nature machine intelligence

Article

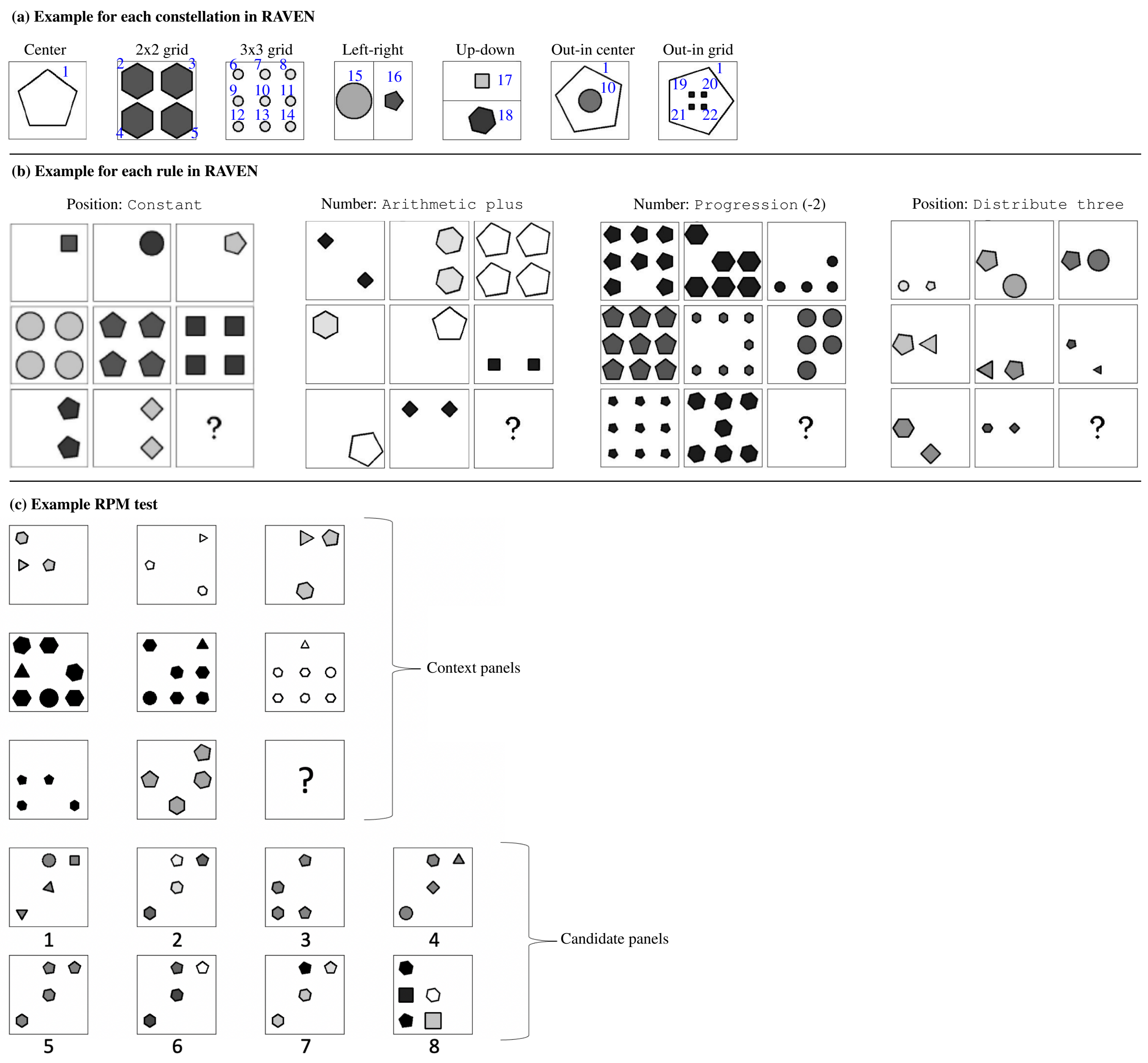
https://doi.org/10.1038/s42256-023-00630-8

A neuro-vector-symbolic architecture for solving Raven's progressive matrices

CONTENTS

SUPPLEMENTARY FIGURES

Supplementary Figure 1: Details on the RAVEN dataset.



Supplementary Figure 1: **(a)** Examples of the seven constellations in the RAVEN dataset. We enumerate 22 unique positions (in blue) across all seven constellations. Moreover, we merge overlapping positions with the same proportions across constellations, which are 1) the object in center, the outer object in out-in center, and the outer object in out-in grid (enumerated with "1"); and 2) the middle object in 3x3 grid and the inner object in out-in center (enumerated with "10"). **(b)** Examples for the four types of rules in RAVEN. In these examples, the rules are applied on the position attribute, or number attributes. A separate rule is applied per attribute, the displayed attribute and rule is just one of them. **(c)** An example of RPM test from the RAVEN dataset using the 3x3 grid constellation. There are eight context panels and eight answer panels. In this example, the number of objects stays constant per row. Moreover, the size values (small, medium, large) of the objects are distributed per row. Even though the shapes do not agree within a panel, they stay constant per row. The arithmetic minus rule is applied on the attribute color. Combining the detected rule leads to the correct answer panel 5.

SUPPLEMENTARY NOTES

Supplementary Note 1: Neural network representation learning over VSA and its generalization

In this Supplementary Note, we present further investigations into the NVSA frontend for the visual perception. In the first subsection (**a**), we explain the direct supervised training of the frontend in the presence of the visual ground-truth attribute labels using a novel additive cross-entropy loss. We also analyze the object classification accuracy and compare it with other loss functions and perceptual methods. In the next two subsections (**b** and **c**), we study the generalizability of the NVSA frontend in isolation for unseen attribute-value combinations (**b**) and unseen combinations of multiple objects (**c**).

a. Supervised training with additive cross-entropy loss and comparisons

We consider a supervised training setup in which the visual ground-truth attribute labels for all objects are provided. Therefore, the frontend can be trained standalone. We mutually train a universal NVSA frontend on all training constellations by enumerating all possible positions and merging the identical positions across constellations (see Supplementary Fig. 1). For an image panel $\mathbf{X}$, containing k objects, with k target indices $Y := \{y_i\}_{i=1}^{k}$, the trainable parameters θ of ResNet-18 are optimized to maximize the similarity between its output query $\mathbf{q} = f_\theta(\mathbf{X})$ and the bundled vector $\mathbf{w}_{y_1} \oplus ... \oplus \mathbf{w}_{y_k}$. The dictionary matrix $\mathbf{W}$ stays fixed during training. As noted, each vector in $\mathbf{W}$ is computed by multiplicative binding of the codebooks, so we call this $\mathbf{W}$ encoding multiplicative binding. Due to the similarity-preserving property of the bundling operation, maximizing the similarity between the query vector and the bundled vector is equivalent to maximizing the similarity between the query vector and each object vector:

$$\begin{aligned} \theta^* &= \underset{\theta}{\operatorname{argmax}} \operatorname{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_1} \oplus ... \oplus \mathbf{w}_{y_k}\right) & (1)\\ &\approx \underset{\theta}{\operatorname{argmax}} \operatorname{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_1}\right) + ... + \operatorname{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_k}\right). & (2) \end{aligned}$$

We propose to optimize equation (1) utilizing a novel additive cross-entropy loss, defined as

$$\mathscr{L}(\mathbf{X}, Y, \theta) := -\log \frac{e^{s_l \cdot \left(\operatorname{sim}(f_\theta(\mathbf{X}), \mathbf{w}_{y_1}) + ... + \operatorname{sim}(f_\theta(\mathbf{X}), \mathbf{w}_{y_k})\right)}}{e^{s_l \cdot \left(\operatorname{sim}(f_\theta(\mathbf{X}), \mathbf{w}_{y_1}) + ... + \operatorname{sim}(f_\theta(\mathbf{X}), \mathbf{w}_{y_k})\right)} + \sum_{i=1}^{m} e^{s_l \cdot \left(\operatorname{sim}(f_\theta(\mathbf{X}), \mathbf{w}_{y_i})\right)}}, \quad (3)$$

where s_l is an inverse softmax temperature. The loss is optimized using the batched stochastic gradient descent by exclusively updating the parameters θ while freezing $\mathbf{W}$. As the cosine similarity is bound between -1 and +1 and the softmax function embedded in the cross-entropy loss is scale sensitive, we scale the logit vector with a scalar s_l, serving as an inverse softmax temperature for improved training.

As an alternative loss function, the NVSA frontend can be trained with a randomized cross-entropy loss, which focuses on optimizing of the similarity between the query and one randomly picked object vector $\mathbf{w}_{j_i, i \in \{y_1, ..., y_k\}}$ at a time. We compute the m-dimensional logit vector $\mathbf{z} = \mathbf{W}\mathbf{q}$, pick one of the target indices at random, and compute the cross-entropy loss based on the scaled logit vector and the randomly picked target index. By repeating the optimization for multiple epochs, the randomized cross-entropy loss guides f_θ to generate a composite vector that resembles the bundling of all object vectors in the panel.

During inference, ResNet-18 generates a query vector that can be decomposed into constituent object vectors. The decomposition performs a matrix-vector multiplication between the normalized dictionary matrix $\mathbf{W}$ and the normalized query vector, $\mathbf{q}$, to obtain the cosine similarity scores $\mathbf{z}$. The similarity scores are passed through a thresholded detection function g_τ, which returns the indices of the score vector whose similarity exceeds a threshold. The optimal threshold $\tau := 0.23$ is determined by cross-validation and is identical across all constellations. Since the structure of the dictionary matrix is known, we can infer the labels for the attributes, namely position, color, size, and type, from the detected indices.

In the following, we assess the performance of the NVSA frontend by evaluating the panel accuracy when predicting the attribute values of type, size, color, and position for each panel. A correct prediction is counted only if all attribute values of all objects in a panel are predicted correctly. We compare the perception accuracy of the NVSA frontend in different training configurations with the visual perception part of PrAE[1]. The visual perception part of PrAE consists of four separate LeNet-like architectures, which predict objectiveness, type, size, and color. Since the original PrAE was trained only on the 2x2 constellation, we also train the visual perception part of PrAE on each constellation individually.

