A Neuro-vector-symbolic Architecture for Solving Raven's Progressive Matrices

Michael Hersche,[1,2] Mustafa Zeqiri,[2] Luca Benini,[2] Abu Sebastian,[1, a)] and Abbas Rahimi[1, b)]

[1)]*IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland.*

[2)]*Department of Information Technology and Electrical Engineering, ETH Zürich, Gloriastrasse 35, 8092 Zürich, Switzerland.*

Neither deep neural networks nor symbolic AI alone has approached the kind of intelligence expressed in humans. This is mainly because neural networks are not able to decompose joint representations to obtain distinct objects (the so-called binding problem), while symbolic AI suffers from exhaustive rule searches, among other problems. These two problems are still pronounced in neuro-symbolic AI which aims to combine the best of the two paradigms. Here, we show that the two problems can be addressed with our proposed neuro-vector-symbolic architecture (NVSA) by exploiting its powerful operators on high-dimensional distributed representations that serve as a common language between neural networks and symbolic AI. The efficacy of NVSA is demonstrated by solving the Raven's progressive matrices datasets. Compared to state-of-the-art deep neural network and neuro-symbolic approaches, end-to-end training of NVSA achieves a new record of 87.7% average accuracy in RAVEN, and 88.1% in I-RAVEN datasets. Moreover, compared to the symbolic reasoning within the neuro-symbolic approaches, the probabilistic reasoning of NVSA with less expensive operations on the distributed representations is two orders of magnitude faster. Our code is available at `https://github.com/IBM/neuro-vector-symbolic-architectures`.

Human fluid intelligence is the ability to think and reason abstractly, and make inferences in a novel domain. The Raven's progressive matrices (RPM)[1] test has been a widely-used assessment of fluid intelligence and abstract reasoning[2,3]. The RPM is a non-verbal test which involves perceiving pattern continuation, element abstraction, and finding relations between abstract elements based on underlying rules. Each RPM test is an analogy problem presented as a 3×3 pictorial matrix of context panels. Every panel in the matrix is filled with several geometric objects based on a certain rule, except the last panel which is left blank. The participants are asked to complete the missing panel in the matrix by picking the correct answer from a set of candidate answer panels that matches the implicit rule (see Methods and Supplementary Fig. 1c). Solving this test mainly involves two aspects of intelligence: visual perception and abstract reasoning.

How perception is combined with reasoning, and how they interact vary greatly across the spectrum of AI architectures. At one end of the spectrum, in the deep learning architectures[4–12] perception has primacy, and reasoning is more likely to adapt to the representations than vice versa. At the opposite ends of the spectrum, in the classical symbolic AI, the perceptual representations are pre-engineered to e.g., emphasize relations, rather than the representations being generated as a consequence of the reasoning processes. In fact, it has been argued that the construction of appropriate representations is part of the reasoning process[13]. Neuro-symbolic architectures further enrich the spectrum of possibilities by utilizing techniques from both ends: they combine subsymbolic (e.g., neural network) with symbolic AI approaches, aiming to reach human-level generalization[14–17]. Considerable effort has been devoted to integrate the two ends that led to state-of-the-art performance of neuro-symbolic architectures in various tasks, e.g., visual question answering[18–21], causal video reasoning[22], and solving RPM[23,24]. However, the resulting neuro-symbolic architectures are not immune to the potential problems of their individual ingredients (i.e., the neuro and symbolic parts), which are explained in the following.

We first explain the binding problem[25] in neural networks that refers to their inability to recover distinct objects from their joint representation. This inability prevents the neural networks from providing an adequate description of real-world objects or situations that can be represented by hierarchical and nested compositional structures[26]. When we consider the fully local representations, an item of any complexity level can be represented by a single unit, e.g., by one-hot code. Such local representations limit the number of representable items to the number of available units in the pool, and hence cannot represent the combinatorial variety of real-world objects. To address this issue, the distributed representations can provide enough capacity to represent a combinatorially growing number of compositional items. However, they face another issue known as "superposition catastrophe"[27,28]. Let us consider four atomic items `red`, `blue`, `square`, and `triangle`. For representing two composite objects, e.g., a `red square` and a `blue triangle`, the activity patterns corresponding to their atomic items are superimposed without increasing the dimensionality. As shown in Fig. 1a, this results in a blend pattern that is ambiguous because the

a)Electronic mail: ase@zurich.ibm.com

b)Electronic mail: abr@zurich.ibm.com

patterns of all four atomic items are coactivated, which causes "ghost" or "spurious" memory for unavailable objects such as `red triangle` or `blue square`. As a common practice, to bypass this problem, the neuro-symbolic architectures employ two complementary networks (e.g., a Mask R-CNN followed by a ResNet-34) to be able to unambiguously extract the item attributes from multiple objects in an image[18–21]. The first network (Mask R-CNN) generates segment proposals for all objects such that each object can be processed individually by another network. Next, the generated bounding box for each single object paired with the original image is sent to the second network (ResNet-34) for extracting all attributes. However, this approach of having multiple networks increases the number of weights.

The second ingredient of the neuro-symbolic architectures is the symbolic engine which is used for logical reasoning. For solving the RPM tests, the symbolic logical reasoning is implemented as a probabilistic abduction reasoning[23] that can be viewed as searching for a solution in a space defined by prior background knowledge. The background knowledge about the rules is represented in symbolic form by describing all possible rule realizations that could govern the RPM tests[2]. Then, the reasoning process can be solved via an exhaustive search over these symbols to abduce the probability distribution of the rules. The computational complexity of the exhaustive search rapidly increases with the number of objects in the RPM panels. In effect, this exhaustive search problem hinders the utilization of the symbolic logical reasoning for end-to-end training and real-time inference.

We aim to address the two aforementioned problems by exploiting vector-symbolic architectures (VSAs)[29–33]. VSAs are computational models that rely on high-dimensional distributed vectors and algebraic properties of their powerful operations to incorporate the advantages of connectionist distributed representations as well as structured symbolic representations (See[34,35] for review). In a VSA, all representations—from atomic to composite structures—are high-dimensional holographic vectors of the same, fixed dimensionality. The VSA representations can be composed, decomposed, probed, and transformed in various ways using a set of well-defined operations, including binding, unbinding, bundling (i.e., additive superposition), permutations, inverse permutations, and associative memory (i.e., cleanup). Such characteristics of compositionality and transparency enable the use of VSAs in analogical reasoning[36–40], but these inspiring works do not have any perception module to process the raw sensory inputs. Instead, they assume there would be a perception system, e.g., a symbolic parser, providing the symbolic representations that support the reasoning. Further, their inductive logical reasoning has eluded the uncertainty in perception, and supported only one type of rules in RPM: the progression rule[39,40].

Here, we propose a neuro-vector-symbolic architecture (NVSA) in which two powerful machineries, namely, deep neural networks and VSAs are synergistically combined to codesign a visual perception frontend and a probabilistic reasoning backend. Hereby, the demands of reasoning can drive the construction of appropriate perceptual representations. This synergy permits both perception and reasoning ends to tap into the rich resources of VSA as a general computing framework to overcome the problems of neural binding and exhaustive symbolic search as mentioned above, and yet they can be trained end-to-end. Accordingly, the main design objective of the NVSA frontend is to transduce the visual raw sensory inputs to the fixed-width VSA representations (as nested composable structures shown in Fig. 1b). It is achieved by training nonlinear transformations of a neural network as a flexible means of representation learning over VSA. The resulting NVSA frontend addresses the binding problem in the neural networks, especially the superposition catastrophe, by effectively mapping the raw image of multiple objects to the structural VSA representations that still maintain the perceptual uncertainty. These representations can be readily used to solve visual analogy tasks by directly applying binding operations, or can be used to infer the probability mass functions per individual object attributes for further reasoning processes in the backend. The NVSA backend maps the inferred probability mass functions into another vector space of VSA such that the exhaustive probability computations and searches can be substituted by algebraic operations in that vector space. The design objective of the NVSA backend is to facilitate a differentiable and computationally efficient probabilistic reasoning process to support a set of RPM rules. The VSA operations offer distributivity and computing-in-superposition, which significantly reduce the computational costs thus performing probabilistic abduction and execution to predict the missing RPM panel in a generative and real-time manner.

I. NVSA FRONTEND: PERCEPTION

Prior to describing the NVSA frontend, let us provide a brief background on VSA. VSA is a family of similar systems that represent data using random high-dimensional vectors (see[34] for a review). In this section, we use VSA whose vector entries are restricted to being dense bipolar[30]. Initially, one or multiple codebooks are defined as $X := \{\mathbf{x}_i\}_{i=1}^m$, where the elements of each atomic d-dimensional vector $\mathbf{x}_i \in \{-1, +1\}^d$ are randomly drawn from a Rademacher distribution (i.e., equal chance of elements being "-1" or "+1"). We compare two vectors using the cosine similarity (sim). The similarity between two atomic vectors is close to zero with a high probability when d is sufficiently large, typically in the order of thousands[33]; hence, all vectors in the codebooks are quasi-orthogonal with respect to each other.

For representing a given data structure, VSA provides a set of well-defined vector operations. *Bundling* ($\oplus$), superposition, or addition of two or more atomic vectors is defined as the element-wise sum with a subsequent bipolarization operation that sets the negative elements to "-1" and the positive ones to "+1". This operation preserves similarity. On the other hand, *binding* ($\odot$), or multiplication, of two or more vectors is defined with their element-wise product. Binding yields a vector that is dissimilar

to its atomic input vectors. Every vector $\mathbf{x}_i \in X$ is its own inverse with respect to the binding operation, i.e., $\mathbf{x}_i \odot \mathbf{x}_i = \mathbf{1}$, where $\mathbf{1}$ is the d-dimensional all 1-vector. Hence, the individual factors in a product can be exactly retrieved by *unbinding*: $\mathbf{x}_i \odot (\mathbf{x}_i \odot \mathbf{x}_j) = (\mathbf{x}_i \odot \mathbf{x}_i) \odot \mathbf{x}_j = \mathbf{x}_j$.

