.53	3	4
	Motorbike	202	2735.65	3	6
TUD	Proteins	1113	39.06	29	3
	Enzymes	600	32.63	18	3
	DHFR	467	42.23	3	9
	COX2	467	41.22	3	8
	BZR	405	35.75	3	10
PPISP		408	207.64	38	2
Cora	Transductive	1	19,793	100*	70
	Few-shot train	1	17,657	100*	11
	Few-shot test	1	2136	100*	11
Cellular Automata	Life-like	2659	900	2	2
	Voronoi	2700	149.81	2	2
	Spherical-Voronoi	2700	150	2	2
	Small-world (WS)	2700	149.54	2	2
	Scale-free (BA)	2700	149.34	2	2

Table 7: Class-ID information for the Cora class taxonomy. There are 11 disciplines collectively containing 70 classes. These IDs can be used to select classes as loaded by `CitationFull` from PyTorch Geometric.

Discipline	IDs
Information Retrieval	{0,1,4,12}
Databases	{2,10,28,42,44,46,60}
Artificial Intelligence	{5,8,9,11,14,22,33,34,48,53,54}
Machine Learning	{3,20,29,55,57,58,59}
Encryption and Compression	{6,15,26}
Operating Systems	{7,27,45,62}
Networking	{13,16,24,30}
Hardware and Architecture	{17,40,41,50,67,68,69}
Data-Structures Algorithms and Theory	{18,19,21,31,32,35,61,64,66}
Programming	{23,36,37,49,51,52,56,63,65}
Human Computer Interaction	{25,38,39,43,47}

For density-based rules we use birth/survival functions with either the form of the top-hat function:

$$R_0(d, k_1, k_2) = \begin{cases} 0 & \text{for } d < k_1, \\ 1 & \text{for } k_1 \leq d \leq k_2, \\ 0 & \text{for } d > k_2. \end{cases} \tag{12}$$

or $1 - R_0$, i.e.:

$$R_1(d, k_1, k_2) = \begin{cases} 1 & \text{for } d < k_1, \\ 0 & \text{for } k_1 \leq d \leq k_2, \\ 1 & \text{for } d > k_2. \end{cases} \tag{13}$$

The irregular graphs that the density-based rules operate on are generated using Scipy and NetworkX. In each case we sample the number of nodes uniformly from the interval $[100, 200]$. For the planar Voronoi the nodes are positioned at uniformly at random in the unit square and the tessellation is generated using SciPy.[1] For spherical-Voronoi the nodes are positioned uniformly at random over the surface of the sphere and the tessellation is generated using SciPy.[2] For small-world the graphs are generated using the Watts-Strogatz model with $p = 0.1$ and $k = 10$ i.e. the network is initialised in a ring-lattice connected to its 10 nearest-neighbours on the ring and then edges are rerouted with probability 0.1, using the NetworkX implementation.[3] For the scale-free case we use the Barabasi-Albert model with $m = 3$, using the NetworkX implementation.[4]

Cora We base our Cora tasks on the `CitationFull` dataset provided in PyTorch Geometric[5] which is loading the data used by Bojchevski and Günnemann[6], who in turn base their set on that originally gathered by Andrew McCallum of University of Massachussets Amherst.[7] Nodes are research papers with bag-of-word features (8710 words meet the threshold for inclusion by Bojchevski and Günnemann, which was given through correspondence with the authors as a minimum of appearing in 10 documents in the set) that use presence/absence rather than counts (multi-hot). Edges indicate that one of the papers cited the other, though we do not distinguish between citing/cited

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html`

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.SphericalVoronoi.html`

[3] `https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.random_graphs.watts_strogatz_graph.html`

[4] `https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html`

[5] `https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/datasets/citation_full.html`

[6] `https://github.com/abojchevski/graph2gauss`

[7] `https://people.cs.umass.edu/~mccallum/data.html`

and the graph is undirected. The papers belong to one of 70 topics within 11 disciplines of Computer Science, and we present the relevant class-ID information for splitting by discipline in Table 7. In the few-shot learning setup we separate out classes $\{29, 4, 10, 53, 26, 45, 30, 17, 21, 56, 47\}$ for validation and $\{59, 1, 42, 48, 15, 62, 16, 67, 61, 49, 38\}$ for testing, representing 14.94% and 10.79% of the total nodes, respectively. Each of these splits contains a class from every branch (hence 11 classes) with an effort made to ensure the class-to-branch ratios were also approximately 15% and 10%, with preferential selection for the test set and an allowance for producing largely connected subgraphs. For example, on the Encryption-branch there are three classes 15, 26, and 6, containing approximately one sixth, one third and one half of the nodes, respectively, with class 15 being selected for the test set and 26 for the validation set. Practically the connectivity allowance means selecting class 48 (12.1% of AI) rather than 22 (9.0% of AI) for the test set and class 29 (14.7% of ML) rather than 55 (14.9% of ML) for the validation set.

TUD Datasets Proteins and Enzymes are more commonly treated as graph-classification tasks, but there is an intermediate labelling of secondary structural elements (α-helices, β-sheets and β-turns) that can be used in the node classification setup. DFHR, COX2 and BZR consist of small libraries of small molecule inhibitors against each respective protein target (Dihydrofolate Reductase, Cycloxygenase-2 and the Benzodiazapene Receptor). In the typical graph-classification task, molecules are deemed active or inactive on the basis of a thresholded half-maximal inhibitory concentration measure determined through *in vitro* biochemical assays. The node-classification task considered here requires the model to predict node labels representing encodings of atom-type. Node features are *xyz* coordinates of the conformation provided in the datasets.

