Group-Invariant Machine Learning on the Kreuzer-Skarke Dataset

Christian Ewert[a], Sumner Magruder[b], Vera Maiboroda[c,d], Yueyang Shen[e], Pragya Singh[f], Daniel Platt[g,*]

[a]*German Center for Neurodegenerative Diseases (DZNE), Bonn, Germany*
[b]*Yale University, New Haven, CT, USA*
[c]*University Paris-Saclay, Gif-sur-Yvette, France*
[d]*Alternative Energies and Atomic Energy Commission (CEA) Paris-Saclay, Irfu/DPhP, Gif-sur-Yvette, France*
[e]*Statistics Online Computational Resource, Computational Medicine and Bioinformatics, University of Michigan, Ann Arbor, MI, USA*
[f]*Department of Computing, Imperial College London, London, UK*
[g]*Department of Mathematics, Imperial College London, London, UK*

Abstract

We use supervised machine learning to predict Hodge numbers for Calabi-Yau threefolds encoded by reflexive polyhedra. The Hodge number is invariant to the order of the vertices and the swapping of axes. Incorporating these properties, i.e. the invariance of column and row permutations for a matrix containing the polyhedron's vertices, promises better performance for Hodge number prediction. In fact, we find that machine learning models that incorporate these symmetries outperform models that do not. We compare approaches incorporating different degrees of invariance and improve the state-of-the-art on the dataset.

Keywords: Machine Learning, Invariant Machine Learning, Calabi–Yau, String Compactifications, Hodge Number, Kreuzer-Skarke

1. Introduction

Calabi-Yau threefolds are widely studied objects in pure mathematics and string theory. Heterotic string theory is ten-dimensional and Calabi-Yau threefolds are candidates to construct string compactifications down to four dimensions, with the hope of modeling real-world phenomena. However, most such Calabi-Yau manifolds do not satisfy certain consistency conditions dictated by physics (He et al., 2014, Section 3) and are therefore not suitable as models for string theory. Through the definition of "favourable manifold", one datum that enters the checks of the consistency conditions are the Hodge numbers of the Calabi-Yau manifold (He et al., 2014, p.7). Many Calabi-Yau threefolds can be obtained as codimension one submanifolds of toric 4-folds, which are encoded by a reflexive polytope obtained as the image of their moment map. There are $473,800,776$ reflexive polytopes in dimension 4, which give rise to an even larger number of Calabi-Yau threefolds and a complete list of these polytopes was published by Kreuzer and Skarke. The Hodge numbers of all Calabi-Yau threefolds corresponding to reflexive polytopes have been computed explicitly, but other string theoretic consistency conditions have only been checked for a few Calabi-Yau manifolds. Checking these consistency conditions is a computationally hard problem and it is desirable to have a fast method to identify the most promising candidates before running full computations. We study the problem of predicting Hodge numbers as a model case to demonstrate that machine learning is suitable as a fast method for this type of check.

Deep learning architectures are often built to make use of the data structure of the inputs to generate an output. For example, multi-layer perceptrons (MLPs) are used for vector-valued inputs and a trained network relies on the fixed order of the vector components for accurate inference. Convolutional neural networks (CNNs) have been designed to implement translation invariance on image- or tensor-valued inputs. While reflexive polytopes are also encoded as matrices (a stack of their vertices), the Hodge number is invariant to column and row permutations corresponding to a reordering of vertices and a swapping of axes. Incorporating invariance of the output with respect to some action on the inputs promises better performance. In a project at the *London Geometry and Machine Learning (LOGML) Summer School 2022*, we tested various models with different degrees of invariance. In this paper, we compare the performance of the different approaches. Python code for the experiments in this article is available at `www.github.com/danielplatt/kreuzer-skarke-ML`.