The NVSA frontend consists of a trainable neural network and a VSA machinery whose algebra allows construction of appropriate perceptual representations suitable for the cognitive demands. The design of frontend is inspired by the expressiveness of high-dimensional VSA representations as a *common language* between the symbolic representation of a set of objects and their data-driven representations obtained from a neural network. On the one hand, the real-world objects with arbitrary complexities can be symbolically described by VSA, e.g., a scene with multiple objects can be expressed as a hierarchical nested compositional structure of attributes, objects, and a set of objects—all with fixed-width vector arithmetic as shown in Fig. 1b. On the other hand, the resulting VSA representations can also serve as an interface to a data-driven neural network whose layers of transformation are trained accordingly to transduce the raw image of the scene to the VSA representations (see Fig. 2b). This novel combination merges the data-driven representation of the scene into its descriptive symbolic representation for facilitating down-stream reasoning tasks, without exploding the representation dimensionality, or facing the superposition catastrophe. In the following, we describe the steps involved in the NVSA frontend.

A. Defining VSA representations

This first step is to define a dictionary whereby the atomic concepts, their hierarchical compositions, and their relations can be described using the fixed-width vectors. What these concepts are can be guided by the cognitive demands. For instance, in solving RPMs, the reasoning process requires the probability mass functions of the object attributes, therefore the dictionary should be able to provide such appropriate representations by expressing attributes per objects which, in turn, fulfills the reasoning demand. The construction of the dictionary can be done by the application of the VSA operations on the VSA-encoded concepts as explained in the following. Let us consider the object attributes as the atomic concepts. The encoding process starts by randomly generating a set of codebooks for the attributes of interest, e.g., one codebook for the colors ($\mathbf{x}_{\text{red}}, \mathbf{x}_{\text{blue}}$), and another codebook for the shapes ($\mathbf{x}_{\text{square}}, \mathbf{x}_{\text{triangle}}$). Each codebook contains as many atomic d-dimensional vectors as there are attribute values. It therefore provides a symbolic meaning for individual atomic vectors. For describing an object with these two attributes, a product vector $\mathbf{w}$ can be computed by binding two vectors, one drawn from each of the codebooks (see Fig. 1b). For example, a `red square` object is represented as $\mathbf{w} = \mathbf{x}_{\text{red}} \odot \mathbf{x}_{\text{square}}$. The random construction of the atomic vectors and the properties of multiplicative binding yield a unique $\mathbf{w}$ that is quasi-orthogonal (i.e., dissimilar) to the VSA representations of all other attributes and their combinations (e.g., $\mathbf{x}_{\text{blue}} \odot \mathbf{x}_{\text{triangle}}$, or $\mathbf{x}_{\text{blue}} \odot \mathbf{x}_{\text{square}}$, or $\mathbf{x}_{\text{red}} \odot \mathbf{x}_{\text{triangle}}$). This means that the expected cosine similarity between two different object vectors is approximately zero with a high probability. Therefore, when their VSA representations are coactivated, it results in minimal interference such that each object can be recovered (see Fig. 1c.)

We have shown how to derive the object vectors from the elementary attribute vectors. In the next level of the hierarchy, we are interested in an object-centric definition of the scene. Here, we define the scene as the union of the objects. Therefore, a scene with multiple objects is represented by bundling together their object vectors: $\mathbf{s} = (\mathbf{x}_{\text{red}} \odot \mathbf{x}_{\text{square}}) \oplus (\mathbf{x}_{\text{blue}} \odot \mathbf{x}_{\text{triangle}})$. The bundling operation creates an equally-weighted superposition of multiple objects, and preserves similarity; hence, the bundled vector $\mathbf{s}$ is similar to both object vectors present in the scene, and dissimilar to other vectors in the system, as shown in Fig. 1c. This similarity preservation property allows the bundled vector to be solely matched with its constituent object vectors, which avoids the superposition catastrophe by design (e.g., $\text{sim}(\mathbf{s}, \mathbf{x}_{\text{red}} \odot \mathbf{x}_{\text{triangle}}) \approx 0$). In summary, VSA can construct higher-level symbols of multiple objects by combining lower-level symbols of individual objects, and more elementary symbols of object attributes by using its dimensionality-preserving operators.

Next, we illustrate the generalization of the previous 2-attribute to 4-attribute objects suitable for solving RPM (an RPM test example is shown in Fig. 2a). Similarly, we randomly generate a set of compact codebooks for the available attributes in the RAVEN dataset as $T := \{\mathbf{t}_i\}_{i=1}^{5}$, $S := \{\mathbf{s}_i\}_{i=1}^{6}$, $C := \{\mathbf{c}_i\}_{i=1}^{10}$, and $L := \{\mathbf{l}_i\}_{i=1}^{22}$ which respectively represent the type, size, color, and position of a single object, considering the cross-configuration equivalent positions with the same proportions (see Methods for more details). We set $d = 512$ that is sufficiently large to supply the atomic quasi-orthogonal vectors for every attribute value, while it is at least one order of magnitude smaller than the number of all possible combinations of attribute values ($m = 6600$) for a single object. Using these four codebooks, a quasi-orthogonal vector for every possible combination of a single object is computed as the Hadamard product of its attribute vectors (i.e., a 4-way multiplicative binding). These d-dimensional vectors are stored in a dictionary $\mathbf{W} \in \{-1, +1\}^{m \times d}$ which contains all m possible single object combinations. An arbitrary set of these single object vectors can be further composed by the bundling operation to describe e.g., an RPM panel as the union of its distinct objects.

The dictionary $\mathbf{W}$ is generated once, based on the initialization of codebook vectors, and is kept frozen during training. As the second step of the frontend, in the next subsection, we show how this dictionary can be connected to the data-driven representations of its objects.

B. Neural network representation learning over VSA

To avoid the pitfalls of pure symbolic approaches and the need for a symbolic parser, we exploit the deep neural network representation learning over the defined VSA representations ($\mathbf{W}$) such that an image panel $\mathbf{X} \in \mathbb{R}^{r \times r}$ with resolution r can be transformed and matched to the corresponding VSA representations using a mapping f_θ with learnable parameters θ. To do so, we propose using a ResNet-18, motivated by its good performance[7], and interface its fully connected layer to the dictionary $\mathbf{W}$ as shown in Fig. 2b. With this interface, the last fully connected layer has an output dimension $d = 512$ to be able to search on $\mathbf{W}$. We insert a hyperbolic tangent (*tanh*) activation at the output of ResNet-18 to guide its real-valued output towards the bipolar representation of $\mathbf{W}$. By exploiting the VSA principles, the ResNet-18 can learn to superpose multiple, say k, objects in the d-dimensional vector from which all the attributes of the compound objects can be reliably recovered by $\mathbf{W}$ without facing the superposition catastrophe. Alternatively, other neuro-symbolic architectures[18–21] require at least two separate neural networks that collectively increase the number of trainable parameters by $\approx 6\times$ compared to our NVSA frontend.

The NVSA frontend and backend can be trained end-to-end as we show in the next section. However, in a fully supervised setting in which the labels of the visual attributes are given, the NVSA frontend can be trained independent of the backend. Let $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m$ be the quasi-orthogonal representations of the object classes within $\mathbf{W}$, where m is the number of single-object combinations. For an image panel $\mathbf{X}$ containing k objects, with k target indices $\{y_i\}_{i=1}^{k}$, the trainable parameters θ of ResNet-18 can be optimized. The optimization maximizes the similarity between the output query $\mathbf{q} = f_\theta(\mathbf{X})$ of ResNet-18 and the bundled vector $\mathbf{w}_{y_1} \oplus ... \oplus \mathbf{w}_{y_k}$ using a novel additive cross-entropy loss together with batched gradient descent. We provide the details of this loss, and show its superiority compared to other loss functions and perception methods (see Supplementary Note 1a).

We also analyze the generalization of the NVSA frontend to unseen combinations of attribute values in a novel object. Although we observe that the frontend with the multiplicative binding cannot generalize to unseen combinations of the attribute values, we enhance it by a multiplicative-additive encoding that can generalize up to 72% (see Supplementary Note 1b). The multiplicative binding-based encoding however generalizes well to unseen combinations of multiple objects (see Supplementary Note 1c). Importantly, the employed multiplicative encoding results in learning the powerful perceptual representations in readiness for solving high-level reasoning tasks, such as visual analogy. We have shown 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 relation, or multiple relations, between its two sets of objects ($A : B$), and a target domain that shares the same relation(s) between its object sets ($\alpha : \beta$). Binding the neural network representations obtained from the source domain allows to capture the relation(s) solely from a single example, that can be applied to novel circumstances in the target domain by another application of binding operation. See Supplementary Note 2.