Protein-Protein Interaction Site Prediction This node-classification task utilises protein structural data collated in (Zeng et al., 2019), representing protein structures as graphs of interacting residues. Nodes are featurised with low dimensional embeddings of physicochemical properties (Meiler et al., 2001), encodings of secondary structure, solvent accessibility metrics, and position-specific scoring matrices which capture evolutionary information as protein-protein interaction residues have been shown to be evolutionarily conserved. Edge features represent one-hot encodings of intramolecular interaction types. Node labels indicate whether or not that amino acid takes part in an experimentally determined protein-protein interaction. Graphs are constructed using Graphein (Jamasb et al., 2020).

D Experimental and Model Details

D.1 Experiments

Recently introduced benchmark suites such as OGB (Hu et al., 2020) and those proposed by Dwivedi et al. (2020) aim to improve the quality of evaluation of graph-based models. However, we find they lack tasks that are appropriate for evaluating meta-learning frameworks of the kind we present, i.e. they do not host tasks that match the problem statement in Section 1. To this end, in addition to standard formulations of TUD (Kersting et al., 2020) and PPI (Zeng et al., 2019) tasks, we present new task formulations of the existing ShapeNet (Chang et al., 2015; Yi et al., 2016) and full Cora (McCallum et al., 2000; Bojchevski & Günnemann, 2018) datasets and entirely novel synthetic tasks based on cellular automata (Von Neumann et al., 1951; Wolfram, 1984; Turing, 1990).

D.1.1 Baselines and Model Details

We evaluate against a variety of baselines that collectively leverage all sources of information present in the tasks (featural, relational & contextual). This helps highlight where the advantages of the MPNP lie in a given setting. Table 8 from Appendix D indicates which baselines were run on each task and why. The **label propagation algorithm** (LP) (Zhu & Ghahramani, 2002) makes direct use the context labels (nodes are labelled by their neighbours, who label their neighbours, and so on) and is best suited to segmentation-like tasks. The strength of LP as a baseline is confirmed in recent work showing simple models based on label propagation are competitive with state-of-the-art GNN models (Huang et al., 2021). Where relevant, we include **guessing the most common context-label** (Mode), as this may significantly outperform the uniform-prior ($1/|C|$) for some tasks. **Graph neural networks** (GNNs) use training data in the inductive setup, but are not equipped to use the

context points provided at test time. GNNs are expected to perform well on tasks with fixed classes and little variation in the generative process across the set of datasets being modelled. We note that these models are not designed to handle arbitrary labelling tasks and their expected performance is bound by chance, i.e. $\mathbb{E}[\text{acc.}] = {}^1/_{|C|}$: as predictions do not depend on class labellings, for any given task example we can construct a set of equivalent tasks by permuting the labels, and over the set of permutations the average performance will be chance (a formal derivation is provided in Appendix F). As such, we do not include this baseline on these tasks. In our setup, the GNN consists of GCN layers with skip-connections (Appendix D.4 contains a detailed description of the architecture).

Non-message-passing **Neural Processes** (NPs) are limited only by their inability to leverage relational information between points, though this is, of course, a serious limitation in the settings we consider. We use the same **Message Passing Neural Process** and NP architectures for most Cora, ShapeNet and biochemical tasks, with the addition of Maxout layers (Goodfellow et al., 2013) for CA tasks. Other modifications are described with the experiment in which they are used, and full model details for each scenario are provided in Appendix D.

D.1.2 Fixed Labelling Tasks

We first consider tasks where the same set of classes appear in every example and the class labelling is 'fixed'. Inductive GNNs are designed for this setting and provide a useful baseline performance.

Two tasks are adapted from the **TUD collection** (Kersting et al., 2020): Enzymes and DHFR. The Enzymes dataset consists of proteins represented as networks of secondary-structural elements (α-helices, β-sheets, β-turns; SSEs) with biochemical features describing these units and edges between connected elements. DHFR is a library of small molecules that inhibit a particular protein, represented as graphs of atoms connected by bonds with spatial positions as features. Table 1 shows the MPNP narrowly outperforms the NP at the Enzymes task and by a much greater margin for DHFR, though in each case an inductive GNN is more successful. This suggests that the relational information present in the Enzymes dataset is of secondary importance to the featural information of the SSEs, and that there is limited variation over both datasets, given that an inductive model can perform well without any context points. Nevertheless, it is promising that MPNPs are able to use the relational information in DHFR to improve greatly on NPs.

The **Protein-Protein Interaction (PPI) Site Prediction** task involves predicting which nodes (amino acids) in an amino acid residue graph are involved in an experimentally-determined PPI (Zeng et al., 2019). Solving this task is thought to depend strongly on being able to use relational information, and there is great variation between examples. As expected following the TUD results, the MPNP excels in this setting, with SOTA-competitive results at plausible context rates presented in Table 2. The prefix 'R' indicates that the message-passing scheme has an edge-type dependency, as in the R-GCN (Schlichtkrull et al., 2018). Full details for this model are provided in Appendix D.5.

The **ShapeNet** repository (Chang et al., 2015; Yi et al., 2016) is a collection of large-scale 3D shapes, represented as point clouds for our applications.[8] We embed the points as a nearest-neighbours graph ($\mathbf{A}$) and use the (x, y, z) position as node features ($\mathbf{X}$). There are 16 object categories, each with a fixed number of parts, ranging from two to six. The labels have consistent meaning across datasets within a category. For example, we model the process that produces *chairs* with *arms, legs, seats* and *backs*, which we can consistently label $\{1, 2, 3, 4\}$.

Part labelling results are presented in Figure 2. We use the mean-Intersection-over-Union (mIoU) metric, which is standard for segmentation tasks: the ratio of overlap (TP) to the union (TP+FP+FN) is found for each part, and averaged (higher is better, T/F P/N = true/false positives/negatives). In 11 object categories, the MPNP outperforms the NP at more than 95% confidence across the entire context sampling range, and is the top-performing model over some of the sampling range in 13 out of 15 categories. At 30% sampling, label propagation dominates as expected. We do not include results for guessing-the-mode as this strategy is unsuited to the mIoU metric (performance is bound by $1/C$ for C-part objects as $\text{TP}_\text{C} = 0$ for all but one part.)