2. Related Work

Several supervised machine learning models have been trained to predict Hodge numbers of Calabi-Yau manifolds. This line of research was started by He (2017), where a shallow, fully-connected neural network was used to learn Hodge numbers of complete intersection Calabi-Yau threefolds. Following this, more sophisticated neural networks (Bull et al., 2018, 2019; Erbin and Finotello, 2021) were applied to the problem, improving the original prediction accuracy. Like the reflexive polytopes dataset, this problem is invariant under the action of a large symmetry group. Aslan et al. (2023) applied group-invariant machine learning models to the same dataset further

*D.P. is the corresponding author (E-Mail: daniel.platt.berlin@gmail.com)

improving performance. Berglund et al. (2021) trained a neural network to predict the Hodge numbers of Calabi-Yau threefolds arising from polytopes of the dataset by Kreuzer and Skarke.

3. Dataset and Methods

3.1. Dataset

In Kreuzer and Skarke (2000), the 473,800,776 reflexive four-dimensional polyhedra were classified, giving rise to (at least) this many Calabi-Yau threefolds. For each reflexive polytope, the dataset lists some information about the dual polytope alongside the Euler characteristic and Hodge numbers of the corresponding Calabi-Yau threefold. In this paper, we consider the subset of around 78,000 reflexive polyhedra in dimension four with 26 vertices and their corresponding first Hodge numbers. The polyhedra are encoded as matrices $M \in \mathbb{R}^{4\times 26}$ (26 vertices stacked in column vectors) and the considered subset includes 35 different Hodge numbers $h^{1,1}(M)$, i.e. 6-39 and 42. We use the subset as an example but we expect similar results for other numbers of vertices. We randomly partition the dataset into 80% training data and 20% test data once and use this split for every training and evaluation. The Hodge number $h^{1,1}$ is invariant to column and row permutations of M. On one hand, applying a column permutation merely changes the order of vertices in M but leaves the polyhedron unchanged. On the other hand, applying a row permutation yields an isomorphic polyhedron. Thus, the function $M \mapsto h^{1,1}(M)$ is invariant under the action of the group $S_4 \times S_{26}$, where S_n denotes the symmetric group on a set of n elements. For mathematical background on the dataset, we refer the interested reader to Section A.2.

3.2. Preprocessing

In addition to the dataset above which we refer to as *original dataset*, we obtain group-invariant features from the dataset by applying the preprocessing step from Aslan et al. (2023, Section 2.2), denoted by π and referred to as *preprocessed dataset*.[1] The rationale behind this is that precomposing any model $\phi : \mathbb{R}^{4\times 26} \to \mathbb{R}$ with an approximately group-invariant map $\pi : \mathbb{R}^{4\times 26} \to \mathbb{R}^{4\times 26}$ yields an approximately group-invariant composition $\phi \circ \pi$. In practice, we implemented this as follows: instead of training our machine learning models ϕ on matrices $M \in \mathbb{R}^{4\times 26}$, we instead used the training data $\pi(M)$. We also build a permuted version of the original dataset where columns and rows of the matrices are randomly permuted (denoted as M_p) and refer to this as the *permuted dataset*. For the last dataset, we apply the aforementioned preprocessing π to obtain group-invariant features on the permuted dataset ($\pi(M_p)$) and refer to this as *preprocessed permuted dataset*.

3.3. Algorithms

Random Guess (baseline). For each polytope, we randomly assign one of the 35 Hodge numbers appearing in the dataset.

Majority Class Only (baseline). For each polytope, we choose the most frequent Hodge number in the dataset, i.e. the Hodge number 16.

MLP with reduced input (baseline). Berglund et al. (2021) trained a neural network to predict $h^{1,1}$ from features invariant to column and row permutations, e.g. the number of vertices in the polytope and its dual. While they trained their method on matrices $M \in \mathbb{R}^{4\times k}$ with arbitrary k, we use their results on the entire dataset as the first baseline in our subproblem where $k = 26$ and refer to it as *MLP with reduced input*.

XGBoost. We used the gradient-boosted decision tree implementation of Chen and Guestrin (2016). All inputs were flattened, we trained with a maximum depth of 6 and used 50% of the $4 * 26 = 104$ features to build each tree. The $L1$ regularisation factor was 1 and the model was trained for a maximum of 60000 epochs and with early stopping using patience 100. The model was trained on the regression objective with Tweedie loss.