For learning the parameters of our NVSA frontend and PrAE[1], we extract the 16 panels (eight context panels and eight answer panels) and use ground-truth attribute values provided by the dataset as meta-labels. We exclusively trained and tested the models on the RAVEN training and testing sets, respectively. Moreover, we also train our NVSA frontend on a partial training set containing only 6000 training samples (instead of full 42,000 samples) by taking training samples from the individual constellations based on a share that corresponds to their number of possible locations, e.g., 3x3 grid provides 9× more training samples than the center. The trainable parameters are trained using batched stochastic gradient descent (SGD) with a weight decay of 10^{-4}1e-4 and a momentum of 0.9.

Supplementary Table I compares the panel accuracy of these different perception methods. Training the NVSA frontend on the full training set yields a highly accurate model that significantly outperforms the constellation-dependent PrAE models, where the additive cross-entropy loss results in 2.4% higher accuracy compared to the random loss (99.76% vs. 97.33%). The additive cross-entropy loss notably outperforms the randomized cross-entropy loss in the constellations with many possible locations, e.g., in the 3x3 grid (98.61% vs. 85.70%) or the out-in grid (99.95% vs. 97.30%). Moreover, when training the perception only on the partial training set (i.e., 1/7 of the full training set), the NVSA frontend accuracy is almost preserved with both the additive (99.76% vs. 97.16%) and the random cross-entropy loss (97.33% vs. 96.78%) while reducing the training set to the size of a single constellation (42,000 samples vs. 6000 samples). This showcases the sample efficiency of our approach.

Finally, we merge this instance of the NVSA frontend, which is trained on the complete training set with the additive cross-entropy loss, with the NVSA backend to solve the complete RPM tests. Tables II and III show the performance in the last row. NVSA achieves the highest accuracy of 98.5% and 99.0% on RAVEN and I-RAVEN, respectively, thanks to its accurate perception.

Supplementary Table I: Panel accuracy (%) of the visual perception methods on the RAVEN test set. The methods are trained with the visual attribute labels. Avg denotes the average accuracy over all test constellations. L-R stands for left-right, U-D for up-down, O-IC for out-in center, and O-IG for out-in grid.

Method	Training Loss	Training Constellation(s)	# Training Samples	Avg	Center	2x2 grid	3x3 grid	L-R	U-D	O-IC	O-IG
PrAE[1] perception	Rand. CEL	Individual⋆	42,000	85.27	88.65	93.56	73.95	100.0	100.0	94.23	46.52
NVSA frontend	Add. CEL	Full†	42,000	99.76	100	99.83	98.61	99.97	99.96	99.97	99.95
NVSA frontend	Rand. CEL	Full†	42,000	97.33	100	99.30	85.70	99.67	99.56	99.73	97.30
NVSA frontend	Add. CEL	Partial†	6,000	97.16	98.84	99.26	86.23	99.66	99.55	99.75	96.85
NVSA frontend	Rand. CEL	Partial†	6,000	96.78	99.83	99.18	84.95	99.63	99.52	99.74	94.57

⋆ The training constellation is identical to the testing constellation, thus seven independent models were trained and tested.
† A universal model on all seven constellations was trained and tested.

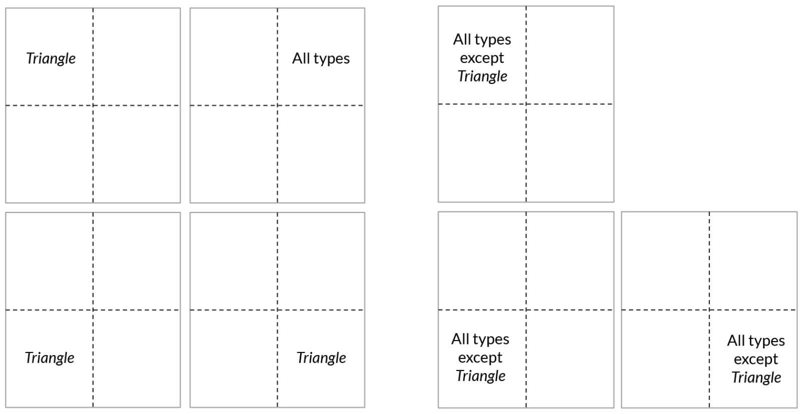
b. Generalizability of multiplicative binding to unseen combinations of attribute values

In this part, we investigate the generalizability of the NVSA frontend to unseen attribute-value combinations. To that end, we consider the single object case for the 2x2 grid constellation. After choosing a set of values for each of the four attributes (position, color, size, and type), the training and test datasets for each of the six pairs of attributes are generated as follows. For the pair of attributes $\{A_i, A_j\}_{i,j\in\{1,..,4\}}$ and their associated set of values V_i and V_j, an object is considered during training if at least one of its values for A_i or A_j is in the prespecified sets V_i and V_j. The test set only contains objects that do not satisfy the condition for A_i nor A_j.

For instance, when we choose the first quadrant for position and the triangle for type as the pair of attribute-value of interest, the training set contains panels with single objects of all types placed on the first quadrant or triangles located on the remaining three quadrants. In this case, the test set is composed of single objects of all types except triangles placed on all quadrants, excluding the first one. We note that in this setting, the considered objects can have any value for the rest of attributes that is color and size. The datasets for the example above are depicted in Supplementary Fig. 2. Supplementary Table II lists the training and testing attribute combinations for all six pairs of attributes.

The second type of generalization experiment involves determining two sets of values per attribute. For attribute A_i, the two sets are denoted $V_{i,1}$ and $V_{j,2}$. Accordingly, for the pair of attributes $\{A_i, A_j\}_{i,j\in\{1,..,4\}}$ and their respective four sets of values $V_{i,1}$, $V_{i,2}$, $V_{j,1}$, and $V_{j,2}$, the training set comprises objects with values for the two attributes in $V_{i,1}$ and $V_{j,1}$, or in $V_{i,2}$ and $V_{j,2}$. The test set is the remaining data points in the complement of the training set. This partitioning is inspired by the CLEVR dataset[2] for compositional generalization experiments.

In both investigated generalizability settings, the NVSA frontend based on the multiplicative binding cannot provide the correct predictions for the test sets, resulting in 0% test accuracy for all six attribute pairs (see the 4th column of Supplementary Table II). According to these results, this frontend instance does not show any sign of attribute-value generalizability. In fact, the frontend's inability to generalize is an inherent property of the multiplicative binding of quasi-orthogonal vectors; each entry of the cosine similarity scores vector **z** corresponds to the similarity value with the embedding vector of a given combination of the considered four attributes. Through the additive cross-entropy loss, the model learns to maximize the entry corresponding to the target and minimize the rest. In the context of the generalization experiments, the set of components of the vector **z**


Triangle
All types
Triangle
Triangle
All types except Triangle
All types except Triangle
All types except Triangle

(a) Training set (b) Test set

Supplementary Figure 2: The types of panels considered in the training set (**a**) and test set (**b**) for the first type of generalization experiments in the following pair of attribute-value: the first quadrant for position, and triangle for type.

corresponding to the training set and that of the test set are mutually exclusive. Therefore, when the dictionary matrix **W** contains quasi-orthogonal vectors, the model cannot be expected to perform accurate predictions for any unseen combination of the attributes.

After identifying this limitation in the attribute-value generalizability of the multiplicative binding encoding, we enhance this encoding by the addition (bundling) operation to add key-value bound pairs. In this enhanced multiplicative-additive encoding, we describe an object **w** with attribute values a, b, c, d for type, size, color, and position as

$$\mathbf{w} = \left(\mathbf{r}_{\text{type}} \odot \mathbf{t}_a\right) \oplus \left(\mathbf{r}_{\text{size}} \odot \mathbf{s}_b\right) \oplus \left(\mathbf{r}_{\text{color}} \odot \mathbf{c}_c\right) \oplus \left(\mathbf{r}_{\text{pos}} \odot \mathbf{l}_d\right), \tag{4}$$

where $\mathbf{r}_i \in \{-1,+1\}^d$ are randomly initialized key vectors. The key vectors are bound with the corresponding value vectors ($\mathbf{t}_a$, $\mathbf{s}_b$, $\mathbf{c}_c$, and $\mathbf{l}_d$), yielding key-value pairs which are added (bundled) to represent all attributes of the object. The multiplicative-additive encoding allows the extraction of each attribute's value individually by unbinding the object representation (**w**) with the key vector; the knowledge of other attribute values is not required. Hence, this encoding explicitly disentangles the attribute values. This allows us to treat each attribute individually and, more importantly, formulate the attribute recognition as a regression problem where a relationship between values exists. Concretely, we formulate the recognition of the color and the size as a regression problem using hyperspherical prototypes[3]. Instead of dictating the target vector of every attribute value, hyperspherical regression only defines the target vectors of the minimum and maximum values. The intermediate values are uniformly distributed on the hypersphere in terms of cosine similarities. For example, the target vectors for the attribute color with 10 values are $\mathbf{c}_1 = -\mathbf{x}$ and $\mathbf{c}_{10} = \mathbf{x}$, where $\mathbf{x}$ is a randomly initialized vector. The representation of an intermediate color value ($\mathbf{c}_i$) should then have a cosine similarity of

$$\cos(\mathbf{c}_{10}, \mathbf{c}_i) = 2 \cdot \frac{i-1}{m_c - 1} - 1, \tag{5}$$

where $m_c = 10$ is the number of possible color values. Finally, the visual perception module is trained by optimizing the mean-squared error for the hyperspherical prototypes (color and size) and the categorical cross-entropy loss for the position and type.

