END-TO-END NEURAL PERMUTATION PROGRAM SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Permutations are the most general abstraction describing finitary actions affecting finitely many elements, yet also the simplest class of programs imaginable. We propose a neural model that, given input-output pair examples, finds both a minimal set of atomic operations and the programs mapping inputs to outputs in terms of these atoms. Our model, DISPER, achieves 100% program reconstruction accuracy when the atoms are known and performs well even when tasked with identifying distinct groups of atomic operations in a single configuration. Further, DISPER is capable of reconstructing all groups of order less than 10 with recall of 89.5% and consistently produces high-quality minimal atom sets across a variety of permutation program induction scenarios.

1 INTRODUCTION

Permutations are a powerful abstraction that enables the description of those actions on finitely many elements that are themselves finite in their nature. More broadly, anytime a finite set of elements is being manipulated by actions which are invertible and have deterministic compositions, permutations can be employed to model the situation. This is a practical consequence of Cayley's theorem, which states that any group is isomorphic to a permutation group.

The various results of group actions on a collection of elements are often schematised by listing them in a tuple. A permutation P can then be partially or completely described by giving the tuple before and after applying P. For example, the input 4-tuple $x = [1, 2, 3, 4]$ and output $y = [2, 1, 4, 3]$ characterise the permutation $P = (12)(34)$, represented in cyclic notation, as P is the only permutation such that $y = Px$. Forming a complete description of P from one input-output pair is only possible when all elements of x, y are distinct. It would not be possible to identify P uniquely if $x = [1, 1, 3, 4], y = [1, 1, 4, 3]$ as x, y would tell us nothing about the effect P has in the span of the first two elements of x. In general, multiple pairs are needed if elements can be repeated within a tuple.

Permutations can be seen as the simplest class of programs possible, characterised by languages entirely recognisable by machines with a single tape of fixed size. There is no notion of control flow and no explicit concept of memory, although limited memory can always be simulated by fencing out a segment of the permutation that will be ignored in reading of the computation's result.

Despite their simplicity, the space of all permutations grows super-exponentially with the number of elements. Fortunately, all complex permutations can ultimately be seen as a composition of a chain of atomic (primitive) operations or *atoms*, of whom there is generally comparatively handful. As an example, all cyclic k-right-shifts of the 5-tuple $[1, 2, 3, 4, 5]$ (for example $[4, 5, 1, 2, 3]$ or $[5, 1, 2, 3, 4]$) are simply the atomic cycle (12345) applied k times. Let w be the number of elements being acted on, with the after-action state characterised by their positions in a w-tuple. Then there are $w!$ possible actions, each representable as a composition of at most $w - 1$ distinct atoms that may be used arbitrarily many times. This follows from the Cayley's theorem and the fact that adjacent transpositions span the symmetric group S_w.

We view permutations as a natural starting point for end-to-end program synthesis. Permutations are powerful enough to capture intricate phenomena at scale but still simple enough to admit systematic analysis and evaluation. We therefore put them at the centre of our study and set their reconstruction from examples of input-output pairs as our aim.

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ be atomic permutations. A permutation program P is a composition of powers of atoms in arbitrary order P: $a_{i_1}^{j_1} a_{i_2}^{j_2} \cdots a_{i_k}^{j_k}$ for $a_{i_\bullet} \in \mathcal{A}$. Let $\mathcal{P}$ be a finite set of permutation programs, $\mathcal{D} := \{ (x, y) : y = Px \text{ for } P \in \mathcal{P} \}$ a dataset of input-output examples. Our goal is to find a set of atomic permutations $\mathcal{A}'$ from knowing only $\mathcal{D}$ such that that for each pair $(x, y) \in \mathcal{D}$ we can use $\mathcal{A}'$ to form a program that permutes x into y, and further, such that the cardinality of $\mathcal{A}'$ is minimal or at least very small.

We present DISPER, a neural model that succeeds at this goal in multiple settings. We note that the same model architecture routinely achieves 100% accuracy (100% recall and closure overlap, cf. Section 4) in the simplified, *synthesis-only* scenario, where the atomic operations are known upfront. But, in order to succeed at the wider *atomisation-synthesis* objective where atoms are not known, our model must be able to iteratively form hypotheses about the atoms and at the same time already use them in the synthesis of candidate programs. In other words, DISPER has to both "disentangle" larger permutations into smaller, (near-)ground-truth atomic operations, and learn to selectively combine them to reconstruct original programs mapping x to y. DISPER's key characteristic is that it does so without employing an external search strategy or forming Bayesian priors about feasibility of various atoms – in contrast to classical program synthesis approaches, DISPER is an end-to-end neural model.