CNN. This neural network consists of a convolutional neural network (CNN) and subsequent multi-layer perceptron (MLP). The CNN consists of 2D convolution layers with kernel size 2 and 32 and 64 channels respectively, ReLU activations, and 2×2 max-pooling layers. The MLP consists of two hidden linear layers with 128 and 50 neurons respectively and ReLU activations. The network was trained for 1000 epochs using the Adam optimiser with early stopping using patience 100. The model is trained on the classification objective with cross-entropy loss.

Vision Transformer. We use the transformer neural network from Dosovitskiy et al. (2021) which has approximately 25 million parameters over 19 layers and uses 4 attention heads. For the attention heads, we use a patch size of 4×4. The $\mathbb{R}^{4\times 26}$ input matrix is augmented to $\mathbb{R}^{52\times 52}$ via stacking of random row and column permutations. The model was trained for 60 epochs using the Adam optimiser on the classification objective with cross-entropy loss.

Invariant MLP. This model (Hartford et al., 2018) contains three equivariant layers with 256 channels and cross-connections between layers followed by sum-pooling of each channel and five fully-connected layers with (256, 256, 256, 256, 35) neurons. We trained the model with the Adam optimiser and batch size 32 for 300 epochs with early stopping using patience 12. This model is invariant to column and row permutations of the input and uses a mean squared error loss in a regression setting.

PointNet++. The PointNet++ model (Qi et al., 2017b) is a neural network architecture designed for the processing of point clouds extending PointNet (Qi et al., 2017a) with local instead of global information aggregation via graphs. We used the implementation from Fey. The model consists of four linear layers with a message-passing mechanism for local feature aggregation. Each of these layers is followed by a ReLU activation and the block of these layers is followed by a global aggregation of

[1] We use the implementation from `github.com/danielplatt/Fundamental-Domain-Projections`.

features which are used by a single subsequent linear layer to make predictions. We trained the model using the Adam optimiser and batch size 64 for 300 epochs with early stopping using patience 15. This model is invariant under column permutations, but it is *not* invariant under row permutations, as the coordinates of the vertices are the inputs to the first layer of the network. It was trained with a cross-entropy loss function.

MLP with inv. features. This model consists of a column- and a row-encoder and a single decoder. Each encoder contains a single MLP that is applied independently to each column or row of the input. Per column or row, the respective encoder obtains a feature of length 256. Max-pooling operations are applied separately across column and row features similar to the pooling in PointNet (Qi et al., 2017a). The column-encoder obtains a feature invariant to column permutations while the row-encoder obtains a feature invariant to row permutations. Both features are added and processed by the decoder into the Hodge number prediction. The MLPs in the encoder contain two hidden and one output layer with 256 neurons while the MLP in the decoder contains three hidden layers with 256 neurons and the output layer. The decoder MLP uses batch normalization before the activation. Except for the very last layer, all layers have a ReLU activation. The model was trained in a classification setting with a cross-entropy loss function, Adam optimizer, and batch size 32 for 300 epochs with early stopping using patience 12.

Model	Accuracy Original Dataset	Accuracy Permuted Dataset
Random Guess	2.86%	2.86%
Majority Class Only	11.28%	11.28%
MLP with reduced input*	46.89%	46.89%
XGBoost	55.02%	29.70%
XGBoost+π	70.63%	59.84%
CNN	56.71 ± 0.38%	30.78 ± 0.49%
CNN+π	71.98 ± 0.61%	61.28 ± 0.35%
Vision Transformer	62.06 ± 0.44%	43.70 ± 0.60%
Vision Transformer+π	69.00 ± 0.48%	61.02 ± 0.63%
Invariant MLP	79.30 ± 0.90%	78.45 ± 0.92%
Invariant MLP+π	78.15 ± 1.73%	77.96 ± 1.50%
PointNet++	95.04 ± 0.39%	86.56 ± 0.52%
PointNet+++π	95.15 ± 0.97%	90.75 ± 0.44%
MLP with inv. features	**96.86 ± 0.31%**	92.04 ± 0.54%
MLP with inv. features+π	96.66 ± 0.30%	**95.37 ± 0.37%**

* This model was trained and evaluated on a superset containing polytopes with a wide variety of vertices, not just 26 vertices. Furthermore, the inputs contain information about the polytopes' dual (see method description for details).