Supplementary Table II compares the generalization capabilities of the NVSA frontend when using different encodings: the multiplicative versus the multiplicative-additive. Indeed, the multiplicative-additive encoding significantly improves the generalization in four attribute pairs compared to the pure multiplicative encoding: position-color (34.8%), position-size (15.1%), color-size (29.3%), and color-type (72%). There are still two attribute pairs of position-type and size-type that show 0% generalization, which could be due to the spatial structure of the CNN's filters.

c. Generalizability of multiplicative binding to unseen combinations of multiple objects

In the previous subsection, we observe that the NVSA frontend using the encoding of multiplicative-additive with hyperspherical prototypes can generalize to some unseen combinations of the attribute values in a single object, while the encoding with the multiplicative binding of the quasi-orthogonal vectors cannot. Here, we further evaluate whether the multiplicative

Supplementary Table II: Accuracy of attribute-value generalization (%) of the NVSA frontend in the 2x2 grid constellation containing k=1 object. The training and test sets are chosen such that the attribute-value sets are disjoint. The NVSA frontend is trained with the ground-truth attribute labels by optimizing the loss in equation (3) with SGD. The NVSA frontend uses two different encodings: multiplicative binding of quasi-orthogonal vectors, and multiplicative-additive of hyperspherical.

	Training combinations	Testing combinations	Multiplicative + quasi-orthogonal	Multiplicative-additive + hyperspherical
Position-color	Position $\in \{0, 3\}$ OR color $\in \{0, 3, 6, 8\}$	Position $\notin \{0, 3\}$ AND color $\notin \{0, 3, 6, 8\}$	0.0	34.8
Position-type	Position $\in \{0, 3\}$ OR type $\in \{0, 2\}$	Position $\notin \{0, 3\}$ AND type $\notin \{0, 2\}$	0.0	0.0
Position-size	Position $\in \{0, 3\}$ OR size $\in \{1, 5\}$	Position $\notin \{0, 3\}$ AND size $\notin \{1, 5\}$	0.0	15.1
Color-size	Color $\in \{0, 3, 6, 8\}$ OR size $\in \{1, 5\}$	Color $\notin \{0, 3, 6, 8\}$ AND size $\notin \{1, 5\}$	0.0	29.3
Color-type	Color $\in \{0, 3, 6, 8\}$ OR type $\in \{0, 2\}$	Color $\notin \{0, 3, 6, 8\}$ AND type $\notin \{0, 2\}$	0.0	72.0
Size-type	Size $\in \{1, 5\}$ OR type $\in \{0, 2\}$	Size $\notin \{1, 5\}$ AND type $\notin \{0, 2\}$	0.0	0.0

encoding can generalize to unseen combinations of multiple objects. We train the NVSA frontend (hereafter, we simply omit the repetitive multiplicative encoding term) in the 2x2 grid constellation where the training set contains as a basis all possible panels with exactly one object, which are 9600 panels when considering that we have 10 color, 6 size, 5 type, 8 angle, and 4 position attribute values. We provide two training settings $\mathrm{k}_{\mathrm{train}} \in \{1,2\}$, where $\mathrm{k}_{\mathrm{train}}$ is the number of available objects in the panel. In the training setting $\mathrm{k}_{\mathrm{train}} = 1$, we only train the NVSA frontend using the basis training set with a single object, whereas in the $\mathrm{k}_{\mathrm{train}} = 2$ setting we have augmented the basis training set by another 9,600 panels containing 2 objects. The validation sets are always constructed in the same way as the training sets. In the testing, we consider the settings $\mathrm{k}_{\mathrm{test}} \in \{2,3,4\}$, where in each setting, the trained models are tested on 28,800 panels containing a fixed number of $\mathrm{k}_{\mathrm{test}}$ objects in the panel. See the first two rows in Supplementary Table III.

The model parameters are trained using SGD with a weight decay of 1e-4 and a momentum of 0.9. The batchsize was set to 256, and we used the scaling factor $s_l = 1$. Furthermore, we set the learning rate to 0.1 and decay by factor of 10 every 30 epochs. Moreover, the number of epochs is scaled such that all trained models have approximately the same number of updates. The optimal threshold τ is determined by cross-validation, where the selection criteria is $\tau = \mathrm{argmax}_{\hat{\tau}}\, v(\hat{\tau}) - 4\hat{\tau}$, where $v(\tau)$ is the accuracy on the validation set using threshold τ. We included the regularization on the magnitude of τ since we generally predicted too few objects when $\mathrm{k}_{\mathrm{test}}$ was large.

Next, we construct a similar experiment in the 3x3 grid constellation, in which the basis of the training set contains all 21,600 possible single-object panels. Compared to the 2x2 grid, there are 9 position attributes instead of 4. We consider the training settings $\mathrm{k}_{\mathrm{train}} \in \{1,2,3,4\}$, where in $\mathrm{k}_{\mathrm{train}} = 1$ we only use the basis training set to train the NVSA frontend. In the settings where $\mathrm{k}_{\mathrm{train}} \geqslant 2$, we augment the training set which is used in the setting $\mathrm{k}_{\mathrm{train}} - 1$ by 21,600 panels containing exactly $\mathrm{k}_{\mathrm{train}}$ objects. The validation sets are always constructed in the same way as the training sets in each setting. In the testing, we consider the settings $\mathrm{k}_{\mathrm{test}} \in \{2,3,4,5,6,7,8,9\}$, where in each setting, the trained models are tested on 64,800 panels containing a fixed number of $\mathrm{k}_{\mathrm{test}}$ objects in the panel. The training hyperparameters are chosen as in the above experiment, except that the optimal threshold τ is determined using $\tau = \mathrm{argmax}_{\hat{\tau}}\, v(\hat{\tau}) - 9\hat{\tau}$ as our selection criteria. Note that in both experiments, we omit to test the single object setting because every possible single object panel is already contained in the corresponding training set. The results of the two experiment sets are summarized in Supplementary Table III.

In the 2x2 grid constellation, we observe that after training in the $\mathrm{k}_{\mathrm{train}} = 1$ setting, the NVSA frontend is already able to correctly predict the majority of the panels in the $\mathrm{k}_{\mathrm{test}} \in \{2,3\}$ settings, where it achieves 92.4% and 68.1% without even having seen an instance of multiple object panel in training. Nevertheless, there is a significant accuracy drop in the $\mathrm{k}_{\mathrm{test}} = 4$, achieving 19.3%. However, training with the 2 object combinations (i.e., $\mathrm{k}_{\mathrm{train}} = 2$) achieves an average panel accuracy of 97.1% in all testing settings. Considering the out-of-distribution (OOD) testing cases, i.e., $\mathrm{k}_{\mathrm{test}} \in \{3,4\}$, an average panel accuracy of 97.3% is achieved. This indicates that training with the simple cases of up to 2 objects in the panels is enough to generalize to panels that contain up to 4 objects.

In the 3x3 grid constellation, we observe similar trends in the $\mathrm{k}_{\mathrm{train}} = 1$ setting, where it correctly predicts 70.6% of the panels containing 2 objects. However, in the testing settings $\mathrm{k}_{\mathrm{test}} \in \{5,6,7,8,9\}$ the model is overwhelmed by the presence of too many objects at the same time. In $\mathrm{k}_{\mathrm{train}} = 2$, it is able to obtain non-zero panel accuracy in all testing settings except the most complex one ($k_{test} = 9$). This indicates that the model is able to generalize in more complex panels with up to 8 objects after having encountered the panels with at most 2 objects. We observe that increasing the number of objects during training results in a higher OOD average panel accuracy, where the accuracy is monotonically improving with respect to k_{train}. Note that the OOD average is calculated over a more complex set, when we increase k_{train}. Finally, the NVSA frontend achieves an average panel accuracy of 86.3% after training only on the panels which contain less than half of the maximal number of objects allowed in the 3x3 grid constellation.

Supplementary Table III: The NVSA frontend using multiplicative binding and its generalization to a growing number of unseen objects in the RAVEN panel. The frontend is trained with a fixed number of objects k_{train} ranging from 1 to 2 in the 2x2 constellation, and then the test panel accuracy (%) is reported for an unseen number of object combinations (k_{test}) ranging from 2 up to 4 objects. Similar experiments are done in the 3x3 constellation where $k_{train} \in \{1, 2, 3, 4\}$ and $k_{test} \in \{2, 3, 4, ..., 9\}$. Avg denotes the average accuracy over all testing samples and OOD Avg denotes the average accuracy on testing samples with more than k_{train} objects.

Training Constellation	# Training Samples	# Epochs	k_{train}	k_{test}								Avg	OOD Avg
				2	3	4	5	6	7	8	9		
2x2	9600	200	1	92.4	68.1	19.3	-	-	-	-	-	59.9	59.9
2x2	19,200	100	2	96.6	97.3	97.3	-	-	-	-	-	97.1	97.3
3x3	21,600	400	1	77.6	30.1	2.5	0.0	0.0	0.0	0.0	0.0	13.8	13.8
3x3	43,200	200	2	89.2	83.4	60.9	30.7	10.9	2.8	0.6	0.0	34.8	27.0
3x3	64,800	133	3	92.6	92.6	90.7	82.4	67.3	49.1	32.7	17.3	65.6	56.5
3x3	86,400	100	4	89.3	89.9	90.9	91.8	91.4	88.4	81.2	67.8	86.3	84.0

d. Resolution issues in the RAVEN dataset

In addition to its high perception accuracy, the NVSA frontend offers better transparency, allowing us to discover issues in the generative process in the RAVEN dataset. Some objects in the inner part of the out-in grid constellation have a different size attribute value but the same image representation. The problem occurs in cases where the size attribute differs by one value; hence, it can be attributed to an insufficient image resolution. This generation problem only concerns objects of type square and is observed in 42.15% of the panels. For validating the perception accuracy, we solve this issue by merging classes with different sizes but same image representation.