We evaluate DISPER against a set of natural baseline models, trained to output a sequence of atomic permutations directly. We find that DISPER outperforms the baselines, and further, that it benefits from architectural simplicity in its decision logic. We also note that the problem of decomposing a set of permutations into a set of smaller, atomic permutations is a problem conceptually similar to but substantially different from disentanglement as seen for example in computer vision. In particular, the order of factors of a successfully disentangled encoder representation of an image does not matter, whereas in the case of permutations, the order of the atomic or "factor" permutations has an effect on the resulting permutation. Hence, models such as variational autoencoders have no straightforward application in this context.

Our contributions are:

- the introduction of DISPER, a novel neural network model for neural permutation synthesis that identifies a set of primitive operations and simultaneously uses them for synthesis of programs,
- the evaluation of DISPER against baselines set by common neural network architectures,
- the collection of input representations and evaluation datasets for the problem, and
- the application of this model to the task of finding a minimal generating set for all groups of order ≤ 10.

In Appendix A, we also give an analysis concerning the effects of loss hyperparameters controlling the flow of the learning process, the quality of proposed atom sets, and the lengths of the synthesised programs.

2 RELATED WORK

Program induction (or synthesis) has lately been gaining traction and admitting approaches in fields ranging from algorithmic theory and verification to statistics and machine learning.

Algorithmic program synthesis (Bodík & Jobstmann, 2013), traditionally considered a problem in deductive theorem proving, has recently been looked at as a search problem with constraints such as a logical specification of the program behaviour (Feng et al., 2018), syntactic template (Alur et al., 2018; Desai et al., 2016; Polozov & Gulwani, 2015), and, most recently, previously discovered program fragments and utility functions (Huang et al., 2021; Ellis et al., 2021). Several new methods also combine enumerative search with deduction, aiming to rule out infeasible sub-programs as soon as possible (Feng et al., 2017; Feser et al., 2015; Polikarpova et al., 2016). While some of the above methods use neural approaches, none of them provide an end-to-end neural solution, and mostly use deep networks as means of acceleration of otherwise enumerative program search. A further negative result comes in the form of the immense difficulty to leverage deep neural networks to learn even the simplest of computational patterns in a way that generalises out of the training distribution

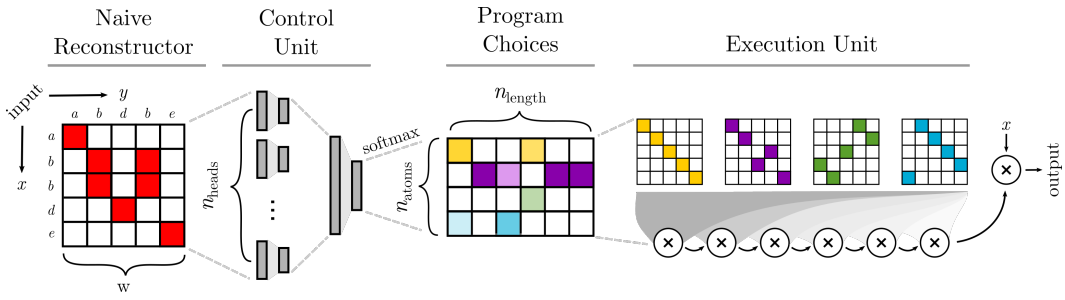


Figure 1: A schema of the architecture of the DISPER neural network. x, y-pairs of tuples $w = 5$ elements wide with x drawn from $\{a, b, c, d, e\}$ with replacement enter the network through the naive reconstructor unit. From there on, the multi-headed control unit decides on the combination of learned atoms that is to be used, producing a program consisting of $n_{\text{length}} = 6$ instructions. Each instruction is a softmax choice over $n_{\text{atoms}} = 4$ atoms being learned. Finally, the choice mixtures of atoms are multiplied to give a program candidate, which is in turn applied to x to produce the output.

(Belcak & Wattenhofer, 2022b). An end-to-end neural solution to combinatorial circuit synthesis (not permitting saving of states between phases of computation) has been proposed in Belcak & Wattenhofer (2022a). In all of this previous work, however, the atomic operations are provided in advance and only learned to be combined and composed to form longer programs.