[8] There exist many techniques that make fuller use of the geometric information available, but for this proof-of-concept we consider only the simplest method.

Figure 3 shows the superior uncertainty-modelling capabilities of the MPNP (with extensive visualisations in the Appendix, Figure 6). In the first 3 plots, we **visualise the uncertainty predictions** for a *table* sample. Though the models achieve similar mIoU, the MPNP is only significantly uncertain at the borders between parts (a physically relevant uncertainty), whereas the NP is uncertain along the table-top edges, which are distant from any table-leg points in the internal geometry of the table. On the right, we present the results of an **active learning experiment** similar to that described by Garnelo et al. (2018a). At each step, the target with the greatest uncertainty is added to the context set (i.e. labelled) and predictions are repeated. This shows the power of useful uncertainty estimates in the MPNP.

We introduce tasks based on modelling **Cellular Automata**. Irregular graph-CAs have been used to study traffic networks (Malecki, 2017), social networks (Hunt et al., 2011), urban development (White, 1998) and logistics (Lopez et al., 2019), and cell dynamics (Bock et al., 2009). Our aim is to show how MPNPs extend available modelling capabilities, as existing baselines are likely to struggle in this setting. The model is provided with the states of some cells over a generation and tasked with evolving others. To evaluate generalisation, we prevent rule-set overlap in the train, validation and test sets. This contrasts with the existing work of Gilpin (2018), where the model learns a single rule-set, and that of Mordvintsev et al. (2020), who train a CA to produce a desired pattern. We provide an overview of these tasks, with full details given in Appendix C.

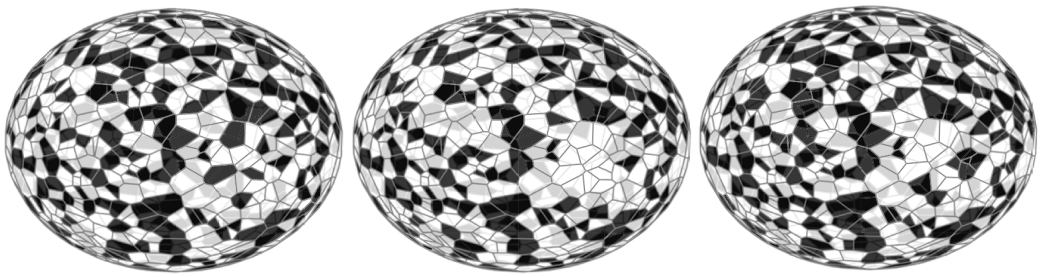

Figure 5: Three generations of a population-density CA on a spherical Voronoi network producing complex patterns in the cells. The MPNP receives the first state (nodes are cells, edges link bordering cells, features are $0/1$ according to cell state) and predicts cell states ($0/1$) after one transition.

Conway's Game of Life (Games, 1970) consists of cells in a 2D lattice governed by simple rules: cells become alive/are born (B) or stay alive/survive (S) depending on the number of living neighbours. The **Life-like family** of CA are the generalisations of these rules over any number of neighbours 0–8, defining 2^{18} variants. Neighbour counts can also be generalised to neighbourhood population-densities, and **density-based rules** can be adapted to irregular graphs and non-planar topologies. We consider single-interval rules, such that cells live or die based on being inside or outside a continuous range of population-densities, on small-world, scale-free, and spherical Voronoi networks (an example of the latter is shown in Figure 5).

State evolution results are presented in Figure 4. Here, the 'Population/State Mode' baselines are versions of 'guessing-the-mode' that output the most common label over the whole context set or by initial state, respectively. The NP is often able to match the state-mode strategy, but this is the ceiling to methods that do not take relational information into account. The MPNP is able to learn effective representations that generalise well to the disjoint test set for density-based rules. For density-based rules, MPNPs perform strongly across a variety of graph structures, while NPs are bound by simple strategies that guess the most common state change. Neither model is able to perform well for the Life-like family, despite the existence of a solution to this problem for MPNPs, outlined in Appendix G.

D.1.3 Arbitrary Labelling Tasks

Garnelo et al. (2018a) applied the CNP model in the arbitrary labelling setting, where each dataset includes samples drawn from a fixed number k of class types, where the total number of types

$K \gg k$. As the total number of classes could be very large and test examples may include unseen classes, using fixed-classes is infeasible. Instead, arbitrary labellings $(1, \ldots, k)$ are assigned on a per-dataset basis, and models are required to adapt accordingly.

The Cora-ML task is a widely used community detection benchmark. Papers are represented by bag-of-words vectors with edges indicating that one of the papers cited the other. Our task, **Cora-Branched**, is derived from a more complete dataset, with 70 classes over 11 computer science disciplines (McCallum et al., 2000; Bojchevski & Günnemann, 2018). There are 10 times as many classes and the bag-of-words feature vectors are tripled in length. Given a partially labelled subgraph of the network, the task is to label the rest. We consider the **transductive setup** (Yang et al., 2016; Veličković et al., 2017) where every class is observed during training and a **few-shot learning task** where, at test time, the models are presented with classes unobserved during training, to evaluate how well the learned representations generalise. Results for the transductive setting are presented in Table 3. Both models perform well in the low-sampling rate regime, indicating a strong feature signal, though the MPNP-c significantly outperforms the NP-c in every test, by up to 10% for 7-class. LP performs best at higher sampling rates and for more classes, as expected. Table 4 compares the quality of NP and MPNP representations in the few-shot learning context—the MPNP generalises better to unseen categories.