C. Inferring probability mass functions from data-driven VSA representations

We describe the last step of the frontend here. Given an RPM panel, the ResNet-18 generates a VSA query vector that can be decomposed into the constituent object vectors, each derived from a unique combination of the attributes. 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 $\mathbf{z}$. Since the structure of the dictionary matrix is known, we can infer the attributes namely, position, color, size, and type from the detected indices. Based on the similarity scores, we derive the probability mass functions (PMFs) for every object attribute, which include $\mathbf{v}_{type}$, $\mathbf{v}_{size}$, and $\mathbf{v}_{color}$ by marginalizing all non-negative similarity scores over all attribute combinations and a consecutive scaled softmax activation per object attribute. In addition, we derive the probability of whether an object is present at a given position in $\mathbf{v}_{exist}$. After inferring these PMFs of the object attributes, we infer the PMFs of the panel attributes. We combine all object PMFs to five PMFs which represent the position, number, type, size, and color distribution of the entire panel. These PMFs are denoted by $P := (\mathbf{p}_{pos}, \mathbf{p}_{num}, \mathbf{p}_{type}, \mathbf{p}_{size}, \mathbf{p}_{color})$. See Methods for more details.

Given an RPM test, we obtain a set of PMFs $P^{(i,j)}$ for each of the eight context panels, indexed by their row i and column j, and a set of PMFs $P^{(i)}$ for each of the answer panels, as shown in Fig. 2c. The set of context PMFs ($P^{(i,j)}$) form the probabilistic scene representations that are further transformed in the backend whose objective is to find the underlying rule. The chosen rule is executed to generate $\hat{P}^{(3,3)}$ for the missing panel.

II. NVSA BACKEND: REASONING

Here, we describe the NVSA backend that provides a computationally-efficient, differentiable, and transparent implementation of the probabilistic abductive reasoning. The NVSA backend re-designs reasoning by exploiting the VSA representations and operators that permit handling large problem sizes that cannot be solved by traditional symbolic search-based reasoning approaches. As the first step in the NVSA backend, the inferred PMFs from the frontend are transformed into the distributed VSA representations in an appropriate vector space. Next, this vector space should allow the application of VSA operators to implement the first-order logical rules such as addition of the attribute values, or subtraction, distribution, and more (see Fig. 3a).

The efficient VSA manipulations result in computing the rule probability for each possible rule, from which the most probable rule can be chosen and executed. These two main steps in the backend, followed by the end-to-end training of frontend and backend are described in the following subsections.

A. VSA representations of probability mass functions

The RAVEN dataset[7] applies an individual rule to each of the five attributes (position, number, color, size, and type), which is either `constant`, `progression`, `arithmetic`, or `distribute three` (see Methods). The rules are applied row-wise across the context matrix. Based on the downstream rule, each attribute can be treated as continuous where there are relations among its set of values, or discrete where there are no explicit relations between the values. For instance, the color attribute is treated as discrete in the `distribute three` rule, while the `arithmetic` rule treats it as continuous. To make our VSA transformation general, we treat every attribute as both discrete and continuous, and it is up to the rule to use the proper representation. To achieve this in NVSA backend, we switch from the previously used bipolar dense representations to binary sparse block codes[41,42]. This VSA framework with the help of fractional power encoding[32] permits the representation of continuous PMFs. The basis vectors in the binary sparse block codes are d-dimensional, binary-valued vectors with κ non-zero elements. More specifically, the vectors are divided in κ distinct blocks which contain exactly one non-zero element. The binding in the binary sparse block codes is defined as the block-wise circular convolution; similarly, the unbinding is the block-wise circular correlation. The similarity of two vectors is the sum the inner product normalized by the number of blocks κ. The bundling of two or more vectors is computed via the element-wise addition. Optionally, the bundled vector could be sparsified to have only one non-zero element per block again, however, this results in loss of information. Hence, in this work, the bundling is performed without sparsification.

In the following, we illustrate how a PMF can be transformed to this VSA format. To represent the PMF of an attribute in the VSA space, we first generate a codebook $B := \{\mathbf{b}_i\}_{i=1}^n$, where $\mathbf{b}_i \in \{0,1\}^d$. For a discrete attribute, we use a codebook with n unrelated basis vectors $\mathbf{b}_i$. For representing the PMF of a continuous attribute, we use a codebook with basis vectors generated by the fractional power encoding[32], 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., $\mathbf{b}_v = \mathbf{e}^v$. See Methods on how to create the codebooks for discrete and continuous attributes. Each PMF is represented through the weighted superposition with the values in the PMF used as weights and the corresponding codewords as basis vectors (see Fig. 3a):

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

Every attribute PMF is transformed separately to its corresponding VSA representation, e.g., the PMF of the attribute number is transformed to $\mathbf{a}_{num}^{(i,j)} := g(\mathbf{p}_{num}^{(i,j)})$.

B. VSA-based probabilistic abduction and execution

The attribute PMFs of the panel are mapped to the VSA format where we can use the VSA algebra to implement the functions embedded in the underlying rules. Let us consider the `arithmetic plus` rule for the number attribute, which is treated as continuous and shown in Fig. 3b. In each row, the number of objects in the third panel is the sum of the number of objects in the first two panels. As this rule is of continuous nature, we represent the PMFs using fractional power encoding. The VSA representations of PMFs obtained in equation (1) are bound to compute $\mathbf{r}_i$ vectors for the first and second row using the first two panels:

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

To better understand equation (2), let us assume that the distribution of the PMFs of the context panels is maximally compact, i.e., the values in $\mathbf{p}_{num}^{(i,j)}$ are "1" at the correct number of objects and "0" elsewhere. Then, the bound vector of the first row can be formulated as $\mathbf{r}_1^+ = \mathbf{e}^{v_1} \odot \mathbf{e}^{v_2} = \mathbf{e}^{v_1+v_2}$, where v_1 and v_2 are the numbers of objects in the first and second panel. If the rule applies, i.e. $v_3 = v_1 + v_2$, we expect the bound vector $\mathbf{r}_1^+$ to be identical to the VSA representation of the last panel in the row $\mathbf{a}^{(1,3)} = \mathbf{e}^{v_3} = \mathbf{e}^{v_1+v_2}$, thanks to the properties of fractional power encoding.

For supporting arbitrary PMFs, we validate the rule using the similarity between the bound vectors. Combining the row-wise similarities with additional constraints yields us an estimation of the rule probability:

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

where h_a is an additional rule-dependent constraint (see Supplementary Note 3). When the rule probability for the `arithmetic plus` is the highest among all possible rules, we estimate the vectorized representation of the number attribute for the missing panel by

$$\hat{\mathbf{a}}^{(3,3)} = \mathbf{a}^{(3,1)} \odot \mathbf{a}^{(3,2)}. \tag{4}$$

This bound vector represents the estimation of the PMF. If the PMFs in the last row are maximally compact, the bound vector corresponds to the correct number of objects of the missing panel. Otherwise, the bound vector represents a superposition of the correct number vector and additional terms which can be considered as noise terms, stemming from the smaller non-zero contributions in the PMF.

To compute the PMF of the missing panel attribute, we do an associative memory search between the bound vector and all atomic vectors in the codebook B, followed by a normalization:

$$\hat{\mathbf{p}}_{num}^{(3,3)} = \text{norm}\left(\left[\text{sim}\left(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_1\right), \text{sim}\left(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_2\right), ..., \text{sim}\left(\hat{\mathbf{a}}^{(3,3)}, \mathbf{b}_n\right)\right]\right). \tag{5}$$

Next, we show how the NVSA backend supports the rules with the discrete treatment of the attributes such as the `distribute three` rule. Without loss of generality, we explain our method for the position attribute in the panel constellation with a 3x3 grid (see Fig. 3c). The position is described with a 9-bit code where a "1" indicates that the position is occupied inside the 3x3 grid. This 9-bit position index p takes values from 1 to $n = 511$, considering the constraint of having at least one object per panel. A different value of the position attribute (from 1 to n) appears in each of the three panels of a row. The `distribute three` requires that the same values appear in each row with a distinct permutation. The same holds with respect to the columns.

We transform the position PMF of every panel using equation (1) in combination with a discrete codebook B. These VSA representations are used to compute the product vectors for the first and second rows and columns of the context matrix:

$$\mathbf{r}_i = \mathbf{a}^{(i,1)} \odot \mathbf{a}^{(i,2)} \odot \mathbf{a}^{(i,3)}, \tag{6}$$

$$\mathbf{c}_j = \mathbf{a}^{(1,j)} \odot \mathbf{a}^{(2,j)} \odot \mathbf{a}^{(3,j)}, \quad i, j \in \{1, 2\}. \tag{7}$$

Equations (6) and (7) describe a VSA-based conjunctive formula grounded over the row and column being considered, respectively. For example, given a set of arbitrary PMFs in a row, the resulting product vector ($\mathbf{r}_1$ or $\mathbf{r}_2$) is unique. However, for any order of PMFs in the row, the computed product vectors are the same due to the commutative property of the binding operation. We exploit this property to detect whether the `distribute three` rule applies by simply checking if the product vectors are similar among rows and columns, i.e., $\text{sim}(\mathbf{r}_1, \mathbf{r}_2) \gg 0$ and $\text{sim}(\mathbf{c}_1, \mathbf{c}_2) \gg 0$, and combine them together to estimate the rule probability

$$\mathbf{u}[\text{distribute three}] = \text{sim}(\mathbf{r}_1, \mathbf{r}_2) \cdot \text{sim}(\mathbf{c}_1, \mathbf{c}_2) \cdot h_d(\mathbf{a}^{(1,1)}, \mathbf{a}^{(2,1)}, ..., \mathbf{a}^{(2,3)}), \tag{8}$$

where h_d is an additional rule-dependent constraint (see Supplementary Note 3). To execute the rule, we first unbind two vectors ($\mathbf{a}^{(3,1)}$ and $\mathbf{a}^{(3,2)}$) from one of the row product vectors ($\mathbf{r}_1$ or $\mathbf{r}_2$), which results in an unbound vector $\hat{\mathbf{a}}^{(3,3)}$. The PMF $\hat{\mathbf{p}}_{pos}^{(3,3)}$ of the missing panel is estimated by the associate memory in equation (5) which searches on the values of the position attribute.