The learning of permutations – a fundamentally very discrete concept – has been given only very little attention in deep learning research, with most of the work in the field being focused at the exact opposite goal of designing and training networks that are permutation-invariant (Mukhopadhyay et al., 2022), especially in the context of graph neural networks (Keriven & Peyré, 2019; Niu et al., 2020). Standing out is AutoShuffleNet (Lyu et al., 2020), which learns to permute image channels in a computer vision pipeline. Authors propose a Lipschitz-continuous loss that enforces double-stochasticity and report high accuracy of learned permutations. In Section 3, we give a loss that enforces only right-stochasticity and observe that our loss is very similar to the second term of the loss suggested by AutoShuffleNet. Our loss is not only Lipschitz continuous but also differentiable.

3 MODEL

Our model takes a dataset of input-output pairs of w-tuples (w standing for width) as input and produces a set of candidate atomic permutations $\mathcal{A}'$ as output. Individual programs as compositions of atoms can then be read out by presenting the model with particular input-output pairs.

The model consists of a neural network controlled by loss functions akin to those seen in Lyu et al. (2020) and disentangling variational auto-encoders Higgins et al. (2016); Chen et al. (2018). It is trained in a conventional, linear manner.

3.1 ARCHITECTURE

The neural network takes a pair of w-tuples x, y as input and produces a permutation of x for output. It consists of five units used in succession to produce the permutation that is eventually applied to x. See Figure 1 for an overview.

3.1.1 NAIVE RECONSTRUCTOR

The naive reconstructor attempts to recreate the permutation matrix that turned x into y. Considering x, y as two w-dimensional column vectors, it computes the naive reconstruction matrix NR by

$$\text{diff}\,(x, y) = x \otimes \mathbb{1}_{w\times 1} - \mathbb{1}_{w\times 1} \otimes y, \qquad \text{NR}_{ij}\,(x, y) = 1 - \frac{\text{diff}\,(x, y)^2_{ij}}{\max_{kl} \text{diff}\,(x, y)^2_{kl}},$$

where $\otimes$ denotes the outer product and $\mathbb{1}_{w\times 1}$ is the w-dimension column vector consisting solely of ones.

3.1.2 CONTROL UNIT

The control unit consists of n_{heads} heads and a combinator. Each head is a ReLU-activated network consisting of two layers of widths 24 and 8 respectively and densely connected to the output of naive reconstructors. The combinator has a hidden ReLU-activated layer $w_{\text{combinator}}$ neurons wide that spans across the outputs of head, and a linear final layer containing $n_{\text{atoms}} \times n_{\text{length}}$ neurons. The control unit is 4 layers deep in total.

3.1.3 PROGRAM CHOICES

The choices from among the atomic permutations are formed by reshaping the output of control unit into a $n_{\text{atoms}} \times n_{\text{length}}$-matrix and then taking softmax across the first (atom) dimension, yielding C. This can be interpreted as having a list of n_{length} instructions (columns of C), where each instruction is a soft choice from among the atoms.

3.1.4 ATOM MATRICES

Atoms $A_1, \ldots, A_{n_{\text{atoms}}}$ are simply learnable $w \times w$-matrices with softmax computed across their rows. The row softmax ensures that the matrices are all left-stochastic. Right-stochasticity (and therefore double-stochasticity) is then enforced with the help of the heterogeneity loss (cf. Section 3.2).

3.1.5 EXECUTION UNIT

The final unit of the network, the execution unit, takes the program choices, computes a sum of all atoms per program instruction weighted by the softmax choice, and then takes the matrix product. Denoting the i, j-coordinate of C by $(C)_{ij}$,

$$P = \prod_{i=1}^{n_{\text{length}}} \left(\sum_{j=1}^{n_{\text{atoms}}} (C)_{ij} A_j \right).$$

To produce the final output, the synthesised permutation P is applied to x.

3.2 LOSS

The total training loss is computed as a weighted sum of the reconstruction, eye divergence, right-stochasticity, and heterogeneity losses:

$$\text{L} = \text{RL} + \beta\text{EDL} + \gamma\text{RSL} + \delta\text{HL},$$

where $0 \leq \alpha, \beta, \gamma$ are hyperparameters of the model.

3.2.1 RECONSTRUCTION LOSS

To train the model to reconstruct the permutation output through atom matrices, we may use any loss that increases with growing disagreement between the elements of model output Px and y. In our settings, we use point-wise binary cross entropy when x, y are one-hot vectors encoding distinct elements, and mean squared error when x, y are tuples of integers. In general, any loss may be used as long as it differentiably captures the differences between distinct elements.