Supplementary Note 2: Visual Analogies

In this Supplementary Note, we demonstrate another use case in which the appropriate perceptual representations in the NVSA frontend can be used directly to solve higher-level reasoning tasks. Specifically, we show that the predicted perceptual representations at the output of ResNet-18 can be directly manipulated by the binding operations to solve visual analogy tasks ($A : B :: \alpha : \beta$). In the studied task, we consider a source domain that shares one relationship, or multiple relationships, between its two sets of objects ($A : B$), and a target domain that shares the same relationship(s) between its object sets ($\alpha : \beta$). Binding the neural network representations obtained from the source domain allows to capture the relationship(s) solely from a single example, which can be applied to novel circumstances in the target domain by another application of the binding operation.

We generate a new RAVEN-like test dataset, in which a visual analogy problem consists of four panels arranged in a 2×2 matrix with a missing panel in the bottom right, as shown in Supplementary Fig. 3a. The first row constitutes the source domain where there is at least one relationship between the two panels (indicated by the blue arrow), and the second row constitutes the target domain that should establish the same relationship between one of its panels and the missing one. We demonstrate how the relationship can be captured from a single example in the source domain and how it can be applied beyond the example from which it was learned (i.e., in the novel circumstances of the target domain). This can be done by solely applying consecutive binding operations at the vector outputs generated from the neural network: the first binding operation captures the relationship in the source domain, and the second one applies it to the target domain.

a. One-to-one relationship

We describe how the relationship can be captured from the source domain and how it can be applied (i.e., transferred) to the target domain. We explain it using the visual analogy example shown in Supplementary Fig. 3a. We name this analogy problem `one_to_one` because there is only one relationship among the two objects in the source domain. Using our NVSA frontend, ResNet-18 generates the VSA representations of $\mathbf{q}_A$, $\mathbf{q}_B$, and $\mathbf{q}_\alpha$ for the objects in panels A, B, and α. By manipulating these VSA representations, we aim to generate the VSA representation of the solution panel β.

To better explain the analogy, let us refer to the ground-truth VSA representations of the object in the panels. We only refer to them for the sake of clarification; note that they are not used for solving analogies. In our example shown in Supplementary Fig. 3a, we have the following ground-truth VSA object representations:

$$\mathbf{o}_A = \mathbf{l}_5 \odot \mathbf{t}_{\text{square}} \odot \mathbf{s}_2 \odot \mathbf{c}_3 \tag{6}$$

$$\mathbf{o}_B = \mathbf{l}_5 \odot \mathbf{t}_{\text{pentagon}} \odot \mathbf{s}_5 \odot \mathbf{c}_3 \tag{7}$$

$$\mathbf{o}_\alpha = \mathbf{l}_2 \odot \mathbf{t}_{\text{square}} \odot \mathbf{s}_2 \odot \mathbf{c}_3 \tag{8}$$

$$\mathbf{o}_\beta = \mathbf{l}_2 \odot \mathbf{t}_{\text{pentagon}} \odot \mathbf{s}_5 \odot \mathbf{c}_3 \tag{9}$$

In this example, $\mathbf{o}_A$ is related to $\mathbf{o}_B$ by changing its type from square to pentagon and increasing its size from 2 to 5. Using the binding operation between the corresponding perceptual representations generated by ResNet-18 ($\mathbf{q}_A$ and $\mathbf{q}_B$) allows capturing this relationship in a VSA representation ($\mathbf{r}_{A:B}$) as a high-dimensional vector via:

$$\mathbf{r}_{A:B} = \mathbf{q}_A \odot \mathbf{q}_B \approx \mathbf{o}_A \odot \mathbf{o}_B = \mathbf{t}_{\text{square}} \odot \mathbf{t}_{\text{pentagon}} \odot \mathbf{s}_2 \odot \mathbf{s}_5. \tag{10}$$

As shown, the resulting relationship vector $\mathbf{r}_{A:B}$ approximately expresses the binding between four attribute vectors that are *actively* involved in the relationship. In fact, $\mathbf{r}_{A:B}$ transparently describes that $\mathbf{t}_{\text{square}}$ should be mapped to $\mathbf{t}_{\text{pentagon}}$, and $\mathbf{s}_2$ should be mapped to $\mathbf{s}_5$. This relationship vector provides an explanation as to how the source objects can inductively be mapped to the target objects. Therefore the relationship can be readily applied beyond the example from which it is learned. In order to apply the relationship $\mathbf{r}_{A:B}$ in the target domain $(\alpha : \beta)$, we bind the relationship vector with the object vector $\mathbf{q}_\alpha$, resulting in:

$$\hat{\mathbf{o}}_\beta = \mathbf{r}_{A:B} \odot \mathbf{q}_\alpha \approx \tag{11}$$

$$\mathbf{r}_{A:B} \odot \mathbf{o}_\alpha = \tag{12}$$

$$(\mathbf{t}_{\text{square}} \odot \mathbf{t}_{\text{pentagon}} \odot \mathbf{s}_2 \odot \mathbf{s}_5) \odot (\mathbf{l}_2 \odot \mathbf{t}_{\text{square}} \odot \mathbf{s}_2 \odot \mathbf{c}_3) = \tag{13}$$

$$\mathbf{l}_2 \odot \mathbf{t}_{\text{pentagon}} \odot \mathbf{s}_5 \odot \mathbf{c}_3 = \mathbf{o}_\beta. \tag{14}$$

We observe that the VSA-generated object ($\hat{\mathbf{o}}_\beta$) matches the target object ($\mathbf{o}_\beta$). In short, the final answer generation consists of binding both panels from the source domain and the first panel from the target domain, i.e., $\hat{\mathbf{o}}_\beta = \mathbf{q}_A \odot \mathbf{q}_B \odot \mathbf{q}_\alpha$.

We generated our `one_to_one` problems by using the 2x2 grid constellation of the RAVEN dataset, where we used the rules `constant` and `distribute_two` in the generation process. The rules are applied to the position, type, size, and color. The `constant` rule fixes the value of an attribute in a row, whereas in the `distribute_two` rule, we sample two valid and distinct values and assign the first value to the object in panels A and α and the second value to panels B and β. The rule of each attribute is selected individually, with the constraint that there is at least one attribute with the `constant` rule. The attribute values of the object in panel α are only allowed to differ if the rule on the attribute is `constant`. We generated 6,000 `one_to_one` problems, where these problems can be further divided into sets of size 2,000 that contain the `distribute_two` rule once, twice, and three times.

For evaluation, we trained the NVSA frontend for 100 epochs with a batchsize of 256 and scaling factor $s_l = 1$ on the panels from the standard RAVEN training set in the 2x2 grid constellation. We have used a learning rate of 0.1, which we have decreased by a factor of 10 every 30 epochs. We consider the analogy to be successfully solved when the ground-truth VSA representation of the object in panel β (i.e., $\mathbf{o}_\beta$) has the highest cosine similarity with our generated answer $\hat{\mathbf{o}}_\beta$. This has been done by an associative memory cleanup that computes the similarities between the generated vector and all the object vectors in the dictionary $\mathbf{W}$, i.e., $\mathbf{o}_\beta = \text{argmax}_{\mathbf{w}\in\mathbf{W}}\text{sim}(\hat{\mathbf{o}}_\beta, \mathbf{w})$. Using the output of our NVSA frontend, we solved the generated `one_to_one` analogies with an accuracy of 100%.

b. One to many relationship

We expand the `one_to_one` analogies to `one_to_many` analogies, where an example is shown in Supplementary Fig. 3b. The main difference compared to the `one_to_one` analogies is that the number of objects in the panels B and β can be more than one, e.g., 2, 3, or 4. This means there are relationships between the objects (depicted with different colors in Supplementary Fig. 3b).

We describe capturing the relationship set and transferring the relationship set to the target domain based on an example with a fixed number of objects (k) in the panels B and β. Since there are multiple (k) objects in the panel B, its ground-truth VSA representation is the addition (i.e., bundling) of the VSA representations of the individual objects ($\mathbf{o}_{B_1}$, ..., $\mathbf{o}_{B_k}$) present in the panel, which is expressed by

$$\mathbf{o}_B = \mathbf{o}_{B_1} \oplus \ldots \oplus \mathbf{o}_{B_k}. \tag{15}$$

Similarly, the ground-truth VSA representation of the panel β is:

$$\mathbf{o}_\beta = \mathbf{o}_{\beta_1} \oplus \ldots \oplus \mathbf{o}_{\beta_k}. \tag{16}$$

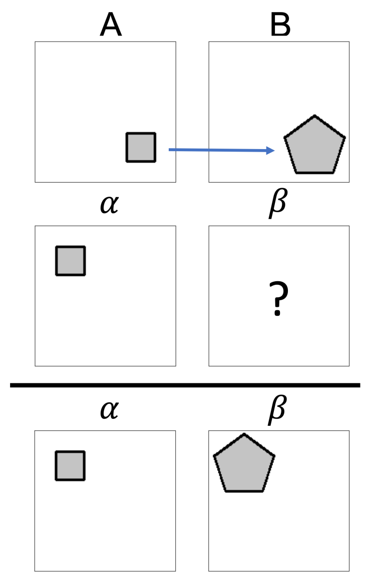
The trained ResNet-18 generates the perceptual representations for the three panels: $\mathbf{q}_A$, $\mathbf{q}_B$, $\mathbf{q}_\alpha$. Similar to Supplementary Note 2a, binding $\mathbf{q}_A$ and $\mathbf{q}_B$ captures the relationship set in $\mathbf{r}_{A:B}$. This is because, in VSA, multiplication (binding) distributes over addition (bundling). The relationship vector $\mathbf{r}_{A:B}$ is an approximation of the bundle of all `one_to_one` analogies present in the source domain, namely:

$$\mathbf{r}_{A:B} = \mathbf{q}_A \odot \mathbf{q}_b \approx \mathbf{r}^1_{A:B} \oplus \ldots \oplus \mathbf{r}^k_{A:B}, \text{where } \mathbf{r}^i_{A:B} = \mathbf{o}_A \odot \mathbf{o}_{B_i} \text{ for } i = 1, \ldots, k. \tag{17}$$

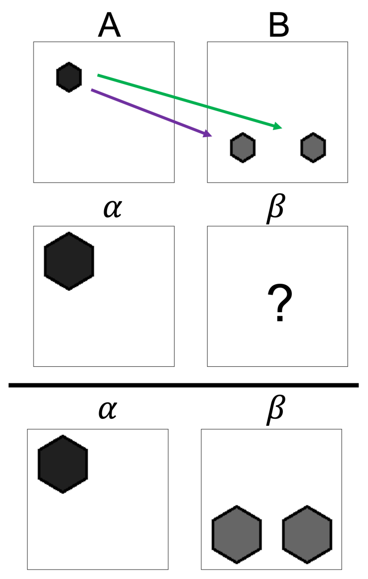
Finally, the relationship vector $\mathbf{r}_{A:B}$ is bound by the object representation in panel α to represent the target domain ($\hat{\mathbf{o}}_\beta = \mathbf{r}_{A:B} \odot \mathbf{o}_\alpha$). This single binding operation performs computation-in-superposition by applying a set of relationships simultaneously. We measure the accuracy of our analogy by calculating the cosine similarity between the ground-truth object representation $\mathbf{o}_\beta$ and our generated representation $\hat{\mathbf{o}}_\beta$. We consider the analogy to be solved when the cosine similarity between the two vectors is at least 0.99.