The associative memory search is only limited to the n atomic vectors in the codebook B; hence, our NVSA backend requires $\mathcal{O}(n)$ in time and space. This is a significant reduction compared to pure symbolic search-based reasoning approaches which search exhaustively through all possible rule implementations that demanding up to $\mathcal{O}(n^3)$ in time and space. For example, the previously described `distribute three` rule on the attribute position would have $\binom{n=511}{3} \cdot 12 \geq 26 \cdot 10^7$ different rule implementations in the 3x3 grid constellation which is prohibitive to compute. This exhaustive rule search forces the neuro-symbolic approach in[23] to considerably limit its search space at the cost of lower accuracy. Instead, our approach efficiently covers the entire search space by simple binding and unbinding operations on the VSA representations followed by a linear associative memory search whose time and space complexity is set as the cube root of the exhaustive search space. This computational advantage of VSA is mainly due to performing search-in-superposition. For example, by comparing the VSA representations of the first and second row, we can sum over all possible combinations in superposition:

$$\text{sim}(\mathbf{r}_1, \mathbf{r}_2) = \text{sim}(\mathbf{a}^{(1,1)} \odot \mathbf{a}^{(1,2)} \odot \mathbf{a}^{(1,3)}, \mathbf{a}^{(2,1)} \odot \mathbf{a}^{(2,2)} \odot \mathbf{a}^{(2,3)}) \tag{9}$$

$$= \text{sim}\left(\left(\sum_{k=1}^{n} \mathbf{p}^{(1,1)}[k] \cdot \mathbf{b}_k\right) \odot \left(\sum_{k=1}^{n} \mathbf{p}^{(1,2)}[k] \cdot \mathbf{b}_k\right) \odot \left(\sum_{k=1}^{n} \mathbf{p}^{(1,3)}[k] \cdot \mathbf{b}_k\right), \mathbf{a}^{(2,1)} \odot \mathbf{a}^{(2,2)} \odot \mathbf{a}^{(2,3)}\right). \tag{10}$$

Without the VSA representations, one would need to compute the maximally expanded version. See Supplementary Note 3 for details about our implementation of the `artihmetic minus`, the `progression`, and the `constant` rule.

C. End-to-end training

We train our NVSA frontend and backend end-to-end. Note that only the neural part of the frontend (i.e., ResNet-18) has trainable parameters, while the dictionary ($\mathbf{W}$) and all the parameters in the backend (e.g., rule representations) are frozen. Every training RPM example provides eight context panels ($\mathbf{X}^{(1,1)}, ..., \mathbf{X}^{(3,2)}$), eight candidate answer panels ($\mathbf{X}^{(1)}, ..., \mathbf{X}^{(8)}$), the ground-truth answer y, and the ground-truth rule per attribute $\mathbf{r}$. First, we pass the context panels through the NVSA frontend and infer $P^{(i,j)}$; similarly, we infer $P^{(i)}$ for the candidate answer panels. Using $P^{(i,j)}$, we compute the rule belief per attribute using the NVSA backend (e.g., using equations (3) and (8) for `arithmetic plus` and `distribute three`). Based on the distribution of the rule beliefs, we then sample an action per attribute and execute it, yielding five probability distributions for the attributes $\hat{P}^{(3,3)}$. 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.

Inspired by PrAE[23], we mutually minimize a loss based on REINFORCE[43] and an additional auxiliary loss. The REINFORCE-based loss combines the negative cross-entropy on the scores, interpreted as a reward function, with the log-likelihood of the sampled action. The auxiliary loss sums up the negative log-likelihood of the ground-truth rules (See Methods). Hence, by minimizing the auxiliary loss, we train the frontend to map the context panels to the PMFs based on which the rule detection yields the correct rule.

III. RESULTS

We evaluate NVSA on the RAVEN[7], I-RAVEN[8], and PGM[4] datasets (see Methods). First, we consider more diverse RAVEN and I-RAVEN datasets. Fig. 4 compares the classification accuracy with the state-of-the-art models in pure deep neural networks (SCL[44]) and neuro-symbolic AI (PrAE[23]), where we have retrained both models five times using different random seeds and used the checkpoint with the highest accuracy on the validation set. A separate SCL model was trained per constellation. On the RAVEN dataset, NVSA achieves an average accuracy of 87.7%, outperforming SCL by 0.5% and PrAE by 27.4%.

There is a short-cut solution in the answer panels of the RAVEN dataset (see Methods). It has been shown that the shortcut pattern can be leveraged by deep neural networks, e.g., CoPINet[6] achieved a higher accuracy when being trained and tested exclusively on the answer panels without considering the context panels at all (context-blind)[8]. In this regard, the understanding of the context matrix and its underlying rules is bypassed by shortcut learning from the answer set only. Therefore, it is recommended to use the I-RAVEN dataset[8], which provides unbiased fair answer panels, when testing RPM reasoning models[45,46]. NVSA achieves the highest accuracy on the I-RAVEN dataset too (88.1%) on average, while the majority of deep learning approaches[4–7,47] face a large accuracy drop by showing $< 50\%$ accuracy on average. Our NVSA does not face any accuracy drop by switching from RAVEN to I-RAVEN because it cannot rely on such a shortcut by design: based on the highest probable rules, it first makes a prediction of the PMFs of the empty panel before individually comparing it to each PMF of the answer panel. The controllability and explainability of NVSA is a great advantage for problems that require it. NVSA also significantly outperforms SCL by 4.2% and PrAE by 17.0% on I-RAVEN, on average. Extended Tables II and III present a detailed comparison with the state-of-the-art methods in the tabular format.

Next, we compare the accuracy and the compute time of the NVSA backend with the PrAE reasoning backend by providing the ground-truth attribute values. As shown in Table I, the PrAE reasoning backend reaches relatively lower accuracies (94.21%–95.68%) in the 2x2 grid, the 3x3 grid, and the out-in grid compared to the other constellations. We identify the root cause of the low accuracy in these three constellations to be the approximations made in the exhaustive search by applying restrictions to get faster execution. We remove these search restrictions from PrAE and create an unrestricted PrAE. This increases the accuracy of those three constellations to 97.5%–99.22%. While the compute time of the unrestricted PrAE remains similar for most configurations, it increases rapidly for the 3x3 grid, requiring 15,408 minutes (10.7 days) instead of the previous 648 minutes (10.8 hours) in the PrAE with restricted search for solving 2000 RPM tests. Note that we run the experiments on the CPUs as the unrestricted PrAE demands more than 53 GB memory that could not fit the GPU providing 32 GB memory (see Methods). However, our NVSA reasoning backend effectively resolves this bottleneck: it reduces the computation time on the 3x3 grid to 63.2 minutes, which is $244\times$ faster than the unrestricted PrAE, and the memory demand to < 10 GB, while maintaining the high accuracy (96.89% vs. 97.50%). Moreover, we demonstrate that frontend and backend of our NVSA can be trained end-to-end, practically in any constellations, and it provides real-time inference for solving the RPM tests (see Supplementary Video).

Moreover, we also evaluate NVSA on the PGM dataset[4], being the first neuro-symbolic approach targeting this dataset, while other neuro-symbolic works[23,24] only targeted RAVEN/I-RAVEN dataset. NVSA achieves an average accuracy of 68.3% and is highly competitive with the reproduced state-of-the-art MRNet. See Supplementary Note 5 for more details.

Lastly, we showcase the out-of-distribution generalizability of our NVSA with respect to unseen attribute-rule pairs in the I-RAVEN dataset. More specifically, we evaluate whether our model is able to solve tasks containing 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). Our NVSA outperforms the baselines (LEN[5] and CoPINet[6]) by a large margin in all unseen attribute-rule pairs (See Supplementary Note 4).

IV. DISCUSSION

The NVSA frontend allows expression of many more object combinations than the dimensions in the vector space. However, it requires to store and search on the dictionary $\mathbf{W}$. Given the quasi-orthogonality of the representations in $\mathbf{W}$, it can be substituted with a set of smaller codebooks by potentially exploiting the VSA operators in a nonlinear dynamical system. A powerful example of this would be the resonator networks[48,49] and their stochastic nonlinear variants[50] that can quickly factorize a product vector in an iterative manner thus reducing the computation/storage demand on the dictionary when decomposing an object vector.

The associative memory search is the central ingredient of NVSA in both perception and reasoning for estimating the PMFs. To reduce the computational complexity of the associative memory, one notable option is to use in-memory computing that executes searches in an analog manner. It has recently been shown that the associative memory can be realized by analog in-memory computing based on crossbar arrays of emerging non-volatile memories[51–54]. Besides improving the computational density and energy efficiency, this paves the way for reducing the timing complexity of the associative memory to $\mathcal{O}(1)$. Other frequent primitives such as binding and bundling can also benefit from low-power hardware realization[55].