Table 1: Test accuracies for the task of predicting the first Hodge number of a Calabi-Yau threefold given by a reflexive polytope. Given are the mean performance and sample standard deviation over five runs. "+π" denotes that the machine learning model was trained on preprocessed data as outlined in Aslan et al. (2023, Section 2.2). We evaluate the methods' performance on the original dataset and a version of the dataset where the columns and rows of each input matrix are randomly permuted.

4. Results and Discussion

In this section, we compare the methods described in Section 3.3 on the test sets of the datasets described in Section 3.2: the original and permuted datasets as well as group-invariant features based on these datasets, i.e. the preprocessed (denoted by $+\pi$) and preprocessed permuted datasets. For every method, we train one instance on the original dataset and one on the preprocessed dataset and report the accuracies on the respective corresponding test-sets. In addition and without retraining, we compare these instances on the permuted versions of the datasets. We list the accuracies in Table 1 and visualise these results in Figure 1. Note that the two models *MLP with reduced input* and *Invariant MLP* are already invariant under $S_4 \times S_{26}$, so composing with π makes no difference.

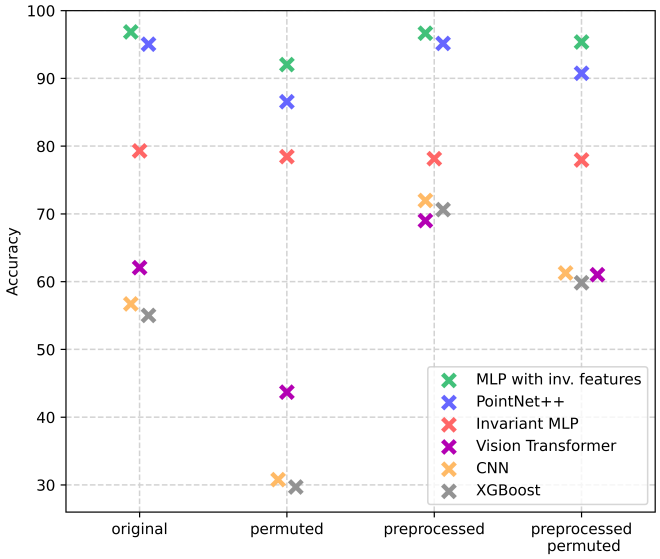


Figure 1: Performance summary of models across the four different datasets: the *original* dataset, a random permutation of the original dataset (*permuted*), the group-invariant preprocessing applied to the original (*preprocessed*) and applied to the randomly permuted dataset (*preprocessed permuted*).

We make the following observations:

1. Improved accuracy: The baseline achieved 46.89%; our best model achieves 96.86%.
2. Classification seems to perform better than regression: While it might seem more natural to formulate the task of predicting Hodge numbers as a regression task, some teams found that a formulation as a classification task, i.e. a multi-label output, performed better. Our best methods also use a classification setting.
3. Accuracy declines significantly (29.6-46%) if inputs are permuted and models are not permutation-invariant: Models such as *XGBoost*, *CNN*, and *Vision Transformer* are not by design permutation-invariant and have also not been trained with suitable data augmentation. As expected, their performance declines when permuted versions of the inputs are presented (see Table 1).