We generated the `one_to_many` problems similar to the `one_to_one` problems, except that multiple objects are required in panels B and β. Due to this exception, the position attribute is only allowed to have the `distribute_two` rule since the `constant` rule could not be instantiated. We generated 2000 `one_to_many` problems.

In the experiments, we used the same NVSA frontend used in Supplementary Note 2a. We achieved 100% accuracy in solving the `one_to_many` visual analogy tasks.



(a) one_to_one analogy



(b) one_to_many analogy

Supplementary Figure 3: **Example analogy in (a) `one_to_one` scenario and (b) `one_to_many` scenario.** In **(a)** we illustrate a `one_to_one` visual analogy problem. In the source domain ($A : B$) there is an underlining relationship of changing type and size of the object which is shown by the blue arrow. The same relationship should be applied to novel circumstances in target domain ($\alpha : \beta$), where for example the position of the objects is different. In **(b)** we illustrate expanded `one_to_many` visual analogy problem. In this problem, there is a set of relationships between the objects in the source domain ($A : B$). In the shown example, the relationship set consists of two `one_to_one` relationships indicated by the green an violet arrows. Both relationships change the color of the object in the same way, however they alter the position attribute to a different value.

Supplementary Note 3: Details on the NVSA backend

In this Supplementary Note, we provide a detailed description of the NVSA backend, including the VSA representation of PMFs, the rule probability computation and execution, and the selection of the rule and the final answer.

VSA representation of PMFs

For all attributes, the PMF is represented through the weighted superposition with the values in the PMF used as weights and the corresponding codewords $\mathbf{b}_k$ as basis vectors taken from the codebook $B := \{\mathbf{b}_k\}_{k=1}^{n}$:

$$\mathbf{a}^{(i,j)} := g(\mathbf{p}^{(i,j)}) = \sum_{k=1}^{n} \mathbf{p}^{(i,j)}[k] \cdot \mathbf{b}_k. \tag{18}$$

The codebook (discrete or continuous) is selected based on the underlying attribute and rule. The number of codewords n is given by the dimensionality of the PMF, which depends on the attribute. For example, the shape PMF is 5-dimensional due to the five different shapes in RAVEN; hence, the transformation requires $n = 5$ codewords. After transforming the PMF to a VSA representation, we can manipulate the PMFs in the VSA representation using the algebra provided by the vector space. This allows estimating the probability $\mathbf{u}[\texttt{rule}]$ for every `rule`. Then, the most probable rule is selected and executed in the vector space, yielding $\hat{\mathbf{a}}^{(3,3)}$. Finally, the PMF is estimated using the similarity computation with a consecutive normalization:

$$\hat{\mathbf{p}}^{(3,3)} := \text{norm}\left(\left[\text{sim}(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_1), \text{sim}(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_2), ..., \text{sim}(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_n),\right]\right). \tag{19}$$

An alternative VSA representation to the binary sparse block codes is Fourier holographic reduced representation (FHRR)[4]. In the following, we describe a similar procedure of transforming a PMF to an FHRR-based VSA representation and compare it with the binary sparse block codes. The basis vectors in FHRR are d-dimensional, complex-valued, unary vectors. Each element is a complex phasor with unit norm and an angle randomly drawn from a uniform distribution $U(-\pi, \pi)$. The dense bipolar representations are a particular case of the FHRR model where angles are restricted to $\{0, \pi\}$. The binding in FHRR is defined as the element-wise modulo-2π sum; similarly, the unbinding is the element-wise modulo-2π difference. The bundling of two or more vectors is computed via the element-wise addition with a consecutive normalization step, which sets the magnitude of each phasor to unit magnitude. The similarity of two vectors is the sum of the cosines of the differences between the corresponding angles. Binding, unbinding, and similarity computation can be done using the polar coordinates, while bundling requires the Cartesian coordinates. For a discrete attribute, we use a codebook with n unrelated basis vectors $\mathbf{b}_i \in \mathbb{C}^d$. For representing the PMF of a continuous attribute, we use a codebook with basis vectors generated by the fractional power encoding[4], where the basis vector corresponding to an attribute value v is defined by exponentiation of a randomly chosen basis vector $\mathbf{e}$ using the value as the exponent, i.e., $\mathbf{b}_v = \mathbf{e}^v$. Each PMF is represented through the normalized weighted superposition with the values in the PMF used as weights and the corresponding codewords as basis vectors:

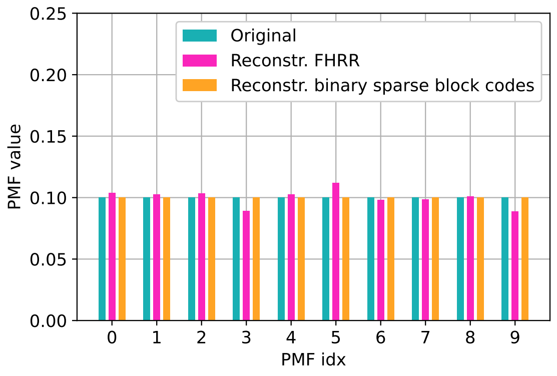
$$\mathbf{a}_{\text{FHRR}}^{(i,j)} := g_{\text{FHRR}}(\mathbf{p}^{(i,j)}) = \text{cnorm}\left(\sum_{k=1}^{n} \mathbf{p}^{(i,j)}[k] \cdot \mathbf{b}_k\right), \tag{20}$$

where cnorm$(\cdot)$ normalizes the magnitude of every phasor of a d-dimensional complex-valued vector.

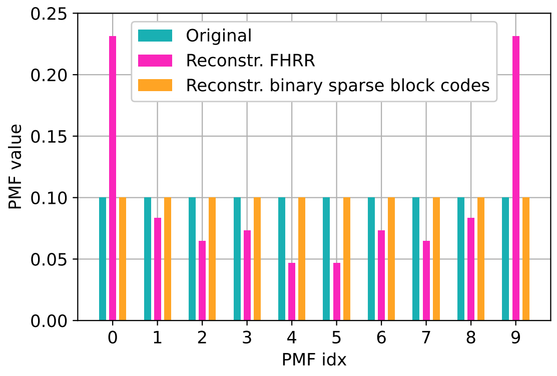
However, we observed nonidealities when mapping PMFs to the FHRR-based VSA representations. In a synthetic experiment, we mapped a uniform distribution to the VSA representation using either the binary sparse block codes (by equation (18)) or FHRR (by equation (20)), and projected them back to the PMF representation again using the associative memory search on the corresponding codebook. Supplementary Fig. 4 shows the original and the reconstructed PMFs for discrete and continuous concepts. We observe that both FHRR and binary sparse block codes can represent the discrete PMFs; however, FHRR faces issues in representing continuous PMFs. This nonideality might stem from the complex normalization step in combination with the constructive interference of the superimposed complex phasors. Thus, in this work, we use sparse binary block codes in our NVSA backend.

Rule probability computation and rule execution

In the following, we describe the probability computation and the execution for every rule.



(a) Discrete



(b) Continuous

Supplementary Figure 4: Reconstruction of uniformly distributed PMF using either FHRR or binary sparse block codes for both discrete and continuous concepts. The dimension of both VSA representations is set to d=1024.

a. Arithmetic plus and minus. The arithmetic rule computes the value of the last panel by adding (arithmetic plus) or subtracting (arithmetic minus) the attribute values of the first two panels. Since this rule relates to a continuous concept, we use a dictionary constructed by fractional power encoding. For determining the rule probability for arithmetic plus, we represent the addition of the first two panels using the binding operation

$$\mathbf{r}_i^+ = \mathbf{a}^{(i,1)} \odot \mathbf{a}^{(i,2)}, \quad i \in \{1,2\}. \tag{21}$$

If the arithmetic plus rule applies, we expect $\text{sim}(\mathbf{r}_i^+, \mathbf{a}^{(i,3)}) \gg 0$. Let us have a closer look at the similarity expression for the first row:

$$\begin{aligned}
\text{sim}(\mathbf{r}_i^+, \mathbf{a}^{(1,3)}) &= \text{sim}(\mathbf{a}^{(1,1)} \odot \mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) && (22)\\
&= \text{sim}\left(\left(\sum_{k_1=1}^{n} \mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{b}_{k_1}\right) \odot \left(\sum_{k_2=1}^{n} \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{b}_{k_2}\right), \left(\sum_{k_3=1}^{n} \mathbf{p}^{(1,3)}[k_3]\cdot \mathbf{b}_{k_3}\right)\right) && (23)\\
&= \sum_{\substack{k_1,k_2,k_3\\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{p}^{(1,3)}[k_3] + \sum_{\substack{k_1,k_2,k_3\\ s.t. k_1+k_2\neq k_3}} \text{sim}(\mathbf{b}_{k_1}\odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3})\mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{p}^{(1,3)}[k_3] && (24)\\
&= \sum_{\substack{k_1,k_2,k_3\\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{p}^{(1,3)}[k_3] + \sum_{\substack{k_1,k_2,k_3\\ s.t. k_1+k_2\neq k_3}} n_{k_1,k_2,k_3}\cdot \mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{p}^{(1,3)}[k_3] && (25)\\
&\approx \sum_{\substack{k_1,k_2,k_3\\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1]\cdot \mathbf{p}^{(1,2)}[k_2]\cdot \mathbf{p}^{(1,3)}[k_3]. && (26)
\end{aligned}$$