3.2.2 "EYE DIVERGENCE" LOSS

A loss is introduced to encourage the model to use fewer atom whenever possible, and this loss can be thought of as a naive counterpart to the Kullback-Leibler divergence commonly appearing in variational autoencoders Higgins et al. (2016); Chen et al. (2018). A typical example of its effect can be seen in Figure 2, where we observe gradual discardment of information in the second atom from the bottom once two sufficient generators have been established (60th epoch onwards).

The eye divergence loss is the mean of the squares of differences between the diagonals of atoms and the diagonal of the identity ("eye") matrix. Denoting $(A_k)_{ij}$ the i,j-coordinate of the k-th atom,

$$\text{EDL} = \frac{1}{n_{\text{atoms}} w} \sum_{k=1}^{n_{\text{atoms}}} \sum_{i=1}^{w} \left(1 - (A_k)_{jj}\right)^2$$

Thus, the contribution of any atom that is identity matrix is 0, and the contribution of any atomic operation that leaves no element unmoved is $\frac{1}{n_{\text{atoms}}}$.

3.2.3 Right-stochasticity loss

The right-stochasticity loss enforces the formation of right-stochastic matrices. Together with atom softmax giving left-stochasticity, the resulting atoms are doubly-stochastic and thus candidates for sharpening towards permutations.

$$\text{RSL} = \frac{1}{n_{\text{atoms}} w} \sum_{k=1}^{n_{\text{atoms}}} \sum_{j=1}^{w} \left(1 - \sum_{i=1}^{w} (A_k)_{ij}\right)^2$$

3.2.4 Heterogeneity loss

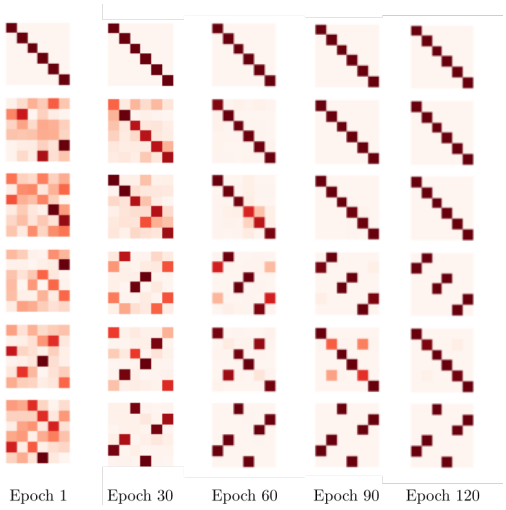


Figure 2: The evolution of atoms as the model instance learns to synthesize programs of D_6. *Horizontally*: training progress. *Vertically*: individual atomic matrices from among which program choices are made. Light/dark colours represent values close to 0/1. DisPer succeeds at finding a minimal generating set for the group, namely $\{(12)(34)(56), (135)(246)\}$. The former is the generator of reflections in the axis of symmetry, the latter is the rotation generator.

To encourage the use of fewer distinct atoms when synthesizing a program for given x, y, we increase the total loss proportionally to the number of different atoms used. This is to further support the model towards arriving at collections of atoms that are minimal or at least small.

$$\text{HL} = \frac{1}{n_{\text{atoms}}} \sum_{i=1}^{n_{\text{atoms}}} \sigma\left(K\left(1 - \sum_{j=1}^{n_{\text{length}}} (C)_{ij}\right)\right),$$

where σ is the logistic (sigmoid) function K influences the sharpness of the sigmoidal transition. Throughout our experimentation we kept $K = 10$ fixed. A simple $\max(\bullet, 0)$ could also be used in place of $\sigma(K\bullet)$, but we have decided for the latter to keep the transition gradual and differentiable around 0.