By accurately and efficiently solving RPM tests, we have demonstrated that NVSA enhances the aspects of both perception and reasoning by adding a distinctive vectorized flavor to them which is based on the high-dimensional distributed representations and operators of VSA. In the proposed NVSA frontend, instead of naive local or distributed representations for the objects, we exploited high-dimensional VSA representations. A multi-attribute meaning was structurally assigned to every object vector by binding its attribute vectors, which can be further bundled to create a composite vector representing multiple objects—all in a fixed dimension that is significantly lower than the combinatorial attributes. These structured representations were used as the target vectors to train the deep neural network. The training can be done end-to-end, or by using the additive cross-entropy loss when the attribute labels are available. Being able to train this deep transformation permitted the simultaneous inference of multiple attributes of multiple objects in a visual scene with neither exploding the representation dimensionality, nor facing the superposition catastrophe. In the NVSA backend, we proposed a computationally-efficient and differentiable reasoning where the probability mass functions of discrete or continuous attributes are expressed as the VSA representations. This permitted the use of VSA operators to efficiently implement the rules which save the computational cost significantly thanks to the distributivity and computing-in-superposition of VSA. As a result, the time/space computational complexity of the `distribute three` rule search was reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$, leading to two orders of magnitude shorter execution time. It was shown that NVSA surpasses both pure deep learning[10] and neuro-symbolic[23] state-of-the-arts by achieving average accuracy of 87.7% in the RAVEN[7] and 88.1% in the I-RAVEN[8] datasets. NVSA also enabled real-time execution on CPUs, which is 244× faster than the functionally-equivalent symbolic logical reasoning.

NVSA is a significant step towards encapsulating different AI paradigms in a unified framework to address task involving both perception and higher-level reasoning.

METHODS

RAVEN and I-RAVEN dataset containing RPM tests

The RAVEN dataset[7] contains a rich set of RPM tests. Every RPM test consists of an incomplete 3×3 matrix of context panels, and eight candidate answer panels. The goal of solving an RPM test is to understand the row-wise underlining rule set, and then to decide which of the candidate panels is the most appropriate choice to complete the matrix. An example of RPM test can be seen in Supplementary Fig. 1. The RAVEN dataset arranges the RPM tests in seven different constellations, namely center, 2x2 grid, 3x3 grid, left-right, up-down, out-in center, and out-in grid which are shown in Supplementary Fig. 1. The panels have a resolution of $r \times r = 160 \times 160$. The dataset provides 10,000 samples for every constellation, which are divided into six training folds, two validation folds, and two testing folds.

The objects inside the panels have the following attributes: number, position, type, size, and color. RAVEN distinguishes between five different types (triangle, square, pentagon, hexagon, and circle), six sizes (enumerated from 1–6), and ten colors in the form of shadings (enumerated from 1–10). The number of objects present in the panel varies from one to the maximum number of possible objects, which is determined by the constellation, e.g., the 2x2 grid contains maximally four objects. The position attribute describes the occupancy of the objects inside the panel. Its range is constellation-dependent too; e.g., the 2x2 grid has 15 different position constellations. Nine panels are arranged to a 3x3 matrix such that one out of the following four rules applies to each attribute in a row-wise manner. The four types of rules can be summarized as:

- `Constant`: The attribute value does not change per row.

- `Progression`: The attribute value monotonically increases or decreases in a row by a value of 1 or 2.

- `Arithmetic`: The attribute values of the first two panels are either added or subtracted, yielding the attribute value of the third panel in the row.

- `Distribute three`: This rule involves the fact that three different values of an attribute appear in the three panels of every row (with distinct permutations of the values in different rows). The same holds with respect to the columns.

Supplementary Fig. 1 shows an example for each rule governing the position attribute or the number attribute.

The answer choices in the RAVEN dataset are generated in a way such that only one randomly chosen attribute value differs from the correct answer. Consequently, by exploiting this shortcut solution, the correct answer can be found by simply considering the mode of attribute values in the answer set without looking at the context panels, which is considered as unfair[8,10]. To this end, the impartial RAVEN (I-RAVEN[8]) provides an alternative answer set, which is generated with an attribute bisection tree ensuring that the modifications of attribute values are well balanced without any detectable pattern. It is therefore recommended to use the unbiased I-RAVEN when testing RPM reasoning models[45,46].

PMF computation for object attributes and panel attributes

Every panel is represented with the attribute PMFs describing the distribution of the attribute values inside the panel. First, the object PMFs are determined using marginalization with a consecutive softmax activation. The marginalization computes the sum of non-negative cosine similarities between the query and each valid attribute value combination. For example, for the attribute type with value j at location k, we determine the sum by

$$\mathbf{v}_{type}^{\prime(k)}[j] = \sum_{\substack{s \in \{1,\dots,6\} \\ c \in \{1,\dots,10\}}} \mathrm{ReLU}\left(\mathrm{sim}(\mathbf{q}, \mathbf{l}_k \odot \mathbf{t}_j \odot \mathbf{s}_s \odot \mathbf{c}_c)\right), \tag{11}$$

where $\mathrm{ReLU}(\cdot)$ is the rectified linear unit activation. Similarly, the sum is computed for the object PMF of attribute color and size. For marginalizing for object existence ($\mathbf{v}_{exist}^{\prime k}[0]$), we sum over all the attribute value combinations, whereas the value of no existence is given by the cosine similarity between the query and the vector ($\mathbf{f}_k$) which indicates that no object is present at position k:

$$\mathbf{v}_{exist}^{\prime(k)}[0] = \sum_{\substack{t \in \{1,\dots,5\} \\ s \in \{1,\dots,6\} \\ c \in \{1,\dots,10\}}} \mathrm{sim}(\mathbf{q}, \mathbf{l}_k \odot \mathbf{t}_t \odot \mathbf{s}_s \odot \mathbf{c}_c) \tag{12}$$

$$\mathbf{v}_{exist}^{\prime(k)}[1] = \mathrm{sim}(\mathbf{q}, \mathbf{f}_k). \tag{13}$$

In this case, we marginalize over all cosine similarities (i.e., no ReLU activation is applied) as the $\mathbf{v}'^{(k)}_{exist}[1]$ would be 0 with high probability at the beginning of training, rendering the end-to-end training infeasible. Finally, for generating a valid object PMF which sums up to one, we apply a scaled softmax non-linearity on the sum vector. More precisely, for each attribute a as either type, color, size, or exist, and the panel position k, we compute

$$\mathbf{v}_a^{(k)} = \mathrm{softmax}(s_m \cdot \mathbf{v}'^{(k)}_a), \tag{14}$$

where s_m is a trainable, inverse softmax temperature.

Next, the PMFs of the different objects are combined to five PMFs representing the attributes of the panel. The constellation is known to the reasoning backend; hence, the dimensions of the PMFs that depend on the constellations (i.e., position, number) are known, too. The position PMF represents the probability of object occupancy inside a panel. An occupancy p is described with the set I_p containing the occupied positions, e.g., $I_1 = \{1\}$ represents the case where only the first object is occupied, and $I_{511} = \{1, 2, 3, ..., 9\}$ the case where all objects are occupied in a 3x3 grid. The position PMF is derived by

$$\mathbf{p}_{pos}[j] = \prod_{k \in I_j} \mathbf{v}^{(k)}_{exist}[0] \prod_{k' \in \{1,...,9\} \setminus I_j} \mathbf{v}^{(k')}_{exist}[1]. \tag{15}$$

For the attribute number, the PMF is derived from the position PMF with

$$\mathbf{p}_{num}[j] = \sum_{\substack{k=1 \\ s.t.\, j=|I_k|}}^{n_{pos}} \mathbf{p}_{pos}[k], \tag{16}$$

where $|I_k|$ represents the number of occupied positions in the set I_k. For the attributes type, size, and color, the PMFs are determined by combining the position PMF with the corresponding attribute PMFs. For example, the PMF for the attribute type is determined by

$$\mathbf{p}_{type}[j] = \sum_{i=1}^{n_{pos}} \mathbf{p}_{pos}[i] \prod_{k \in I_i} \mathbf{v}^{(k)}_{type}[j]. \tag{17}$$

In some RPM tests, the values of some attributes inside a panel can be different, e.g., the types are different. We represent this case with an inconsistency state for the attributes type, size, and color, by extending the PMF with an additional probability, e.g., for the attribute type

$$\mathbf{p}_{type}[n_{type} + 1] = 1 - \sum_{j=1}^{n_{type}} \mathbf{p}_{type}[j]. \tag{18}$$

PMF transformation of discrete and continuous attributes to VSA

The PMF of a discrete attribute is represented with a vector space which is spanned with unrelated basis vectors $B := \{\mathbf{b}_i\}_{i=1}^n$. Each basis vector $\mathbf{b}_i \in \{0, 1\}^d$ is a d-dimensional κ-sparse binary vector, where the vector is divided in κ blocks each containing one non-zero element whose index drawn from a uniform distribution. For representing the PMF of a continuous attribute, we use a vector space which is spanned with a basis taken from the fractional power encoding[32]. Building the basis of the fractional power encoding begins with randomly initializing one single unitary basis vector $\mathbf{e} \in \{0, 1\}^d$. The basis vector corresponding to any arbitrary attribute value v is defined by exponentiation of the basis vector $\mathbf{e}$ using the value as the exponent. For example, the basis vector corresponding to the value "3" is $\mathbf{e}^3 = \mathbf{e} \odot \mathbf{e} \odot \mathbf{e}$. For representing real values $v \in \mathbb{R}$, the corresponding basis vector can be computed in the block-wise Fourier domain, where the final basis vectors can contain more than κ non-zero elements[42]. In RPMs, however, we exclusively encounter attributes with the integer values, e.g., the size attribute of an object is enumerated from 1 to $n = 6$. Thus, the underlying codebook is κ-sparse, of finite size, and defined as $B := \{\mathbf{e}^i\}_{i=1}^n$. Using this codebook, a PMF of a given continuous attribute is then transformed to the VSA representation using the weighed superposition defined in equation (1).