Equation (24) uses the linearity of the similarity and divides the sum into contributions that satisfy the arithmetic plus constraint (LHS), i.e., $\text{sim}(\mathbf{b}_{k_1}\odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3}) = 1$, and contributions which do not satisfy the constraint (RHS). For the latter, we replace the similarity $\text{sim}(\mathbf{b}_{k_1}\odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3})$ with n_{k_1,k_2,k_3}, which can be modeled as vanishing noise as the dimension d increases. The non-satisfying terms converge to zero with sufficiently large dimension d and we can approximate (25) with (26). Equation (26) sums up the products of all valid rule implementations. Indeed, this computation appears in traditional probabilistic reasoning engines such as the PrAE[1]. Supplementary Table IV shows the relation between our NVSA backend and the PrAE backend[1] for computing the probabilities for different rules. Our NVSA backend derives the rule probability based on the similarity between vectors of fixed dimension, while the PrAE computes the rule probability by marginalizing all possible rule implementations.

For interpreting the similarity in equation (22) as a probability, we limit the range of the similarity using a threshold function, which sets all similarity values below 0.05 to 0. This suppresses noise stemming from invalid contributions.

Some of the rules need to satisfy additional constraints to be valid. For the arithmetic plus, the sum of the attributes of the first two panels (k_1+k_2) has to be smaller than n. By computing $\text{sim}(\mathbf{r}_i^+, \mathbf{a}^{(i,3)})$ for $i \in \{1,2\}$, this constraint is embedded for the first two rows. For validating the arithmetic rule in the last row, we compute the constraint

$$h_a(\mathbf{a}^{(3,1)},\mathbf{a}^{(3,2)}) := \min\left(\sum_{k=1}^{n} \mathrm{sim}\left(\mathbf{a}^{(3,1)} \odot \mathbf{a}^{(3,2)}, \mathbf{b}_k\right), 1\right). \tag{27}$$

The constraint accumulates all projections of the binding of $\mathbf{a}^{(3,1)}$ and $\mathbf{a}^{(3,2)}$ (i.e., the addition) to the space spanned by B. The $\min(\cdot,1)$ guarantees the constraint to be $0 \leqslant h_a \leqslant 1$ such that it can be interpreted as a probability. If the majority of the binding falls outside of the space, i.e., the addition is larger than n, the constraint is not satisfied, and its value will be close to zero. Finally, the rule probability is determined as

$$\mathbf{u}[\texttt{arithmetic plus}] = \mathrm{sim}(\mathbf{r}_1^+,\mathbf{a}^{(1,3)}) \cdot \mathrm{sim}(\mathbf{r}_2^+,\mathbf{a}^{(2,3)}) \cdot h_a(\mathbf{a}^{(3,1)},\mathbf{a}^{(3,2)}). \tag{28}$$

If the rule is selected, it is executed by computing

$$\hat{\mathbf{a}}^{(3,3)} = \mathbf{a}^{(3,1)} \odot \mathbf{a}^{(3,2)}. \tag{29}$$

The rule arithmetic minus is implemented analogously, where the row representation is computed using the unbinding operation:

$$\mathbf{r}_i^- = \mathbf{a}^{(i,1)} \circledast \mathbf{a}^{(i,2)}, \quad i \in \{1,2\}. \tag{30}$$

b. Progression. The progression rule describes a positive or negative increment by one or two values along the panels; hence, it is a continuous concept, too. We detect and execute the progression rules with different increments and decrements individually. The RAVEN dataset applies the rules row-wise. We compute the rule probability for positive increments by computing first the unbinding between adjacent panels

$$\mathbf{d}^{+(i,j)} = \mathbf{a}^{(i,j+1)} \circledast \mathbf{a}^{(i,j)} \quad (i,j) \in \{(1,1),(1,2),(2,1),(2,2),(3,1)\} \tag{31}$$

as well as the unbinding between the left-most and right-most panel in the first two rows:

$$\mathbf{d}^{++(i,0)} = \mathbf{a}^{(i,2)} \circledast \mathbf{a}^{(i,0)} \quad i \in \{1,2\}. \tag{32}$$

If the progression rule by an increment of $n \in \{1,2\}$ is active, we expect the unbound vectors $\mathbf{d}^{+(i,j)}$ and $\mathbf{d}^{++(i,0)}$ to be similar to the basis vectors that represent the values n and $2n$ ($\mathbf{b}_n$ and $\mathbf{b}_n^2$):

$$\mathbf{u}[\texttt{progression-plus-}n] = \left(\prod_{i,j} \mathrm{sim}(\mathbf{d}^{+(i,j)},\mathbf{b}_n)\right) \cdot \left(\prod_{i\in\{1,2\}} \mathrm{sim}(\mathbf{d}^{++(i,0)},\mathbf{b}_n^2)\right) \cdot h_p(\mathbf{d}^{(1,1)}). \tag{33}$$

The last term prevents us from confusing the progression rule with the constant rule, which can be interpreted as a progression of zero, and is implemented as

$$h_p(\mathbf{d}^{+(1,1)}) := (1 - \mathrm{sim}(\mathbf{d}^{+(1,1)},\mathbf{0})). \tag{34}$$

It computes the similarity between a difference vector ($\mathbf{d}^{(1,1)}$) and the all-zero vector ($\mathbf{0}$), which is represented with a vector where each block has its non-zero element at index 0. Finally, the progression rule is executed by

$$\hat{\mathbf{a}}^{(3,3)} = \mathbf{a}^{(3,2)} \odot \mathbf{b}_n. \tag{35}$$

The implementation of the progression with decrement is analogous, where we compute the binding in reverse order, e.g.,

$$\mathbf{d}^{-(i,j)} = \mathbf{a}^{(i,j)} \circledast \mathbf{a}^{(i,j+1)} \quad (i,j) \in \{(1,1),(1,2),(2,1),(2,2),(3,1)\}. \tag{36}$$

Supplementary Table IV: Relation of rule probability computation between PrAE[1] and our NVSA backend.

Rule	PrAE	NVSA
`Constant`	$u = \sum_{r=1}^{2}\sum_{v=1}^{n}\prod_{c=1}^{3}\mathbf{p}^{(r,c)}[v] + \sum_{v=1}^{n}\prod_{c=1}^{2}\mathbf{p}^{(3,c)}[v]$	$u = \text{sim}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}) \cdot \text{sim}(\mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) \cdot \ldots \cdot \text{sim}(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)})$
`Progression-plus n`	$u = \sum_{r=1}^{2} \sum_{\substack{v_1,v_2,v_3 \\ s.t.\, v_1+n=v_2 \\ v_2+n=v_3}} \prod_{c=1}^{3}\mathbf{p}^{(r,c)}[v_c] + \sum_{\substack{v_1,v_2 \\ s.t.\, v_1+n=v_2}} \prod_{c=1}^{2}\mathbf{p}^{(3,c)}[v_c]$	$u = \left(\prod_{i,j}\text{sim}(\mathbf{d}^{+(i,j)}, \mathbf{b}_n)\right) \cdot \left(\prod_{i\in\{1,2\}} \text{sim}(\mathbf{d}^{++(i,0)}, \mathbf{b}_n^2)\right) \cdot h_p(\mathbf{d}^{(1,1)})$
`Arithmetic-plus`	$u = \sum_{r=1}^{2} \sum_{\substack{v_1,v_2,v_3 \\ s.t.\, v_1+v_2=v_3}} \prod_{c=1}^{3}\mathbf{p}^{(r,c)}[v_c]$	$u = \text{sim}(\mathbf{r}_1^{+}, \mathbf{a}^{(1,3)}) \cdot \text{sim}(\mathbf{r}_2^{+}, \mathbf{a}^{(2,3)}) \cdot h_a(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)})$
`Dist.three`	$u = \sum_{v^{(1,1)},\ldots,v^{(3,2)} \in I_{d3}} \prod_{r=1}^{2}\prod_{c=1}^{3}\mathbf{p}^{(r,c)}[v^{(r,c)}] \prod_{c=1}^{2}\mathbf{p}^{(3,c)}[v^{(3,c)}]$	$u = \text{sim}(\mathbf{c}_1, \mathbf{c}_2) \cdot \text{sim}(\mathbf{r}_1, \mathbf{r}_2) \cdot h_d(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}), \ldots, \mathbf{a}^{(2,3)})$

c. Distribute three. The distribute three rule relates to a discrete concept, hence we use fully random codewords. First, we compute the row-wise binding of the PMF-vector representation of the first two rows

$$\mathbf{r}_i = \mathbf{a}^{(i,1)} \odot \mathbf{a}^{(i,2)} \odot \mathbf{a}^{(i,3)}, \quad i \in \{1,2\}. \tag{37}$$

Similarly, we can compute the column representations by

$$\mathbf{c}_j = \mathbf{a}^{(1,j)} \odot \mathbf{a}^{(2,j)} \odot \mathbf{a}^{(3,j)}, \quad j \in \{1,2\}. \tag{38}$$

If the distribute three rule applies, we expect both $\text{sim}(\mathbf{r}_1, \mathbf{r}_2) \gg 0$ and $\text{sim}(\mathbf{c}_1, \mathbf{c}_2) \gg 0$.