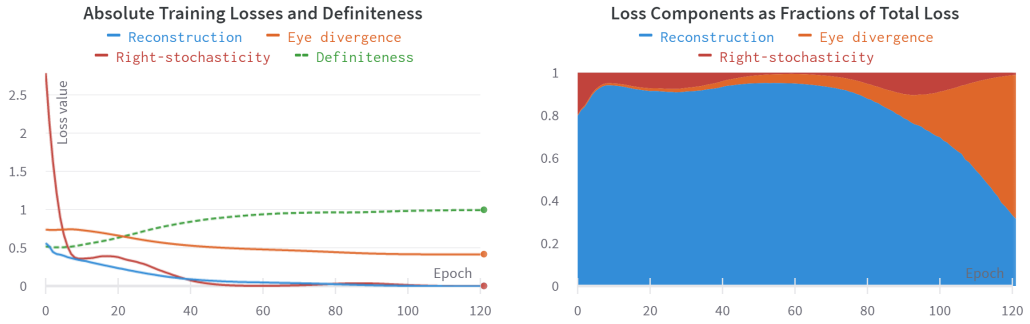
3.3 Training

Our architecture is largely indifferent to the input representation of input-output pairs, as long as the representation admits clear disambiguation between distinct elements by an appropriate, differentiable loss. We have tested our model on two natural element types, one-hot vectors, and integers drawn from a bounded range, and give our results in Section 4. Here we just note that the values of hyperparameters have to be adjusted in line with the scale of the reconstruction loss, which in turn depends on the loss function and input representation used.

Training of DisPer requires no special curriculum, and once the hyperparameters are set, it can be trained using the standard deep learning approach of batching and repeating training over the same dataset in epochs. Figures 2 and 3 show the progression of DisPer's training for all programs of D_6 – the dihedral group of order 6 representing the symmetries of an equilateral triangle – including the evolution of atoms, training curves, and ratios of the losses.

4 Experiments

In this section we systematically evaluate the abilities of DisPer across a variety of datasets and forms of input, and against baseline models.



(a) (b)

Figure 3: A quantitative look on the progression of DISPER's training for D_6 with $\beta = 0.05, \gamma = 0.05, \delta = 0$. The left figure gives the absolute values for individual loss components before scaling by hyperparameters. while the figure on the right shows the total loss in terms its components after β, γ, δ-scaling. We see that the right-stochasticity loss plays a significant role in the early stages of the training and later retires to obscurity. In the middle of the training the reconstruction dominates, and eye divergence takes over towards the end. Definiteness is a training metric (the average of maximum entries in each row of each atom) indicating training progress. We stopped the training at the 120th epoch and inspected the atoms with results shown in Figure 2.

Let $\mathcal{D} := \{ (x, y) : y = Px \text{ for some permutation } P \}$ be a set of input-output examples, let $\mathcal{A} := \{a_i, \ldots, a_k\}$ be fixed atoms of width w that are unknown to the model, and let $\mathcal{P}$ be a set of programs composed from $\mathcal{A}$ that give $\mathcal{D}$. So $\mathcal{A}, \mathcal{P}$ are the ground-truth atoms and programs for $\mathcal{D}$.

In our experience, the hyperparameters of our model can always be tuned to achieve full individual program class synthesis – that is, to find a single set of atoms that can be used to produce any program in $\mathcal{P}$. We wish to support this claim with a theoretical result in our future work and resort for now to evaluation in scenarios where the model hyperparameters are fixed but the ground-truth atom sets vary.

Our broader *evaluation task* here is therefore to find a set of candidate atoms $\mathcal{A}'$ which can be used to form as many programs in $\mathcal{P}$ as possible when evaluating across a range of different input-output pair sets $\mathcal{D}$ with hyperparameters fixed.

4.1 EVALUATION METRICS

We evaluate a quality of a candidate set of atoms $\mathcal{A}'$ by three metrics. The first, *recall rate*, is the proportion of $\mathcal{P}$ that can be arrived at by composing atoms of $\mathcal{A}'$. This simply measures how good of a set of atoms $\mathcal{A}'$ is with respect to $\mathcal{P}$. *Overlap* is the ratio of $\mathcal{P}$ to $\langle \mathcal{A}' \rangle$, where the latter denotes the algebraic closure of $\mathcal{A}'$ under composition (i.e. the smallest group generated by the elements of $\mathcal{A}'$). This is an indication of how well $\mathcal{A}'$ aligns with $\mathcal{A}$, and in the context of full group reconstruction also represents the "precision" of $\mathcal{A}'$ as a generating set for the group being reconstructed. Finally, we take *quality* to be the ratio of the cardinalities of $\mathcal{A}$ and $\mathcal{A}'$ – a measure of how far $\mathcal{A}'$ is from minimality.

4.2 EVALUATION DATASETS

Our experiments are run on three different evaluation datasets, each tailored to assess the model under a different set of circumstances.

- The *partial synthesis* dataset has been constructed by generating input-output pairs by programs of lengths 1-4 acting on tuples of size 5-9, formed by using at most 5 distinct atoms, each being a permutation of between 2 and 5 positions in the tuple.
- The dataset for *minimal atom-set finding* has been built by generating example pairs by programs consisting of cycles only. The cycles act on tuples of sizes 12-14, are between 1 and 3 in $\mathcal{A}$, and have each length between 2 and 4.