End-to-end training

In the following, we give a detailed description of the end-to-end training of NVSA. We are given a training RPM task containing the panels $\mathbf{X}$ (8 context and 8 answer panels), the panel index of the ground-truth answer y_{task}, and the ground-truth

rule y_{rule}. Here, we describe the updates of the frontend trainable parameters θ for one attribute, the generalization to all the rules is straightforward. The operation of NVSA results in the PMF estimation of the rule $\mathbf{u}$, the sampled rule $\tilde{u}$, and the scores s derived from the negative JSD. Based on the scores, we compute the categorical cross-entropy loss $l(\mathbf{X}, y_{task})$. Similar to PrAE[23], we update the trainable parameters θ using the following gradient update

$$\theta \leftarrow \theta - \beta \left(\nabla_\theta l(\mathbf{X}, y_{task}) + l(\mathbf{X}, y_{task}) \nabla_\theta \log(\mathbf{u}[\tilde{u}]) - \nabla_\theta \log(\mathbf{u}[y_{rule}]) \right), \tag{19}$$

where β is the learning rate. The first update term minimizes the cross-entropy loss, where the second term, based on REIN-FORCE, operates on the sampled action log probability. Finally, the third auxiliary term operates on the log probability of the ground-truth rule in order to improve the rule detection.

Experimental Setup

We evaluate different methods on the RAVEN[7] and the I-RAVEN[8] datasets. Our NVSA is exclusively trained on the training data from RAVEN while being tested on both RAVEN and I-RAVEN. I-RAVEN provides the unbiased answer panels, while the constellations and the context matrices stay the same as in RAVEN. In our experiments, in the NVSA frontend we set the dimension of the bipolar vectors to d=512, while in the NVSA backend we set the dimension of the binary sparse block codes to d=1024 and κ=4.

We train a separate NVSA, consisting of the frontend (i.e., a trainable ResNet-18 with the frozen $\mathbf{W}$) and the backend, per constellation. Motivated by[11], the ResNet-18 was pre-trained on the ILSVRC2012 ImageNet-1k dataset. We also adapted ResNet18's first convolutional block by reducing its stride from 2 to 1 and removing the maxpooling, which improved the overall accuracy by 2.9% and 2% on RAVEN and I-RAVEN, respectively. The training was performed for 150 epochs using the Adam optimizer with a weight decay of 1e-4 and a constant learning rate of 9.5e-5. All the training hyperparameters are determined based on the end-to-end reasoning performance on the validation set of RAVEN. We searched through possible batch sizes $\{4, 8, 16, 32\}$. For the training on the majority of the constellations the hyperparameter search yielded an optimal batchsize of 16. As an only exception, the 3x3 grid constellation had to be trained with batchsize of 8 due to the large space requirements.

The models are implemented in PyTorch (version 1.4.0) and trained and validated on a Linux using an NVIDIA Tesla V100 GPU with 32 GB memory. We repeat all experiments five times with a different random seed and report the average results and standard deviation to account for training variability.
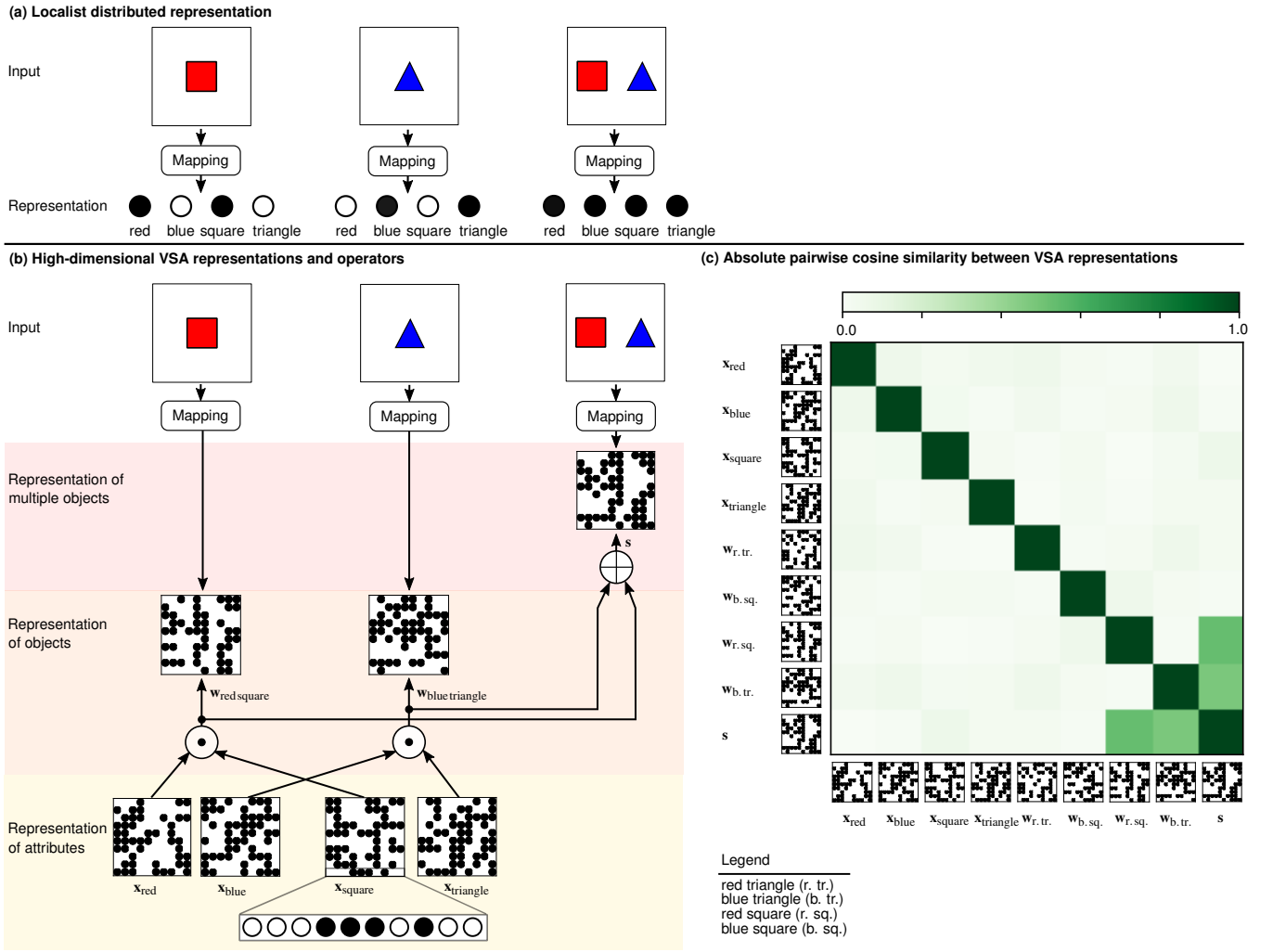
REFERENCES

[1] Raven, J., Court, J. & Raven, J. *Raven's progressive matrices* (Oxford Psychologists Press, 1938).

[2] Carpenter, P. A., Just, M. A. & Shell, P. What one intelligence test measures: a theoretical account of the processing in the raven progressive matrices test. *Psychological review* (1990).

[3] Bilker, W. B. *et al.* Development of abbreviated nine-item forms of the raven's standard progressive matrices test. *Assessment* (2012).

[4] Barrett, D. G. T., Hill, F., Santoro, A., Morcos, A. S. & Lillicrap, T. Measuring abstract reasoning in neural networks. In *Proceedings of International Conference on Machine Learning (ICML)* (2018).

[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] Zhang, C., Gao, F., Jia, B., Zhu, Y. & Zhu, S.-C. Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[8] Hu, S., Ma, Y., Liu, X., Wei, Y. & Bai, S. Stratified rule-aware network for abstract visual reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (2021).

[9] Jahrens, M. & Martinetz, T. Solving raven's progressive matrices with multi-layer relation networks. In *2020 International Joint Conference on Neural Networks (IJCNN)* (2020).

[10] 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).

[11] Zhuo, T. & Kankanhalli, M. Solving raven's progressive matrices with neural networks. *arXiv preprint arXiv:2002.01646* (2020).

[12] Zhuo, T., Huang, Q. & Kankanhalli, M. Unsupervised abstract reasoning for raven's problem matrices. *IEEE Transactions on Image Processing* **30** (2021).

[13] Chalmers, D. J., French, R. M. & Hofstadter, D. R. High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental & Theoretical Artificial Intelligence* **4**, 185–211 (1992).

[14] Fodor, J. A. & Pylyshyn, Z. W. Connectionism and cognitive architecture: A critical analysis. *Cognition* **28**, 3–71 (1988).

[15] d'Avila Garcez, A., Broda, K. B. & Gabbay, D. M. *Neural-Symbolic Learning System: Foundations and Applications* (Springer-Verlag, Berlin, Heidelberg, 2002).

[16] Marcus, G. F. *The algebraic mind: Integrating connectionism and cognitive science* (MIT press, 2001).

[17] Marcus, G. & Davis, E. Insights for AI from the human mind. *Commun. ACM* **64**, 38–41 (2020).

[18] Yi, K. *et al.* Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)* (2018).

[19] Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B. & Wu, J. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations (ICLR)* (2019).

[20] Han, C., Mao, J., Gan, C., Tenenbaum, J. & Wu, J. Visual concept-metaconcept learning. In *Advances in Neural Information Processing Systems (NeurIPS)* (2019).

[21] Mei, L., Mao, J., Wang, Z., Gan, C. & Tenenbaum, J. B. FALCON: Fast visual concept learning by integrating images, linguistic descriptions, and conceptual relations. In *International Conference on Learning Representations (ICLR)* (2022).

[22] Yi, K. *et al.* Clevrer: Collision events for video representation and reasoning. In *International Conference on Learning Representations (ICLR)* (2020).

[23] 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).

[24] Shah, V. *et al.* Knowledge-based analogical reasoning in neuro-symbolic latent spaces. In *Proceedings of the 16th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)* (2022).