In the **ShapeNet mixed-category** setup, we model the process that produces n-part objects (say, $n = 4$ for *chairs* with *arms, legs, seats* and *backs*, as well as *airplanes* with *engines, bodies, tails* and *wings*.) Here, labels have consistent meaning only within a given realisation, so using a fixed ordering of labels implies a meaningless relationship between, say, *chair-backs* and *airplane-wings*. We thus provide an arbitrary permutation of class labels for each example.

Table 5 shows results for the mixed-class part-grouped ShapeNet task. The GNN struggles as expected, with performance below chance. Label propagation is the strongest performer at high sampling rates, with the MPNP-c and NP-c at a relative advantage with fewer context points. The NP-c performs best at 0.1% on 3-class, which may be due to category imbalances (80% of 3-part objects are tables) disrupting the MPNP-c. MPNP-c and label propagation otherwise divide the sampling range as top performers.

All models were trained on a Titan Xp GPU or an RTX 2080 GPU, with `torch.manual_seed(0)` across all experiments. An 80/20 train/test split was used for TUD datasets[9] and the ones provided by PyTorch Geometric[10] for all ShapeNet tasks. **The supplementary material includes code for all models and experiments described in this paper.**

D.2 MPNP

The architecture of the MPNP can be summarised as follows:

1. encoder: Linear(h), ReLU, $\{$MP(h), ReLU$\}\times T$, Linear(r);
2. global latent variable encoder: Linear(r), [Linear(z), Linear(z)] (mean & variance of $\mathbf{z}$);
3. decoder: Linear(h), ReLU, $\{$MP(h), ReLU$\}\times T$, Linear(h), ReLU, [Linear(C), Linear(C)] (mean & variance of $\hat{\mathbf{y}}$).

Across all experiments, the Adam optimiser is used to maximise the ELBO (i.e. minimise the sum of the negative log-likelihood and KL-divergence in equation 5).

D.2.1 TUD

On Proteins and Enzymes, the MPNP hyperparameters are $h = 64$, $r = 128$, $z = 256$; for the MPNP-c, $h = 64$, $r = 96$, $z = 288$; both have $T = 2$. On DHFR, COX2 and BZR, both MPNP and MPNP-c have $h = 64$, $r = 128$, $z = 256$, $T = 1$. We trained both models for 400 epochs with learning rate $7e \times 10^{-5}$ on all datasets except for Enzymes, where we used 700 epochs and learning

[9] `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html#torch_geometric.datasets.TUDataset`

[10] `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html#torch_geometric.datasets.ShapeNet`

Table 8: For each task that MPNPs were evaluated on, we indicate whether a baseline has been run or explain why we did not consider it necessary.

	NP	LP	GNN	Mode
TUD	✓	✓ (just to illustrate the low performance due to low label homophily—Table 1)	✓	✗ low label homophily
PPISP	✓	✗ same argument as above (biochemical data domain)	✓ (R-GCN)	✗ low label homophily
ShapeNet fixed	✓	✓	✓	✗ unsuitable for comparison with the mIoU metric
Cellular Automata	✓	✗ not better than chance (high variation across rules)	✗ not better than chance (for every CA rule there exists the opposite)	✓ (population, state)
Cora-Branched transductive	✓	✓	✗ not better than chance (classes shuffled for each sample)	✓
Cora-Branched few-shot	✓	✗ experiment is measuring generalisation of learned representations (this model does not learn from the training set)	✗ not better than chance (classes shuffled, previously unseen at test time)	✗ experiment is measuring generalisation of learned representations (this model does not learn from the training set)
ShapeNet arbitrary	✓	✓	✓	✗ unsuitable for comparison with the mIoU metric

rate 1×10^{-4}. For all datasets, we sample context and (additional) target points in the 10%–25% range.

D.2.2 SHAPENET

Across all experiments, $h = 64$, $r = 128$, $z = 256$, $T = 2$. The MPNP was trained for 400 epochs on fixed-class and 500 epochs on mixed-class tasks, with 5%–25% context and (additional) target points and a learning rate of 7×10^{-5}.

D.2.3 CORA

In both the transductive and few-shot settings, $h = 64$, $r = 64$, $T = 2$ and $z = N \times 64$ for N-classes. In the transductive setting the model is trained for 500 epochs where little if any overfitting is observed. In the few-shot setting the model is trained for 400 epochs. The model performs significantly better on the training classes in the few-shot case, though this is expected. A learning rate of 7×10^{-5} is used in both cases and we sample context and target points in the 10%–50% range.

D.2.4 CA

The CA models use a modified architecture that includes Maxout layers (Goodfellow et al., 2013) that can be summarised as follows:

1. encoder: MP(h), ReLU, $\{$Linear(h), ReLU$\}\times 3$, Maxout($h, 2$), Linear(r), ReLU;
2. global latent variable encoder: Linear(r), [Linear(z), Linear(z)] (mean & variance of $\mathbf{z}$);
3. decoder: MP(h), ReLU, $\{$Linear(h), ReLU$\}\times 3$, Maxout($h, 2$), (concatenation with $\mathbf{z}$), $\{$Linear(h), ReLU$\}\times 3$, [Linear(C), Linear(C)] (mean & variance of $\hat{\mathbf{y}}$).

Maxout layers use a pool-size of 2 and the decoder delays concatenation with $\mathbf{z}$ until after the Maxout layer (and the part before concatenation matches the encoder). For both the life-like and density-based settings, $h = 64$, $r = 64$, $z = 128$. The models are trained for 200 epochs with a learning rate of 1×10^{-4} and we sample context and target points in the 30%–50% range.

D.2.5 R-MPNP

The architecture of the R-MPNP can be summarised as follows:

1. encoder: Linear(h), ReLU, $\{$R-MP(h), ReLU$\}\times T$, Linear(r);
2. global latent variable encoder: Linear(r), [Linear(z), Linear(z)] (mean & variance of $\mathbf{z}$);
3. decoder: Linear(h), ReLU, $\{$R-MP(h), ReLU$\}\times T$, Linear(h), ReLU, [Linear(C), Linear(C)] (mean & variance of $\hat{\mathbf{y}}$).