Data Form	Metric Type	Task Datasets								
		Partial Synthesis			Minimal Set			Group Reconstr.		
		recall	overlap	quality	recall	overlap	quality	recall	overlap	quality
UNIQUE-w	mean	0.711	0.195	0.925	**0.958**	**0.948**	**1.000**	0.873	0.397	0.511
	weighted	0.712	0.166	0.908	**0.947**	**0.947**	**1.000**	0.862	0.285	0.440
ARBITRARY-w	mean	0.647	0.181	0.940	0.825	0.687	0.891	**0.895**	**0.499**	**0.540**
	weighted	0.638	0.165	0.929	0.826	0.700	0.892	**0.879**	**0.451**	**0.487**
INTEGERS-100	mean	**0.917**	**0.933**	**0.994**	0.671	0.915	**1.000**	0.457	0.764	0.893
	weighted	**0.895**	**0.933**	**0.992**	0.634	0.920	**1.000**	0.310	0.766	0.919

Table 1: The results of the systematic evaluation of DISPER on the datasets and data forms of Section 4. **Emphasis** and **emphasis** mark the best results per dataset for unweighted and weighted (by cardinality of $\mathcal{P}$) statistics, respectively. The best hyperparameters per dataset and data form varied. For UNIQUE-w and ARBITRARY-w, these were in the ranges $\beta \in (0.15, 0.25), \gamma \in (0.05, 0.10), \delta \in (0.00, 0.01)$. For INTEGERS-100 these were scaled by a factor of 100 except for δ which was clamped at 1. All models were allowed to use $n_{\text{atoms}} = 12$, with $n_{\text{length}}, n_{\text{heads}} \in [5, 10]$.

- *Group reconstruction dataset* consists of input-output example collections $\mathcal{D}$ generated by acting on input tuples by the elements of all 18 groups of order ≤ 10.

The collections of input-output examples come in three forms:

- UNIQUE-w. Exactly w distinct tuples one-hot-encoding numbers $1, \ldots, w$ are the elements of the input tuple x. Each element appears in x exactly once.
- ARBITRARY-w. w tuples one-hot-encoding numbers $1, \ldots, w$ are chosen uniformly at random with replacement. Elements may thus appear in x multiple times or not appear at all.
- INTEGERS-100. w integers between 0 and 99 inclusive are drawn uniformly at random with replacement to form the input tuple x. Similarly to above, any integer may appear more than once.

UNIQUE-w pairs x, y always allow complete reconstruction of the permutation matrix that has been applied to them. This is not the case for ARBITRARY-w and INTEGERS-100, which further differ from each other by the difficulty of training. Data in the forms of UNIQUE-w and ARBITRARY-w admits training with binary cross-entropy loss, while INTEGERS-100 calls for a loss for an unbounded range (in our case mean squared error) and would generally take more computational resources to reach the same level of accuracy as UNIQUE-w or ARBITRARY-w. In line with the intuition, of the three data forms, data in UNIQUE-w would always lead to the fastest model training times.

4.3 SYSTEMATIC EVALUATION OF DISPER

The results of our evaluation of DISPER across all tasks and on all datasets are listed in Table 1. We observe that UNIQUE-w, ARBITRARY-w, and INTEGER-100 are each the best form for a different dataset task. In line with our intuition, uniqueness was particularly useful in minimal atom set finding, and the fine granularity of the INTEGER-100 was advantageous in partial synthesis while strongly disadvantageous in full group reconstruction. Overall, high recall, overlap, and quality scores were achieved for at least one data form for both partial synthesis and minimal atom set finding.

Full group reconstruction is a challenging task, requiring the model to keep a wide inventory of permutations to span all the possible actions of the group under reconstruction, but also demanding that in the end, the set of identified atoms is small and its closure does not outspan the group being learned. The overlap and quality scores were noticeably lower for this task. We give an overview of the scores per group attained by DISPER in Appendix B. Here and also from frequent manual inspection, we observed no general condition on the type of groups that could help identify groups that are difficult to fully reconstruct.