4. Building in group invariance improves performance: We notice that the models *MLP with inv. features*, *Invariant MLP*, and *PointNet++* perform better than other models that have no group-invariance built in. More specifically, the *Invariant MLP* is invariant under row and column permutations, *PointNet++* is invariant under column permutations, and the *MLP with inv. features* contains parts to extract features invariant under row and column permutations but not both.
5. Preprocessing step π improves performance: We note that making the models more group-invariant by applying the preprocessing step π increases performance for all models that are not already fully group-invariant on the task in which input matrices were randomly permuted. For the models *XGBoost*, *CNN*, and *Vision Transformer* performance improves more than two times the sample standard deviation, for the other models the performance change (a decrease in two cases) is less than two times the sample standard deviation. For the models *MLP with inv. features* and *PointNet++*, performance is further improved by applying the preprocessing step π for the scenario with random permutations.
6. Partially invariant models outperform fully invariant models: We observe that the two partially invariant models *MLP with inv. features* and *PointNet++* perform better than the fully group-invariant model *Invariant MLP*. We assume that partially invariant models are a compromise between *performance* and *generalizability* (to input changes that they are invariant to). On one hand, designing a neural network with built-in invariance involves enforcing the retrieval of invariant features which can introduce bottlenecks and over-constrain its architecture which, in turn, negatively affects its performance. On the other hand, neglecting the invariances, architectures can have sufficient capacity without bottlenecks but lack the desired invariances and generalizability. Therefore, partially invariant models perform well because they are not overconstrained and additionally benefit from increased generalizability caused by their partial invariance.

5. Conclusion

Our experiments show that the Hodge numbers of Calabi-Yau manifolds defined by reflexive polytopes can be predicted with good accuracy using machine learning.

Furthermore, we observe that building group invariance into the machine learning models improves performance: we see that models that are by definition invariant or partially invariant perform better than models that are not. We also see that models that are not group-invariant benefit from applying a group-invariant preprocessing step in most cases.

Interestingly, our strongest models are those that are partially invariant but not invariant under the full group. We assume that this is the case because the partially invariant models are a compromise between *performance* and *generalizability* (to the input changes that they are invariant to). More specifically, partially invariant models perform well as their architecture is not overconstrained to implement full invariance and benefit from increased generalizability through their partial invariance.

Acknowledgments

This project was started during the *London Geometry and Machine Learning 2022* summer school, and we thank the organisers for creating such a positive and productive environment. We thank Benjamin Aslan and David Sheard for helpful conversations. DP gratefully acknowledges funding from the Simons Collaboration in Special Holonomy. YS acknowledges funding from Statistical Online Computational Resources (SOCR). PS thanks Imperial College London and Goldman Sachs, where she worked previously, for their support. The work was carried out independently and neither Goldman Sachs nor Imperial was involved as an official partner.

Declaration of Competing Interests

The authors have no competing interests to declare.

References

Aslan, B., Platt, D., Sheard, D., 2023. Group invariant machine learning by fundamental domain projections, in: Sanborn, S., Shewmake, C., Azeglio, S., Di Bernardo, A., Miolane, N. (Eds.), Proceedings of the 1st NeurIPS Workshop on Symmetry and Geometry in Neural Representations, pp. 181–218. URL: https://proceedings.mlr.press/v197/aslan23a.html.

Batyrev, V.V., 1994. Dual polyhedra and mirror symmetry for calabi-yau hypersurfaces in toric varieties. Journal of Algebraic Geometry 3, 493–535.

Berglund, P., Campbell, B., Jejjala, V., 2021. Machine Learning Kreuzer–Skarke Calabi–Yau Threefolds. doi:10.48550/arXiv.2112.09117.

Bull, K., He, Y.H., Jejjala, V., Mishra, C., 2018. Machine learning CICY threefolds. Physics Letters B 785, 65–72. doi:10.1016/j.physletb.2018.08.008.

Bull, K., He, Y.H., Jejjala, V., Mishra, C., 2019. Getting CICY high. Physics Letters B 795, 700–706. doi:10.1016/j.physletb.2019.06.067.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery. pp. 785–794. doi:10.1145/2939672.2939785.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N., 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, in: International Conference on Learning Representations. URL: https://openreview.net/pdf?id=YicbFdNTTy.

Erbin, H., Finotello, R., 2021. Inception neural network for complete intersection Calabi–Yau 3-folds. Machine Learning: Science and Technology 2, 02LT03. doi:10.1088/2632-2153/abda61.