Across all experiments, the Adam optimiser is used to maximise the ELBO (i.e. minimise the sum of the negative log-likelihood and KL-divergence in equation 5).

D.2.6 PPISP

The hyperparameters used are $h = 64$, $r = 64$, $z = 256$. Models were trained for 1000 epochs with a learning rate of 4×10^{-5}. We sample context and target points in the 10%–50% range.

D.3 NP BASELINE

The architecture of the NP consists of:

1. encoder: Linear(h), ReLU, Linear(h), ReLU, Linear(r);
2. global latent variable encoder: same as for the MPNP;
3. decoder: Linear(h), ReLU, Linear(h), ReLU, Linear(h), ReLU, [Linear(C), Linear(C)] (mean & variance of $\hat{\mathbf{y}}$).

The Adam optimiser is also used here to maximise the ELBO.

D.3.1 TUD

On Enzymes, the NP and NP-c hyperparameters are $h = 64, r = 128, z = 512$. On Proteins, we used $h = 64, r = 64, z = 512$ for the NP and $h = 64, r = 96, z = 288$ for the NP-c. On DHFR, COX2 and BZR, both NP and NP-c have $h = 64, r = 64, z = 512$. We trained both models for 400 epochs with learning rate $4e^{-5}$ on all datasets except for Enzymes, where we used 700 epochs. For all datasets, we sample 10%–25% context and (additional) target points.

D.3.2 SHAPENET

The same hyperparameters were used for all tasks: $h = 64, r = 64, z = 512$. The NP was trained for 400 epochs on fixed-class and 500 epochs on mixed-class tasks, with 5%–25% context and (additional) target points and a learning rate of $4e^{-5}$.

D.3.3 CORA

In both the transductive and few-shot settings, $h = 64$, $r = 64$ and $z = N \times 64$ for N-classes, matching the MPNP. The model is trained for 500 epochs in the transductive setting and 400 in the few-shot setting. A learning rate of 7×10^{-5} is used in both cases and we sample context and target points in the 10%–50% range, matching the MPNP.

D.3.4 CA

Changes are made to the NP architecture for the CA tasks to match the changes made to the MPNP for this task, with MP layers replaced with linear layers with $2h$ units to match the parameter count of the MP. Otherwise the parameters match that of the MPNP: $h = 64$, $r = 64$, $z = 128$. The models are trained for 200 epochs with a learning rate of 1×10^{-4}.

D.3.5 PPISP

The hyperparameters used are $h = 64$, $r = 64$, $z = 256$. Models were trained for 1000 epochs with a learning rate of 6×10^{-5}. We sample context and target points in the 10%-50% range, matching the R-MPNP.

D.4 GNN BASELINE

This model consists of 3 GCN[11] layers with learnable skip-connections; the operation of a layer is:

$$\mathbf{h}_{t+1} = \text{ReLU}\big(\mathbf{W}_{\text{skip}}\mathbf{h}_t + \text{GCN}(\mathbf{h}_t)\big). \tag{14}$$

We use $h = 64$ across all tasks and train the model for 500 epochs, with the Adam optimiser minimising the cross-entropy loss and a learning rate of $1e^{-4}$. The context and target ranges are as previously described, for each dataset. Note that this model does not make use of the context labels.

D.5 R-GCN BASELINE

The model consists of 3 RGCN[12] layers; the operation of a layer is:

$$\mathbf{h}_{t+1} = \text{ReLU}\big(\text{RGCN}(\mathbf{h}_t)\big). \tag{15}$$

We use $h = 64$ and train the model for 400 epochs, with the Adam optimiser minimising the cross-entropy loss and a learning rate of 7×10^{-5}. This model leverages edge features in the message-passing steps but does not make use of context labels.

E NUMERICAL RESULTS AND UNCERTAINTY PLOTS

In this section, we present the numerical results used to generate the CA and ShapeNet plots in the main text. Tables 9, 10 and 11 show the ShapeNet single-category performances, whereas Table 12 provides the Cellular Automata results. Figure 6 illustrates additional uncertainty visualisations for other classes in ShapeNet, reinforcing the finding that the estimates produced by MPNPs are semantically relevant.

Table 9: Numerical mIoU results for the MPNP on ShapeNet single-category tasks ($\mu \pm \sigma$).

	0.1%	1%	5%	10%	30%
Bag	71.08 ± 3.78	75.12 ± 1.44	75.57 ± 0.89	76.05 ± 0.13	73.21 ± 0.72
Cap	64.00 ± 4.28	68.76 ± 4.32	73.05 ± 0.92	69.42 ± 1.34	67.41 ± 0.47
Knife	79.82 ± 0.25	87.34 ± 1.47	89.93 ± 0.46	90.39 ± 0.34	90.34 ± 0.24
Laptop	90.39 ± 0.31	95.94 ± 0.17	96.71 ± 0.19	96.75 ± 0.00	97.07 ± 0.12
Mug	75.90 ± 1.37	85.63 ± 2.27	87.80 ± 1.02	88.70 ± 1.20	88.14 ± 0.02
Earphone	49.80 ± 4.45	57.14 ± 1.99	59.41 ± 1.44	55.59 ± 1.22	55.35 ± 0.70
Guitar	77.95 ± 0.61	89.12 ± 0.31	92.33 ± 0.25	92.75 ± 0.31	93.17 ± 0.11
Pistol	67.68 ± 0.95	82.51 ± 0.60	85.82 ± 0.29	85.57 ± 0.46	86.48 ± 0.16
Rocket	54.61 ± 0.21	56.03 ± 0.48	60.78 ± 0.74	64.26 ± 2.48	62.90 ± 0.04
Skateboard	41.67 ± 2.01	51.75 ± 0.76	55.10 ± 0.13	53.44 ± 1.05	52.33 ± 0.15
Table	75.19 ± 0.81	83.64 ± 0.30	85.80 ± 0.04	85.94 ± 0.01	86.61 ± 0.03
Airplane	58.32 ± 0.82	81.55 ± 0.16	86.68 ± 0.06	87.32 ± 0.05	87.90 ± 0.00
Car	43.08 ± 0.91	69.02 ± 1.45	76.56 ± 0.37	78.02 ± 0.12	78.53 ± 0.10
Chair	72.29 ± 0.00	86.73 ± 0.32	89.88 ± 0.26	90.22 ± 0.00	90.70 ± 0.03
Lamp	61.80 ± 1.31	79.44 ± 0.00	84.03 ± 0.15	84.45 ± 0.25	84.89 ± 0.01
Motorbike	27.54 ± 1.36	48.10 ± 1.09	53.94 ± 0.16	53.17 ± 0.38	53.76 ± 0.02