Model	Task Datasets								
	Partial Synthesis			Minimal Set			Group Reconstr.		
	recall	overlap	quality	recall	overlap	quality	recall	overlap	quality
DNN	0.175	0.111	0.350	0.191	0.120	0.387	0.160	0.110	0.290
1D CNN	0.560	0.759	0.876	0.612	0.890	0.951	0.732	**0.876**	0.512
2D CNN	0.908	0.730	0.997	**0.962**	0.932	0.991	**0.908**	0.451	0.600
RNN	0.470	0.782	0.921	0.532	0.841	0.931	0.390	0.327	0.379
Transformer Encoder	0.361	0.725	0.674	0.423	0.429	0.412	0.432	0.291	0.300
DISPER	**0.942**	**0.956**	**1.000**	**0.962**	**0.952**	**1.000**	0.900	0.621	**0.615**

Table 2: The results of the evaluation of DISPER against the baselines. For each model and task dataset, the results reported are from the best-performing data form. **Emphasis** marks the best results in terms of the weighted score.

4.4 EVALUATION AGAINST BASELINE MODELS

To assess the impact of architectural choices made in the design of DISPER, we evaluated it against a multi-layer fully-connected neural network, one-dimensional and two-dimensional convolutional neural networks, a recurrent neural network, and a transformer encoder. We describe these in more detail in Appendix C. The results of this evaluation are listed in Table 2. We see that DISPER outperforms the baselines in most evaluation instances, with the two-dimensional convolutional neural network coming in as a close second.

For the minimal generating set task, the recall performance of the two-dimensional convolutional neural network matches that of DISPER, but the two-dimensional CNN has lower overlap and quality, Notice also that for the group reconstruction task where the one-dimensional convolutional neural network produces a set of higher overlap than DISPER, both the recall and quality are lower, suggesting that the network managed to identify atoms of larger generating orbits in the group but from among predictions of atoms that were more in quantity but less often correct.

Observe that even the best DNN exhibited very weak performance when compared to other architectures, despite using sufficiently wide layers and being six layers deep. When viewed in contrast with the one-dimensional CNN, this suggests that the identification of the appropriate permutation to reconstruct requires more sophistication than possessed by fully-connected layers. We find the results of individual baseline model architectures on permutations to be aligned with the outcomes of the benchmarking effort for finitary abstractions (Belcak et al., 2022).

5 CONCLUSION

We have introduced a purely neural model – DISPER – for permutation program analysis and synthesis, and demonstrated its ability to learn useful atomic permutations and then use them to produce programs that correctly map input-output examples. We further evaluated it against a set of baseline models and found that it outperforms common architectures.

DISPER can be used both as a tool for identification of atomic permutations and as a synthesizer that produces a program for a given input-output pair. This duality of purpose arises from it simultaneously forming a set of atoms and attempting program reconstruction throughout its training.

We see permutations as a simple but rich class of programs for induction from examples and hope that our work will help to facilitate further advancements in neural program synthesis. To this end and in the interest of reproducibility we make our code and data available at *anonymised*.

REFERENCES

Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. Search-based program synthesis. *Communications of the ACM*, 61(12):84–93, 2018.

Peter Belcak and Roger Wattenhofer. Neural combinatorial logic circuit synthesis from input-output examples. In *2nd Workshop on Math-AI (MATH-AI @ NeurIPS)*, 2022a.

Peter Belcak and Roger Wattenhofer. Periodic extrapolative generalisation in neural networks. In *IEEE Symposium on Deep Learning*, 2022b.

Peter Belcak, Ard Kastrati, Flavio Schenker, and Roger Wattenhofer. Fact: Learning governing abstractions behind integer sequences. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17968–17980. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/72372ec86dd49238900fc0b68bad63f8-Paper-Datasets_and_Benchmarks.pdf.

Rastislav Bodík and Barbara Jobstmann. Algorithmic program synthesis: introduction, 2013.

Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems*, 31, 2018.

Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, and Subhajit Roy. Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*, pp. 345–356, 2016.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850, 2021.

Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. Program synthesis using conflict-driven learning. *ACM SIGPLAN Notices*, 53(4):420–435, 2018.

John K Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. *ACM SIGPLAN Notices*, 50(6):229–239, 2015.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Di Huang, Rui Zhang, Xing Hu, Xishan Zhang, Pengwei Jin, Nan Li, Zidong Du, Qi Guo, and Yunji Chen. Neural program synthesis with query. In *International Conference on Learning Representations*, 2021.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. Autoshufflenet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 608–616, 2020.

Diganta Mukhopadhyay, Kumar Madhukar, and Mandayam Srivas. Permutation invariance of deep neural networks with relus. In *NASA Formal Methods Symposium*, pp. 318–337. Springer, 2022.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 4474–4484. PMLR, 2020.

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. *ACM SIGPLAN Notices*, 51(6):522–538, 2016.

Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 107–126, 2015.

ACKNOWLEDGEMENTS

We thank Alec Pauli for his work and support with the experimental setup.