[25] Rosenblatt, F. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms* (Spartan Books, 1961).

[26] Rachkovskij, D. A. & Kussul, E. M. Binding and normalization of binary sparse distributed representations by context-dependent thinning. *Neural Computation* **13**, 411–452 (2001).

[27] Malsburg, C. V. D. Am I thinking assemblies? In *Brain Theory*, 161–176 (Springer, Berlin, Heidelberg, 1986).

[28] Malsburg, C. V. D. The what and why of binding: The modeler's perspective. *Neuron* **24**, 95–104 (1999).

[29] Gayler, R. W. Multiplicative binding, representation operators & analogy. In *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences* (1998).

[30] Gayler, R. W. Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience. In *Joint International Conference on Cognitive Science (ICCS/ASCS)* (2003).

[31] Plate, T. A. Holographic reduced representations. *IEEE Transactions on Neural Networks* **6**, 623–641 (1995).

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

[33] Kanerva, P. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* **1**, 139–159 (2009).

[34] Kleyko, D., Rachkovskij, D. A., Osipov, E. & Rahimi, A. A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. *arXiv preprint arXiv:2111.06077* (2021).

[35] Kleyko, D., Rachkovskij, D. A., Osipov, E. & Rahimi, A. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *arXiv preprint arXiv:2112.15424* (2021).

[36] Kanerva, P. Large patterns make great symbols: An example of learning from example. In *Proceedings of the International Workshop on Hybrid Neural Systems* (1998).

[37] Plate, T. A. Analogy retrieval and processing with distributed vector representations. *Expert Systems* (2000).

[38] Gayler, R. W. & Levy, S. D. A distributed basis for analogical mapping. In *New Frontiers in Analogy Research: Proceedings of the Second International Analogy Conference-Analogy* (2009).

[39] Rasmussen, D. & Eliasmith, C. A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science* **3**, 140–153 (2011).

[40] Emruli, B., Gayler, R. W. & Sandin, F. Analogical mapping and inference with binary spatter codes and sparse distributed memory. In *International Joint Conference on Neural Networks (IJCNN)* (2013).

[41] Laiho, M., Poikonen, J. H., Kanerva, P. & Lehtonen, E. High-dimensional computing with sparse vectors. In *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (2015).

[42] Frady, E. P., Kleyko, D., Kymn, C. J., Olshausen, B. A. & Sommer, F. T. Computing on functions using randomized vector representations. *arXiv preprint arXiv:2109.03429* (2021).

[43] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* **8**, 229–256 (1992).

[44] 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).

[45] Małkiński, Mikołaj and Mańdziuk, Jacek. Deep learning methods for abstract visual reasoning: A survey on raven's progressive matrices. *arXiv preprint arXiv:2201.12382* (2022).

[46] Mitchell, M. Abstraction and analogy-making in artificial intelligence. *Annals of the New York Academy of Sciences* **1505**, 79–101 (2021).

[47] Zhuo, T. & Kankanhalli, M. Effective abstract reasoning with dual-contrast network. In *International Conference on Learning Representations (ICLR)* (2021).

[48] Frady, E. P., Kent, S. J., Olshausen, B. A. & Sommer, F. T. Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures. *Neural Computation* **32**, 2311–2331 (2020).

[49] Kent, S. J., Frady, E. P., Sommer, F. T. & Olshausen, B. A. Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods. *Neural Computation* **32**, 2332–2388 (2020).

[50] Langenegger, J. *et al.* In-memory factorization of holographic perceptual representations. *arXiv preprint arXiv:2211.05052* (2022).

[51] Karunaratne, G. *et al.* In-memory hyperdimensional computing. *Nature Electronics* **3**, 327–337 (2020).

[52] Karunaratne, G. *et al.* Robust high-dimensional memory-augmented neural networks. *Nature Communications* **12** (2021).

[53] Lin, H. *et al.* Implementation of highly reliable and energy efficient in-memory hamming distance computations in 1 kb 1-transistor-1-memristor arrays. *Advanced Materials Technologies* **6** (2021).

[54] Li, H. *et al.* Memristive crossbar arrays for storage and computing applications. *Advanced Intelligent Systems* **3** (2021).

[55] Serb, A., Kobyzev, I., Wang, J. & Prodromakis, T. A semi-holographic hyperdimensional representation system for hardware-friendly cognitive computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **378** (2020).

[56] 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).

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

[58] 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).