[11] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.GCNConv

[12] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.RGCNConv

Table 10: Numerical mIoU results for the NP on ShapeNet single-category tasks ($\mu \pm \sigma$).

	0.1%	1%	5%	10%	30%
Bag	52.62 ± 0.00	54.20 ± 2.23	52.87 ± 0.35	53.06 ± 0.12	53.46 ± 0.13
Cap	45.08 ± 6.01	56.88 ± 0.60	55.21 ± 0.40	59.67 ± 1.18	57.94 ± 0.71
Knife	72.84 ± 0.33	87.02 ± 0.65	89.49 ± 0.12	89.68 ± 0.31	89.12 ± 0.20
Laptop	82.42 ± 2.05	93.49 ± 0.24	96.07 ± 0.38	96.46 ± 0.12	96.72 ± 0.02
Mug	68.52 ± 1.71	80.95 ± 0.41	84.92 ± 0.61	84.58 ± 0.97	85.57 ± 0.01
Earphone	36.42 ± 7.28	47.98 ± 1.06	47.94 ± 0.68	47.79 ± 0.04	49.04 ± 0.24
Guitar	69.00 ± 0.36	83.41 ± 0.64	87.83 ± 0.50	88.12 ± 0.16	89.11 ± 0.01
Pistol	63.84 ± 2.02	70.42 ± 0.79	70.64 ± 0.15	71.94 ± 0.22	71.79 ± 0.17
Rocket	56.71 ± 2.41	59.76 ± 0.69	63.69 ± 0.54	62.50 ± 0.58	63.85 ± 0.31
Skateboard	32.13 ± 1.71	41.84 ± 0.07	40.78 ± 0.03	40.36 ± 0.19	40.25 ± 0.09
Table	76.53 ± 0.21	83.11 ± 0.08	84.18 ± 0.08	84.20 ± 0.12	84.26 ± 0.01
Airplane	44.92 ± 0.06	78.05 ± 0.22	83.05 ± 0.02	83.65 ± 0.02	84.04 ± 0.05
Car	38.86 ± 0.43	57.90 ± 1.33	63.89 ± 0.08	64.73 ± 0.14	65.55 ± 0.46
Chair	69.68 ± 1.14	84.78 ± 0.51	87.35 ± 0.10	87.69 ± 0.11	87.81 ± 0.01
Lamp	57.04 ± 1.73	71.88 ± 0.54	75.40 ± 0.45	75.49 ± 0.19	76.19 ± 0.01
Motorbike	21.28 ± 1.41	25.44 ± 0.05	25.66 ± 0.04	25.69 ± 0.16	25.73 ± 0.05

Table 11: Numerical mIoU results for GCN and `labelprop` on ShapeNet single-category tasks ($\mu \pm \sigma$). Note that the GCN does not use the context labels and thus produces deterministic outputs.

	GCN	`labelprop`				
	0.1% / 1% / 5% / 10% / 30%	0.1%	1%	5%	10%	30%
Bag	69.76	54.62 ± 1.88	70.10 ± 4.35	86.16 ± 1.05	90.45 ± 1.10	95.67 ± 0.54
Cap	65.85	47.76 ± 5.07	74.19 ± 3.53	84.97 ± 0.49	88.83 ± 0.62	93.43 ± 0.41
Knife	79.61	57.48 ± 3.40	88.82 ± 0.56	93.70 ± 0.36	95.01 ± 0.24	97.03 ± 0.10
Laptop	94.23	58.61 ± 2.64	88.16 ± 0.64	93.76 ± 0.17	95.46 ± 0.12	97.34 ± 0.05
Mug	85.40	47.44 ± 1.48	74.93 ± 2.42	88.15 ± 0.45	91.09 ± 0.71	94.19 ± 0.15
Earphone	49.76	36.35 ± 2.29	66.68 ± 1.96	78.45 ± 1.21	82.50 ± 0.73	88.24 ± 0.27
Guitar	89.01	37.85 ± 1.69	78.79 ± 1.97	92.06 ± 0.20	94.10 ± 0.20	96.44 ± 0.09
Pistol	76.72	39.80 ± 1.22	69.09 ± 1.56	83.25 ± 0.94	87.02 ± 0.23	92.39 ± 0.21
Rocket	56.31	38.43 ± 3.14	59.84 ± 7.51	80.95 ± 0.52	85.67 ± 1.15	91.53 ± 0.37
Skateboard	57.19	32.83 ± 1.57	56.59 ± 1.35	80.33 ± 0.71	84.91 ± 0.78	90.76 ± 0.35
Table	76.54	42.22 ± 0.53	68.88 ± 0.27	83.32 ± 0.08	86.83 ± 0.06	91.16 ± 0.07
Airplane	79.50	20.81 ± 0.21	60.48 ± 0.33	79.77 ± 0.13	84.50 ± 0.07	90.47 ± 0.04
Car	71.95	19.20 ± 0.38	45.91 ± 0.99	67.85 ± 0.38	75.33 ± 0.31	85.36 ± 0.05
Chair	77.84	33.06 ± 0.62	70.97 ± 0.27	87.04 ± 0.12	90.65 ± 0.07	94.65 ± 0.02
Lamp	53.78	58.61 ± 2.64	88.16 ± 0.64	93.76 ± 0.17	95.46 ± 0.12	97.34 ± 0.05
Motorbike	46.18	15.66 ± 1.14	38.46 ± 1.28	63.11 ± 1.46	74.05 ± 0.51	85.51 ± 0.36

Table 12: Numerical accuracy results for the Cellular Automata tasks ($\mu \pm \sigma$).