Fey, M., . Point Cloud Classification. URL: https://colab.research.google.com/drive/1D45E5bUK3gQ40YpZo65ozs7hg5l-eo_U.

Fulton, W., 1993. Introduction to Toric Varieties. volume 131 of *Annals of Mathematics Studies*. Princeton University Press. doi:10.1515/9781400882526.

Hartford, J., Graham, D., Leyton-Brown, K., Ravanbakhsh, S., 2018. Deep Models of Interactions Across Sets, in: Dy, J., Krause, A. (Eds.), Proceedings of the 35th International Conference on Machine Learning, pp. 1909–1918. URL: https://proceedings.mlr.press/v80/hartford18a/hartford18a.pdf.

He, Y.H., 2017. Machine-learning the string landscape. Physics Letters B 774, 564–568. doi:10.1016/j.physletb.2017.10.024.

He, Y.H., Lee, S.J., Lukas, A., Sun, C., 2014. Heterotic model building: 16 special manifolds. Journal of High Energy Physics 2014. doi:10.1007/jhep06(2014)077.

Kreuzer, M., Skarke, H., . CY/4d: reflexive polyhedra in 4 dimensions. URL: http://hep.itp.tuwien.ac.at/~kreuzer/CY/.

Kreuzer, M., Skarke, H., 2000. Complete classification of reflexive polyhedra in four dimensions. Advances in Theoretical and Mathematical Physics 4, 1209–1230. doi:10.4310/atmp.2000.v4.n6.a2.

Moon, K.R., van Dijk, D., Wang, Z., Gigante, S., Burkhardt, D.B., Chen, W.S., Yim, K., van den Elzen, A., Hirn, M.J., Coifman, R.R., Ivanova, N.B., Wolf, G., Krishnaswamy, S., 2019. Visualizing structure and transitions in high-dimensional biological data. Nature Biotechnology 37, 1482–1492. doi:10.1038/s41587-019-0336-3.

Qi, C.R., Su, H., Mo, K., Guibas, L.J., 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition 2017, IEEE Computer Society. pp. 77–85. doi:10.1109/cvpr.2017.16.

Qi, C.R., Yi, L., Su, H., Guibas, L.J., 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

Yau, S.T., 2008. A survey of Calabi-Yau manifolds. Surveys in differential geometry. Vol. XIII. Geometry, analysis, and algebraic geometry: forty years of the Journal of Differential Geometry, Int. Press. doi:10.4310/sdg.2008.v13.n1.a9.

A. Appendix

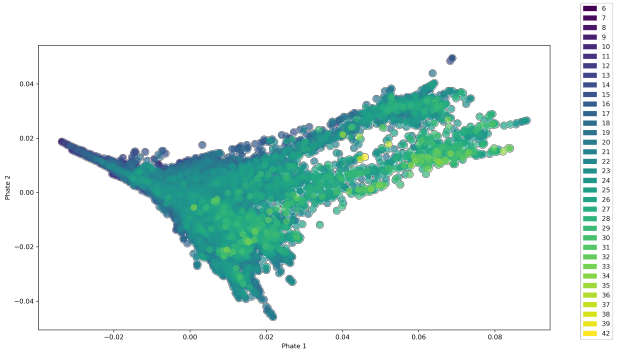
A.1. Visualisation

To better understand the reflexive polytopes dataset we employ dimensionality reduction techniques to visualize the low dimensional embedding of input matrices. In Figure A, we provide plots of input data under the *Potential of Heat-diffusion for Affinity-based Trajectory Embedding* (PHATE) (Moon et al., 2019). Data points are coloured according to their first Hodge number, though this Hodge number is *not* used in the computation of the dimensionality reduction. PHATE was chosen specifically due to its ability to preserve both the high-dimensional global and local distances in low dimensions.