A HYPERPARAMETER CORRELATIONS

Experimenting with hyperparameter grid search consisting of 100 DISPER model instances evaluated across the hyperparameter range of Table 1, we observed correlations between hyperparameters and recall, precision, and quality metrics. For the correlation measure we employed the Pearson correlation coefficient.

Relatively high values of β, γ lead to sharp deterioration in recall and were thus filtered out as outliers. Even then, β retained a slight correlation of -0.1 with recall, but had correlations of $+0.54$ and $+0.23$ with overlap and quality, respectively. γ was neutral with respect to recall, -0.38 with respect to overlap and $+0.24$ with quality. δ was -0.51 to recall, $+0.20$ to overlap and $+0.12$ to quality. n_{atoms} was $+0.12$ to recall, $+0.40$ to overlap, but -0.49 to quality.

AdamW optimizer consistently performed better over Adam, RMSProp, and Adadelta, with higher learning rates leading to significantly worse recalls (-0.45) and smaller atom sets ($+0.37$).

Higher batch sizes had the tendency to further hinder recall (-0.27) but increase overlap and quality ($+0.38$, $+0.12$).

B FULL GROUP RECONSTRUCTIONS

Group	recall	overlap	quality	Group	recall	overlap	quality
1	1.00	1.00	1.00	C_8	0.63	0.01	0.25
C_2	1.00	1.00	1.00	$C_4 \times C_2$	0.13	0.01	0.67
C_3	1.00	1.00	1.00	D_8	0.50	0.06	0.40
C_4	1.00	1.00	1.00	Q_8	1.00	0.50	0.50
$C_2 \times C_2$	0.75	0.33	0.50	$C_2 \times C_2 \times C_2$	0.25	0.02	1.00
C_5	1.00	1.00	1.00	C_9	0.11	0.01	0.33
S_3	1.00	0.50	1.00	$C_3 \times C_3$	1.00	0.50	0.50
C_6	1.00	1.00	0.33	D_{10}	0.20	0.04	1.00
C_7	1.00	0.02	0.33	C_{10}	1.00	0.33	1.00

Table 3: The results for DISPER applied to perform full reconstructions of groups of order up to 10. UNIQUE-w was used as a data form, with $\beta = 0.025, \gamma = 0.005, \delta = 0.001, n_{\text{atoms}} = 5, n_{\text{length}} = 10$.

C BASELINE MODELS

To assess the impact of architectural choices made in the design of DISPER, we evaluated it against a multi-layer fully-connected neural network, one-dimensional and two-dimensional convolutional neural networks, a recurrent neural network, a transformer encoder, and a modification of DISPER's original architecture in which the control unit is replaced by a transformer encoder appended with a linear layer.

It turned out that the success of the modification's training was very sensitive to hyperparameters and, when it trained successfully, significantly underperformed all other models considered. This is likely due to the relative scarcity of information available to the transformer through the interface of program choices made from among the candidate atoms – we believe that the interface of program choices, through which the control unit interacts with the output and the loss, acted as an information bottleneck for the transformer control unit.

The following five model architectures and configurations were considered:

- **DNN**, or a multi-layer feed-forward neural network, consisting of six ReLU-activated fully-connected layers narrowing down towards the output.
- **1D CNN**, consisting of two one-dimensional convolutional layers followed by a max-pooling layer and five fully-connected layers.

- **2D CNN**, consisting of two two-dimensional convolutional layers and five fully-connected layers.
- **RNN**, consisting of one fully-connect layer followed by a recurrent layer, followed by a collapsing fully-connected layer.
- **Transformer Encoder**, consisting of 4 layers using 8 attention heads each.

The best configurations of these models (in terms of the numbers of layers, layer widths, kernel sizes, etc.) were first identified in separate grid searches. The models were provided with the same inputs as the DISPER model (varying across dataset types). All the models accepting one-dimensional input were operating on flattened tensors (rows and columns were collapsed into one tensor dimension). Each of the baseline architectures was appended with a linear layer, whose outputs were interpreted in order as n_{length} w-by-w matrices giving the desired permutation.

The most substantial difference between the baseline models and DisPer lies in how they produce the final permutation. While DISPER learns the atomic permutations and to choose from them in parallel, the baseline models predict each of the operations constituting the permutation directly. In that manner, the repeated use of an atom by DISPER manifests itself as the same atom being chosen by the control unit multiple times for a single resulting permutation, the baseline models have to repeatedly reconstruct it in full.