	MPNP			NP		
	10%	30%	100%	10%	30%	100%
Small-world	$\mathbf{88.14 \pm 0.82}$	$\mathbf{95.09 \pm 0.45}$	$\mathbf{97.33 \pm 0.57}$	77.28 ± 4.85	78.97 ± 4.87	79.59 ± 4.83
Scale-free	$\mathbf{84.73 \pm 2.87}$	$\mathbf{93.18 \pm 3.25}$	$\mathbf{95.49 \pm 3.47}$	74.84 ± 0.73	76.91 ± 0.43	77.57 ± 0.47
Voronoi	$\mathbf{83.13 \pm 2.77}$	$\mathbf{90.01 \pm 5.16}$	$\mathbf{92.39 \pm 6.26}$	73.96 ± 4.09	76.09 ± 4.46	76.70 ± 4.34
Spherical Voronoi	$\mathbf{82.91 \pm 2.66}$	$\mathbf{91.68 \pm 3.90}$	$\mathbf{94.93 \pm 4.57}$	74.00 ± 4.12	75.81 ± 4.21	76.53 ± 3.86
Life-like	63.81 ± 2.03	65.40 ± 2.73	65.77 ± 2.85	61.38 ± 0.77	62.40 ± 0.06	62.62 ± 0.03
	Population mode			State mode		
	10%	30%	100%	10%	30%	100%
Small-world	68.40 ± 0.11	69.53 ± 0.13	70.10 ± 0.00	80.34 ± 0.13	81.62 ± 0.11	82.11 ± 0.00
Scale-free	66.54 ± 0.42	67.64 ± 0.12	68.25 ± 0.00	74.46 ± 0.17	76.30 ± 0.12	76.90 ± 0.00
Voronoi	67.36 ± 0.32	68.71 ± 0.04	69.37 ± 0.00	76.54 ± 0.17	78.17 ± 0.07	78.74 ± 0.00
Spherical Voronoi	66.09 ± 0.29	67.31 ± 0.07	67.91 ± 0.00	76.46 ± 0.20	78.13 ± 0.04	78.72 ± 0.00
Life-like	62.08 ± 0.07	62.55 ± 0.02	62.69 ± 0.00	$\mathbf{84.22 \pm 0.04}$	$\mathbf{84.48 \pm 0.02}$	$\mathbf{84.57 \pm 0.00}$