Hence, the rule probability is computed by

$$\mathbf{u}[\texttt{distribute three}] = \text{sim}(\mathbf{c}_1, \mathbf{c}_2) \cdot \text{sim}(\mathbf{r}_1, \mathbf{r}_2) \cdot h_d(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}), \ldots, \mathbf{a}^{(2,3)}), \tag{39}$$

where

$$h_d(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}), \ldots, \mathbf{a}^{(2,3)}) := (1 - \text{sim}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}) \cdot (1 - \text{sim}(\mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) \cdot \ldots \cdot (1 - \text{sim}(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)}) \tag{40}$$

validates the constraint that panels are not equal within a row. If the rule is selected, it is executed by

$$\hat{\mathbf{a}}^{(3,3)} = \mathbf{r}_1 \circledast \left(\mathbf{a}^{(3,1)} \odot \mathbf{a}^{(3,2)}\right). \tag{41}$$

d. Constant. The computation of the constant rule probability involves the row-wise similarities:

$$\mathbf{u}[\texttt{constant}] = \text{sim}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}) \cdot \text{sim}(\mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) \cdot \text{sim}(\mathbf{a}^{(2,1)}, \mathbf{a}^{(2,2)}) \cdot \text{sim}(\mathbf{a}^{(2,2)}, \mathbf{a}^{(2,3)}) \cdot \text{sim}(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)}). \tag{42}$$

The execution of the constant rule requires no transformation; thus, there is no need to map the PMF to the vector space and back. Therefore, the PMF of the missing panel can be directly estimated by using one of the PMFs in the bottom row, e.g.,

$$\hat{\mathbf{p}}^{(3,3)} = \mathbf{p}^{(3,1)}. \tag{43}$$

Selection of the rule and the final answer

For each attribute, we compute the rule probability of all rules using the NVSA backend. In the RAVEN dataset, the arithmetic and progression rules on the position attribute are implemented in the binary system; thus, we compute the rule probability computation in the original PMF space for those rules. For every attribute, we select the rule with the highest rule probability and apply it to get $\hat{P}^{(3,3)} := (\hat{\mathbf{p}}_{\text{pos}}, \hat{\mathbf{p}}_{\text{num}}, \hat{\mathbf{p}}_{\text{type}}, \hat{\mathbf{p}}_{\text{size}}, \hat{\mathbf{p}}_{\text{color}})$. Finally, we compute for each candidate answer panel j, the Jensen–Shannon divergence (JSD) between each of the five probability distributions in $P^{(k)}$ and $\hat{P}^{(3,3)}$, and sum the five JSD values to obtain a score for the answer panel j. The predicted answer panel $j^{\star}$ is the one with the lowest total divergence.

Supplementary Note 4: Out-Of-Distribution generalization to unseen attribute-rule pairs

In this Supplementary Note, we evaluate the generalizability of our NVSA, including both frontend and backend, for to unseen attribute-rule pairs. More specifically, we evaluate whether our model is able to solve an unseen target attribute-rule pair (e.g., the constant rule on the type attribute) when it has been trained on the examples containing all of the attribute-rule pairs except the specific target one (e.g., the constant rule on size and color, the progression rule on all attributes, and the distribute rule on all attributes). Hence, this setting tests the out-of-distribution generalization for the attribute-rule pairs. To do so, we generate a new training and validation set containing all examples except those with the target attribute-rule pair and a test set containing examples exclusively with the target attribute-rule pair. The datasets are generated by filtering the existing splits in RAVEN and I-RAVEN. As a result, the sets contain fewer samples depending on the target attribute-rule pair; specifically, the training sets contain 2622–3437 samples, the validation sets 841–1160 samples, and the test sets 803–1117.

Supplementary Table V shows the experimental results on I-RAVEN in the L-R constellation. The results of LEN[5] and CoPINet[6] are based on experiments conducted by Wu *et al.*[7]. Our NVSA was trained end-to-end and outperformed both baselines by a large margin in all target attribute-rule pairs. Minor accuracy degradations are observed in the continuous rules (i.e., progression and arithmetic). This might point out the importance of the continuous rules being present for the NVSA to learn all attribute values properly.

Supplementary Table V: Out-Of-Distribution generalization on the unseen rule-attribute pairs of the I-RAVEN dataset in the L-R constellation. We report accuracy (%) on test set that contains exclusively examples with the target attribute-value pairs on which it has not been trained on.

	Type			Size				Color			
	Constant	Progress.	Dist.3	Constant	Progress.	Dist.3	Arithmetic	Constant	Progress.	Dist.3	Arithmetic
LEN[5]	28.0	24.0	29.4	24.4	27.9	27.6	-	25.3	25.3	22.0	-
CoPINet[6]	25.1	36.2	32.9	37.2	36.2	36.4	-	38.8	35.8	29.2	-
NVSA	100	81.8	100	100	100	100	77.8	99.9	81.7	100	80.9

Supplementary Note 5: Experiments on the PGM dataset

This section describes the application of our NVSA to the Procedurally Generated Matrices (PGM) dataset[8].

PGM Dataset

The PGM dataset provides RPM tests with two constellations, line and shape, which can simultaneously be present in a panel. The objects in the shape constellation are arranged in a 3x3 grid, each taking one out of ten gray shadings, ten sizes, and seven geometrical types (see Supplementary Table VI). Moreover, the line constellation contains six different line types, each taking one out of ten gray shadings. Both constellations can have no objects present in a panel.

Each PGM example has 1 to 4 active rules, either applied row-wise or column-wise. This contrasts the RAVEN dataset, where only a row-wise rule governs every attribute. The rules can be described as follows:

- `Progression`: The attribute value monotonically increases by a value of one in a row/column.
- `XOR`, `OR`, and `AND`: The set of attribute values in the third panel in a row/column corresponds to the logical `XOR`, `OR`, or `AND` operation of the first two panels. Let us consider an example with the attribute type in the shape constellation. The first panel contains objects with squares and triangles and the second only triangles. Consequently, the third panel would either contain only squares (`XOR`), both squares and triangles (`OR`), or only triangles (`AND`).
- `Consistent union`: The same set of attribute values appear in the three panels of every row/column (with permutations of the values in different rows/columns). This is a relaxed version of the `distribute three` rule in the RAVEN dataset since it does not require all the permutations to be distinct.

Supplementary Table VI summarizes the attribute rules of the two constellations. The PGM dataset contains 1,200,000 examples for training, 20,000 for validation, and 200,000 for testing. Moreover, the active rules are provided as meta-labels. Note that the meta-labels do not contain the orientation of the rule (i.e., row-wise or column-wise).

NVSA frontend

We start by defining the codebooks for the two constellations. The line constellation has two codebooks, $T_L := \{\mathbf{t}_i\}_{i=1}^{6}$ for type and $C_L := \{\mathbf{c}_i\}_{i=1}^{10}$ for color. Similarly, the codebooks for the shape constellation are $T_S := \{\mathbf{t}_i\}_{i=1}^{7}$, $S_S := \{\mathbf{s}_i\}_{i=1}^{10}$, $C_S := \{\mathbf{c}_i^S\}_{i=1}^{10}$, and $L_S := \{\mathbf{l}_i\}_{i=1}^{9}$ representing the type, size, color, and position of a single object. We build the dictionaries $\mathbf{W}_L \in \{-1,+1\}^{m_L \times d}$ and $\mathbf{W}_S \in \{-1,+1\}^{m_S \times d}$ by binding the vectors from the codebooks of all possible combinations for line and shape. This yields m_S=6300 combinations for the shape and m_L=60 combinations for the line.

In our earlier NVSA frontend using one ResNet-18, we identified a limitation in the end-to-end training with multiple attributes, where in most cases, only one attribute was learned, and the others remained at random chance. The limitation might stem from the larger number of attribute combinations in PGM: the shapes in PGM have $> 2\times$ more attribute combinations than the largest 3x3 grid constellation in RAVEN (6300 vs. 2700). To simplify the end-to-end training, we train four ResNet-18 models, each focusing on two attributes: one for the line constellation (type-color attributes) and three for the shape constellation (type-position, size-position, color-position). We intentionally included the position attribute for every attribute since it influences the scene probability computation of the other attributes (e.g., see equation (17) in Methods). Moreover, we use a larger dimension $d = 1024$ for better performance.

Probabilistic scene representation

For every panel, we compute a PMF for each object (e.g., $\mathbf{v}^{(k)}_{\text{exist}}$, $\mathbf{v}^{(k)}_{\text{type}}$, $\mathbf{v}^{(k)}_{\text{size}}$, $\mathbf{v}^{(k)}_{\text{color}}$ for object k in the shape constellation) using the marginalization with consecutive softmax approach (see equation (12)–(14) in Methods). We then derive the PMFs representing the attributes of the panel. Here, most attribute-rule pairs use the same computation of the panel PMF as in RAVEN. The exceptions are the logical rules (`XOR`, `OR`, and `AND`) on color, size, and type, where we need to describe every attribute value combination separately. More specifically, we describe the occupancy with the set of occupied values I_j; e.g., $I_3 = \{1,2\}$ represents a scene with at least one object with attribute value 1 and at least one with attribute value 2. The probability that a panel contains the attributes a with values in I_j is determined by

$$\mathbf{p}'_a[j] = \prod_{k \in I_j} \min\left(\sum_{l=1}^{n} v^{(l)}_{\text{exist}}[0] v^{(l)}_a[k], 1 \right), \tag{44}$$

Supplementary Table VI: Summary of attributes and rules in the PGM dataset.