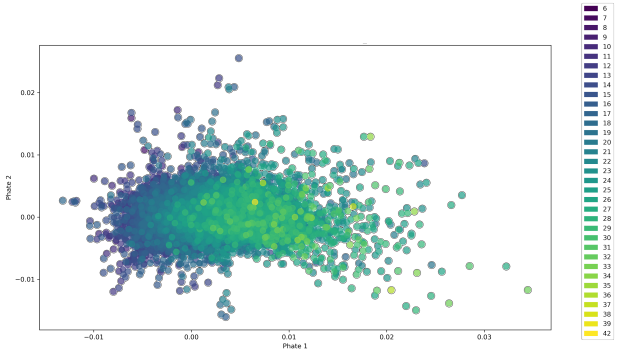
In the embeddings, Hodge numbers generally increase along the first axis. For the original dataset, PHATE creates a thin region of similar input matrices with low Hodge numbers. This structure disappears when PHATE embeddings of input data with random row and column permutations applied are computed. For both the original and the permuted dataset, applying the group-invariant preprocessing step from Aslan et al. (2023, Section 2.2) only has a minor effect on the plot of the dimensionality reduction. It seems to lead to areas of matrices with low Hodge numbers that are slightly better separated from the rest of the input data, though we are unable to quantify this. For example, for the original data, the thin region of matrices corresponding to low Hodge numbers is elongated.

A.2. Mathematical background

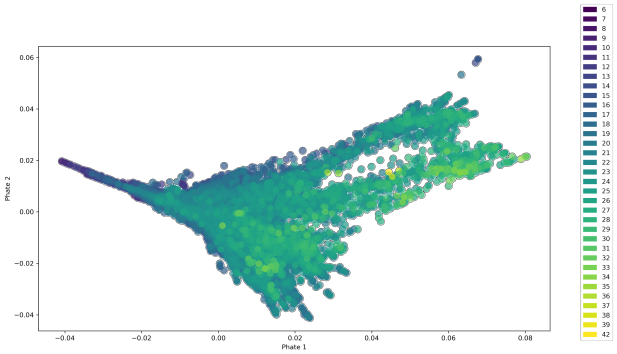
Calabi-Yau threefolds are complex manifolds of complex dimension 3. Many ways to construct them are known (see survey by Yau (2008, Section 2.5)). One construction is the following: a reflexive polyhedron defines a toric variety, see e.g. the textbook by Fulton (1993, Section 1.5). A hypersurface in this toric variety, suitably desingularised, is then a Calabi-Yau manifold, which was first observed by Batyrev (1994). If one applies this construction to four-dimensional reflexive polyhedra, then the resulting Calabi-Yau manifold is of complex dimension 3.



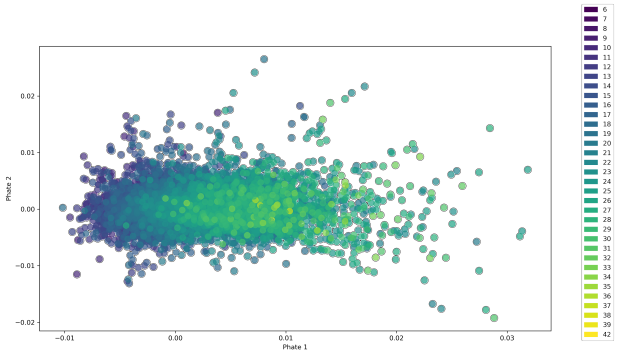
(a) Original Dataset



(b) Permuted Dataset



(c) Preprocessed Dataset



(d) Preprocessed Permuted Dataset

Figure A: The PHATE embeddings of inputs. Each point represents one $\mathbb{R}^{4\times 26}$ matrix, which in turn encodes a Calabi-Yau manifold. Points are coloured according to the first Hodge number of the corresponding Calabi-Yau manifold. The Hodge numbers are not used during the computation of the embeddings. (a) Embedding of the original dataset, (b) embedding of the original dataset after random row and column permutations were applied to the input matrices, (c) embedding after a group-invariant preprocessing step was applied to the original data, (d) embedding after a group-invariant preprocessing step was applied to the randomly permuted data.