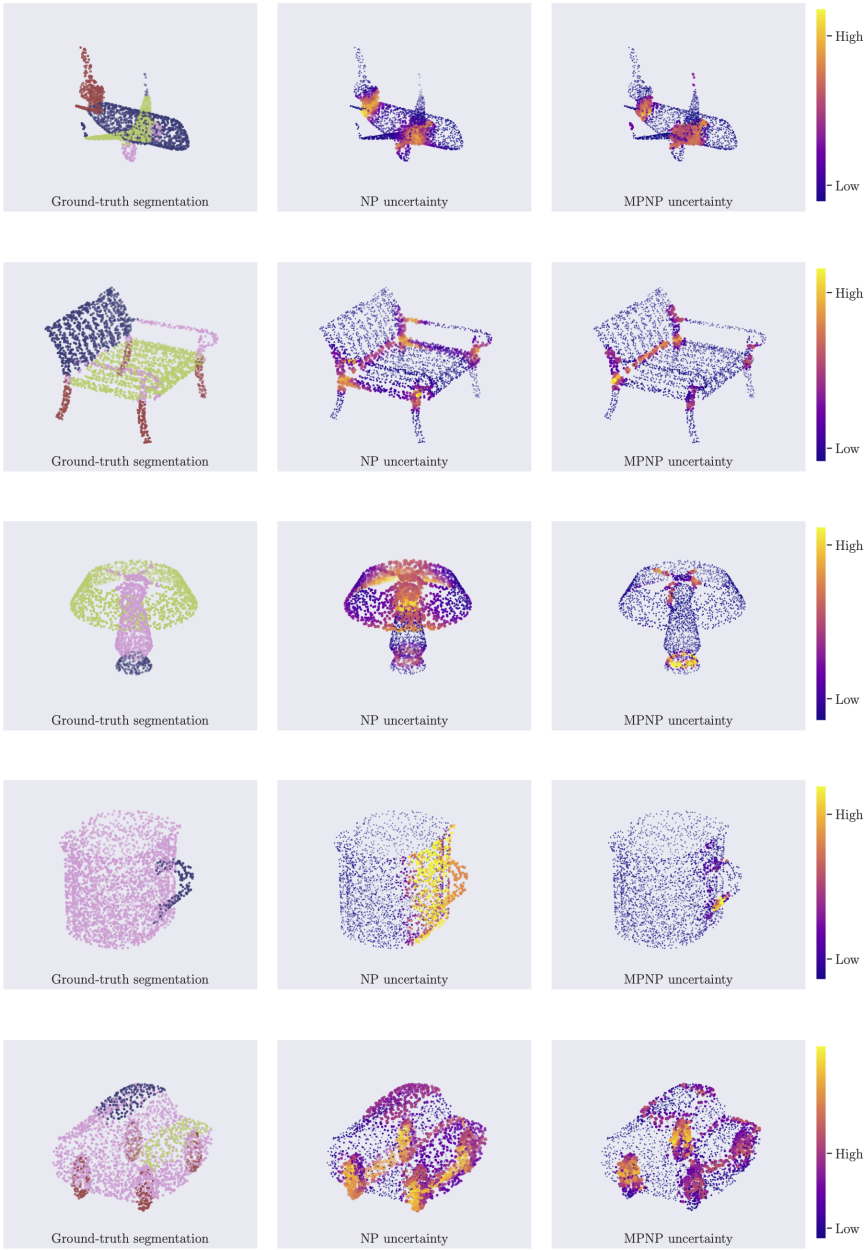

Figure 6: Uncertainty visualisations on examples from the airplane, chair, lamp, mug and car categories. In each case the MPNP is able to better localise the uncertainty to semantically relevant locations (i.e. border regions). The NP tends to be uncertain in large simple volumes, having the entire handle side of the mug being very uncertain, for instance. Similar effects are seen for the top of the lamp, the car axles, and the edges of the chair seat. The airplane is generally harder, with borders between the wings, fuselage and engines occurring in a relatively compact region, though we still see better localisation in the tail.

F INDUCTIVE GNNS WITH ARBITRARY LABELLING

When introducing the baselines in the Experiments section, we noted that the expected performance of inductive GNNs in the arbitrary labelling setting is no better than chance. This is because the predictions of such a model do not depend on the labelling scheme and for any particular labelling of a task we can produce a set of equivalent tasks by permuting the labels. First consider the two class case: outputs are either 1 or 2 and labels are either A or B, giving the mutually exclusive, collectively exhaustive groups $\{1\mathrm{A}\}, \{1\mathrm{B}\}, \{2\mathrm{A}\}, \{2\mathrm{B}\}$, which we normalise to sum to 1 by dividing by the number of examples. In the case that $(\mathrm{A}, \mathrm{B}) = (1, 2)$, the accuracy is:

$$\mathrm{acc}_{\mathrm{AB}} = \{1\mathrm{A}\} + \{2\mathrm{B}\}$$

and if the labels are permuted:

$$\mathrm{acc}_{\mathrm{BA}} = \{2\mathrm{A}\} + \{1\mathrm{B}\}$$

which average to:

$$\mathrm{acc}_{\mathrm{mean}} = \frac{\{1\mathrm{A}\} + \{2\mathrm{B}\} + \{2\mathrm{A}\} + \{1\mathrm{B}\}}{2} = \frac{1}{2}.$$

Generalising, outputs are in $1, ..., \mathrm{N}$ and labels in $\{\mathrm{A}, ..., \Omega\}$, for the matrix of pairs:

$$\begin{bmatrix} 1\mathrm{A} & \dots & 1\Omega \\ \vdots & \ddots & \\ \mathrm{NA} & & \mathrm{N}\Omega \end{bmatrix}$$

with the sum of all these elements being 1. There are N! permutations of the arbitrary labelling, and therefore N! equivalent tasks. Each term in the matrix appears in $(\mathrm{N}-1)!$ accuracy sums (with that term fixed, there are $\mathrm{N}-1$ free terms with $(\mathrm{N}-1)!$ permutations), so the mean accuracy is:

$$\mathrm{acc}_{\mathrm{mean}} = \frac{(\mathrm{N}-1)!\,(1\mathrm{A} + \cdots + \mathrm{N}\Omega)}{\mathrm{N}!} = \frac{(\mathrm{N}-1)!}{\mathrm{N}!} = \frac{1}{\mathrm{N}}.$$

G AN MPNP SOLUTION TO THE LIFE-LIKE FAMILY

The Life-like rules can be viewed as 18 separate rules that act in parallel, one for each neighbourhood count (9) for each state (2) and hence the 2^{18} variants noted in the main text (experimental section). A solution can be built using the concatenation encoder where the first steps describe the situation being observed at a given node as a one-hot encoding in an 18-element vector, and then summarises these using a max aggregator (or sum or mean with corrections later) and then concatenating by whether the cell lives or dies (i.e. concatenate by class). The max aggregation gives all the observed conditions that lead to a cell being alive in the next generation, and all those that lead to a cell being dead[13]. This representation is then used without modification as the latent variable. The decoder first extracts the observation to the format used by the encoder and then compares it with the latent variable, if it matches a condition found in the living-half of the latent variable, then the cell is alive in the next generation, if it matches a condition in the dead-half then the cell dies or stays dead.

The non-obvious parts are producing a one-hot encoding from a scalar (the neighbourhood count is produced simply by the MP) and checking the decoder observation against the latent variable. One-hot encodings of length N can be produced using Maxout layers as follows. First using a 2-pool Maxout as:

$$\text{Maxout}(x) = \max_j \big((W_1 x + b_1)_j, (W_2 x + b_2)_j\big),$$

setting $W_1 = -1$ and $W_2 = 1$, $b_1 = (-1, 0, ..., N-2)$ and $b_2 = (-1, -2, ..., -N)$. The elements of this function take the form of the max of $(-x + j - 1)$ and $(x - j - 1)$ which is a 'v' with unit slopes centred at j with a minimum value of -1. If we follow the Maxout with a linear layer $(-I)$ and a ReLU activation, we can first flip the 'v' and then flatten the edges to give a triangular hat centred at j with height 1. Thus, if $x = 2$ the first element (zeroth) is 0, the second element is 0,

[13]This could be compressed further using the fact that the rules are deterministic and do not overlap.

the third element is 1 and the rest are 0s. In this way, the first parts of the encoder and decoder can accurately represent the observed states. To compare an observation against the latent variable, we can take the sum of the observation and latent and subtract 1s (i.e. an AND).