Constellation	Attribute name	Number of attribute values	Rules
Shapes	Color	10	`Progression`, `XOR`, `OR`, `AND`, `consistent union`
	Size	10	`Progression`, `XOR`, `OR`, `AND`, `consistent union`
	Number	10	`Progression`, `consistent union`
	Position	9	`XOR`, `OR`, `AND`
	Type	7	`XOR`, `OR`, `AND`, `consistent union`
Line	Color	10	`Progression`, `XOR`, `OR`, `AND`, `consistent union`
	Type	6	`XOR`, `OR`, `AND`, `consistent union`

where n is the number of positions in the scene (n=6 for line and n=9 for shape), $v_{\text{exist}}^{(l)}$ the probability that an object exists at position l, and $v_a^{(l)}[k]$ the probability that the object at position l has attribute a with value k. We limit the set I_j to contain at most four different values to keep the compute and memory demands low. Overall, we get a scene representation for the line, $P_L := \{\mathbf{p}_{\text{pos}}, \mathbf{p}_{\text{color}}, \mathbf{p}'_{\text{color}}\}$, and for the shape constellation, $P_S := \{\mathbf{p}_{\text{pos}}, \mathbf{p}_{\text{num}}, \mathbf{p}_{\text{type}}, \mathbf{p}'_{\text{type}}, \mathbf{p}_{\text{size}}, \mathbf{p}'_{\text{size}}, \mathbf{p}_{\text{color}}, \mathbf{p}'_{\text{color}}\}$. Note that the attributes color, type, and size now have two scene representations: the standard PMF $\mathbf{p}_a$ (see equation (17) in Methods) and the novel extended $\mathbf{p}'_a$ (see equation (44)). As opposed to the RAVEN dataset, an inconsistency state is not required for the PGM dataset.

NVSA backend

Here, we describe the rule probability computation and execution for the PGM dataset. Similar to NVSA's application to the RAVEN dataset, the `progression` rule is implemented with VSA-enhanced vector operations. Moreover, the `consistent union` rule implementation benefits from the VSA-enabled computation-in-superposition, similar to the `distribute three` rule in RAVEN. The logical rules (`XOR`, `OR`, and `AND`) are simple logical operations that can be implemented more efficiently in the original low-dimensional PMF space. We compute the rule probability along the rows and columns and execute it accordingly. In the following, we describe the row-wise implementation; the column-wise implementation is done by feeding the transposed context matrix to the NVSA backend.

a. Progression. The progression rule's probability computation and execution are implemented as described in equation (33) and equation (35), respectively, where only the increment by one value is detected and executed in this case.

b. Consistent union. The consistent union rule relates to a discrete concept; hence, we use random codewords. First, we compute the row-wise binding of the PMF-vector representation of the first two rows:

$$\mathbf{r}_i = \mathbf{a}^{(i,1)} \odot \mathbf{a}^{(i,2)} \odot \mathbf{a}^{(i,3)}, \quad i \in \{1,2\}. \tag{45}$$

The rule probability is computed by

$$\mathbf{u}[\texttt{consistent union}] = \text{sim}(\mathbf{r}_1, \mathbf{r}_2) \cdot h_{c.u.}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}, ..., \mathbf{a}^{(2,3)}), \tag{46}$$

where

$$h_{c.u.}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}, ..., \mathbf{a}^{(2,3)}) := \left(\prod_{i \in \{1,2\}} \prod_{j \in \{1,2\}} (1 - \text{sim}(\mathbf{a}^{(i,j)}, \mathbf{a}^{(i,j+1)}) \right) \cdot (1 - \text{sim}(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)}) \tag{47}$$

validates the constraint that panels are not equal within a row. If the rule is selected, it is executed by

$$\hat{\mathbf{a}}^{(3,3)} = \mathbf{r}_1 \circledast \left(\mathbf{a}^{(3,1)} \odot \mathbf{a}^{(3,2)} \right). \tag{48}$$

c. XOR, OR, and AND. The rule probability of the logical rules is computed in the original PMF space by summing up all possible rule implementations. For example, the rule probability for the `XOR` rule is determined by:

$$u[\texttt{XOR}] = \left(\sum_{r=1}^{2} \sum_{\substack{v_1, v_2, v_3 \\ s.t.\, \mathbb{1}_{XOR}(v_1, v_2, v_3)}} \prod_{c=1}^{3} \mathbf{p}^{(r,c)}[v_c] \right) + \sum_{v_1, v_2} \prod_{c=1}^{2} \mathbf{p}^{(3,c)}[v_c], \tag{49}$$

Supplementary Table VII: End-to-end accuracy (%) on the neutral split of the PGM test set. The upper part shows the results reported in the literature. In the lower part, we report the average accuracy $\pm$ the standard deviation over five runs with different seeds for our NVSA and the reproduced baselines.

Method	Accuracy
CNN+MLP[8]	33.0
LSTM[8]	35.8
Resnet-50[8]	42.0
Wild-ResNet[8]	48.0
CoPINet[6]	56.4
WReN[8]	62.8
LEN[5]	88.9
SCL[7]	88.9
MRNet[9]	93.4
PrAE[1]	N/A[⋆]
LEN[5]	N/A[†]
SCL[7]	N/A[†]
MRNet[9]	68.34±4.73
NVSA (end-to-end tr.)	68.30±4.93

[*] PrAE only applied to RAVEN.
[†] No code for PGM available on the code repositories.

where $1_{XOR}(v_1, v_2, v_3)$ indicates the correctness of the `XOR` rule within a row/column given the indices v_1, v_2, and v_3. Similarly, the rule probability for `OR` and `AND` are determined with the corresponding indication function. For executing the rule, we marginalize all combinations in the last row, i.e.,

$$\hat{\mathbf{p}}^{(3,3)}[v] = \sum_{\substack{v_1,v_2 \\ s.t.\ 1_{XOR}(v_1,v_2,v)}} \prod_{c=1}^{2} \mathbf{p}^{(3,c)}[v_c]. \tag{50}$$

Selection of the final answer

For every attribute, we execute the rule with the highest probability yielding the estimated PMFs for the line and shape constellation: $\hat{P} := \{\hat{P}_L, \hat{P}_S\}$. Finally, we compute the score of every candidate answer panel i by summing up the JSD of the individual attributes:

$$s(P^{(i)}, \hat{P}) = -\sum_a w_a \cdot \mathrm{JSD}(\mathbf{p}_a^{(i)}, \hat{\mathbf{p}}_a), \tag{51}$$

where w_a weights the contribution of the attribute a. In the RAVEN dataset, all the attributes equally contribute to the final score (i.e., $w_a = 1, \forall a$). In contrast, in the PGM dataset, only 1 to 4 attributes have an active rule. Hence, we use a learnable small-sized multi-layer perceptron (MLP) which predicts the set of active attributes given the rule probabilities and the JSD errors. More concretely, the MLP takes the concatenation of all rule probabilities, their maximizing values, and the JSD errors as input and predicts the values w_a. The MLP contains one hidden layer with a dimension of 75 and a ReLU activation, and a sigmoid activation at the output. The MLP is learned by optimizing the binary cross-entropy loss between the predicted attribute weights and the ground-truth values, which are derived from the auxiliary attribute rules.

Training details

We train each perception frontend separately with training data containing examples with the corresponding attribute rules. For example, for training the frontend corresponding to the position and type in the shape constellation, we filter the training set such that it contains either rules with attribute position or type. For improved training, we restrict the training samples to have only one rule. This yields around 314,000 examples for training the type-color frontend for the line constellation and 146,000 examples for each shape constellation frontend (position-color, position-type, and position-size). The models are trained and validated on a Linux machine using an NVIDIA Tesla A100 GPU.

Similar to our approach on RAVEN, we optimize the REINFORCE loss, which is augmented with an auxiliary loss (see equation (19) in Methods), where only attributes with active rules contribute to the loss (provided by meta-labels). We train the shape-related frontends for 45 epochs and the line-related one for 25 epochs using the Adam optimizer with weight decay 10^{-4}, a constant learning rate of 9.5×10^{-5}, and batchsize of 16.

After the NVSA training, the attribute selection MLP is learned on a randomly selected subset of the complete training data (i.e., no rule-based filtering), which turned out to be sufficient to the large dataset size. We train the MLP for 50 epochs using a batchsize of 64 and a learning rate of 0.01, where we randomly select only 128 batches in each epoch (0.68% of the entire dataset).

Experimental results

Supplementary Table VII compares the end-to-end accuracy of our NVSA with various baselines. The upper part of the table shows the accuracy reported in the literature, where the highly accurate methods are LEN[5] (88.9%), SCL[7] (88.9%), and MRNet[9] (93.4%). The lower part of the table compares the accuracy of the reproduced methods with our NVSA. PrAE[1] was only developed for the RAVEN dataset; hence, it could not be easily applied to the PGM dataset. Similarly, the open-sourced code of LEN[5] and SCL[7] can only be applied to RAVEN, even though PGM results are reported in the corresponding works. Finally, MRNet[9], the current state-of-the-art method on PGM, provides code for this dataset. However, training the architectures from scratch with randomly initialized weights with different seeds yielded significantly lower accuracy than the one reported in their paper (68.3% vs. 93.4%), despite optimizing the weight decay for better training. Our NVSA achieves an average accuracy of 68.3%, being competitively with the reproduced MRNet.

SUPPLEMENTARY REFERENCES

[1]Zhang, C., Jia, B., Zhu, S.-C. & Zhu, Y. Abstract spatial-temporal reasoning via probabilistic abduction and execution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2021).

[2]Johnson, J. *et al.* Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[3]Mettes, P., van der Pol, E. & Snoek, C. Hyperspherical prototype networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32 (2019).

[4]Plate, T. A. *Holographic Reduced Representations: Distributed Representation for Cognitive Structures* (Center for the Study of Language and Information, Stanford, 2003).

[5]Zheng, K., Zha, Z.-J. & Wei, W. Abstract reasoning with distracting features. *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[6]Zhang, C. *et al.* Learning perceptual inference by contrasting. In *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[7]Wu, Y., Dong, H., Grosse, R. & Ba, J. The scattering compositional learner: Discovering objects, attributes, relationships in analogical reasoning. *arXiv preprint arXiv:2007.04212* (2020).

[8]Barrett, D., Hill, F., Santoro, A., Morcos, A. & Lillicrap, T. Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, 511–520 (PMLR, 2018).

[9]Benny, Y., Pekar, N. & Wolf, L. Scale-localized abstract reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021).