ACKNOWLEDGEMENTS

FIG. 1: **Illustration of the binding problem in the neural networks and our solution.** **(a)** A localist distributed representation is used for objects with the color and shape attributes. The color attribute is mapped locally on a group of two neurons (red vs. blue). The shape attribute is similarly mapped to square vs. triangle neurons using a trainable mapping. This results in two distinct activated patterns for a red square and a blue triangle. If one of these objects are presented, the responses of the output neurons are sufficient to determine the identity of the object (i.e., its color and shape). However, when both objects are presented, their elementary patterns (symbols) are simultaneously activated that leads to binding ambiguity meaning that the resulting blend activity is insufficient to determine which object is in which color. This is often referred to as the superposition catastrophe[28]. **(b)** The high-dimensional distributed VSA representations and operators can address this problem when properly combined with a neural network as the trainable mapping function. At the lowest level of the hierarchy, the four attribute values are represented by randomly drawing four d-dimensional vectors ($\mathbf{x}_{\mathrm{red}}$, ...). The vectors are dense binary, and arranged as $d = 10 \times 10$ for the sake of visual illustration. At the next level, the red square object is described as a fixed-width product vector by binding two corresponding vectors ($\mathbf{x}_{\mathrm{red}} \odot \mathbf{x}_{\mathrm{square}}$) whose similarity is nearly zero to all attribute vectors and other possible product vectors such as $\mathbf{x}_{\mathrm{blue}} \odot \mathbf{x}_{\mathrm{triangle}}$, $\mathbf{x}_{\mathrm{red}} \odot \mathbf{x}_{\mathrm{triangle}}$, etc. as shown in **(c)**. This quasi-orthogonality allows the VSA representations to be co-activated with minimal interference. At the highest level, the two object vectors are bundled together by similarity-preserving bundling to describe the scene. The bundled vector is similar solely to those objects vectors and dissimilar to others.
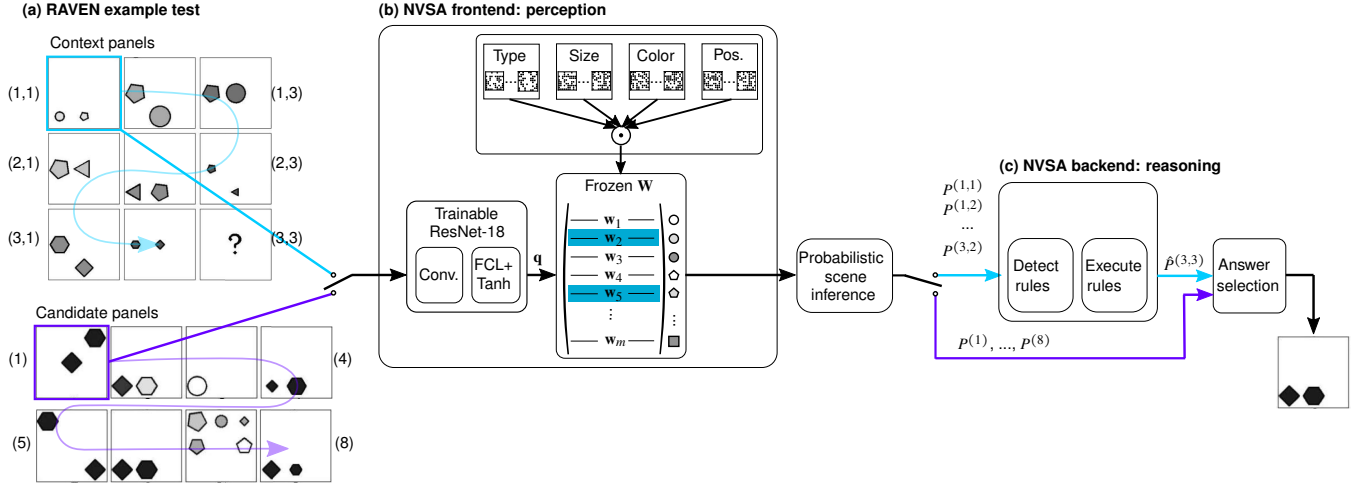
FIG. 2: **Proposed neuro-vector-symbolic architecture (NVSA). (a)** An example of RPM test taken from the RAVEN dataset that can be solved by NVSA. The context panels are composed of eight panels followed by a missing panel at (3,3). The candidate panels provide the potential candidates for the missing context panel. The correct answer is the 6th panel. **(b)** NVSA frontend for perception. The frontend uses a trainable neural network (ResNet-18) and a frozen dictionary $\mathbf{W}$ that is generated by the four codebooks of d-dimensional vectors to cover all possible objects ($\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m$, where $d \ll m$ in the RAVEN dataset). The last fully connected layer of ResNet-18 with a *tanh* activation is connected to $\mathbf{W}$. This dictionary forms a meaningful and semantically-informed VSA representation for individual object that is kept frozen during training, while the weights of ResNet-18 are learned by performing end-to-end training. The training procedure directs ResNet-18 to generate a query vector, $\mathbf{q}$, such that it resembles the superposition of the VSA objects ($\mathbf{w}_2 \oplus \mathbf{w}_5$) available in the panel currently under the visual fixation. This results in merging the data-driven neural representation of the objects with their corresponding compositional VSA representation. **(c)** NVSA backend for reasoning. From the perceived similarities between $\mathbf{q}$ and $\mathbf{W}$, a probabilistic scene inference computes a set of probability mass functions (PMFs) for the attributes. The PMFs are shown as $P^{(i,j)}$ for every panel at location (i, j) in the context matrix. These are used in the NVSA backend to predict the PMF of the missing panel ($\hat{P}^{(3,3)}$). The backend performs probabilistic calculations according to a set of rules in a differentiable and computationally-efficient manner, while not requiring any trainable parameters. The final answer is selected by choosing the candidate panel that minimizes the divergence between the predicted PMF ($\hat{P}^{(3,3)}$) and the PMF of the candidate panels ($P^{(1)}, ..., P^{(8)}$).
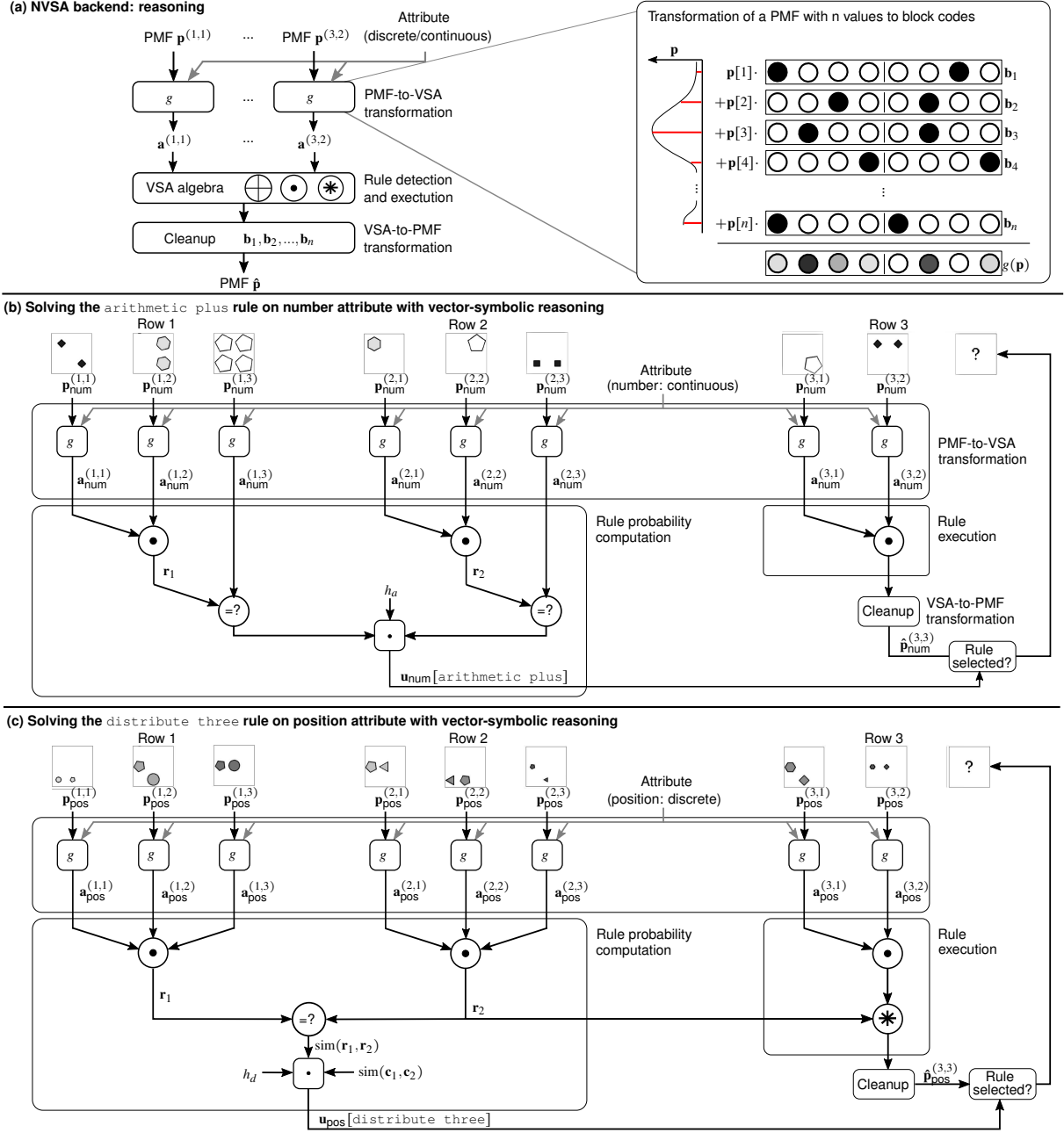
FIG. 3: **NVSA backend. (a)** Steps involved in the NVSA reasoning. First, a PMF of a discrete or a continuous attribute is transformed to the VSA format of binary sparse block codes. This transformation (g) is illustrated for the PMF of an attribute with n possible values. Next, the VSA algebra can be applied on the VSA representations of PMFs to implement the rule of interest. Last, after executing the rule, the resulted VSA representation can be cleaned up by doing an associative memory search on the codebook of values that returns an output PMF. **(b)** The use of backend in solving the `arithmetic plus` rule. The PMF of each panel $\mathbf{p}_{num}^{(i,j)}$ is transformed to the VSA representation. In the rule probability computation step, the VSA representations of first two panels are bound together per row, yielding two row vectors $\mathbf{r}_1$ and $\mathbf{r}_2$. The rule probability is computed by multiplying the similarities between the row vectors $\mathbf{r}_i$ and the last panel of the first two rows, and an additional constraint h_a. In the rule execution, the VSA representation of the missing panel is predicted by binding the VSA representations of position (3,1) and (3,2). Finally, an associative memory cleanup computes the similarities between the predicted vector and all atomic vectors to determine the PMF $\hat{\mathbf{p}}_{num}^{(3,3)}$. **(c)** The NVSA backend for solving the `distribute three` rule. In the rule detection step, the VSA representations of the first two rows are bound together per row, yielding $\mathbf{r}_1$ and $\mathbf{r}_2$. The rule probability is the product of the similarity between the two row vectors, the similarity between the first two column representations $\mathbf{c}_1$ and $\mathbf{c}_2$. In the rule execution, the VSA representation of the missing panel is predicted by unbinding the vector representations of position (3,1) and (3,2) from one of the two row representations, e.g., $\mathbf{r}_2$.
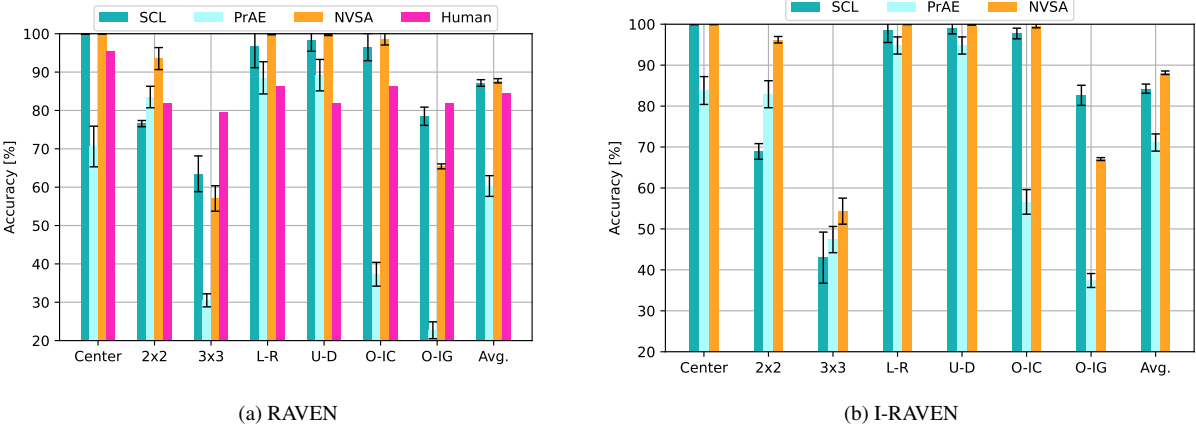
(a) RAVEN

(b) I-RAVEN

FIG. 4: **Average accuracy on the (a) RAVEN and (b) I-RAVEN test sets.** It compares the classification accuracy of the state-of-the-art approaches in deep neural networks (SCL[44]) and neuro-symbolic AI (PrAE[23]) with our NVSA, when all models are trained end-to-end. Error bars indicate the standard deviation from five training and validation runs with different seeds (the exact numeric values are provided in the Extended Tables II and III). The human performance reported in[7] is also shown in **(a)** for the RAVEN dataset. The I-RAVEN dataset is a revised unbiased version of the RAVEN dataset which generates a fair set of answer panels to eliminate the shortcut solution in the RAVEN dataset (see Methods for details).

TABLE I: Reasoning accuracy (%) and CPU compute time (min) when solving 2000 examples on the RAVEN test sets using the ground-truth panel attributes. Experiments where conducted with Intel Xeon E5-2640 cores running at 2.4 GHz. NVSA backend was configured with vector dimension d=1024 and κ=4 blocks.

Method	Accuracy (%)			CPU compute time (min)		
	2x2	3x3	O-IG	2x2	3x3	O-IG
PrAE[23]	94.67	94.21	95.68	1.0	648.1	1.4
Our unrestricted PrAE	98.82	97.50	99.22	1.1	15,408.5	2.2
NVSA backend	99.19	96.89	99.55	12.6	63.2	18.5

TABLE II: End-to-end model accuracy (%) on the RAVEN test set. The upper part of the table shows the results reported in the literature, while 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. The NVSA was either trained end-to-end (see equation (19)) or with the auxiliary visual attribute labels (see Supplementary equation (3)).

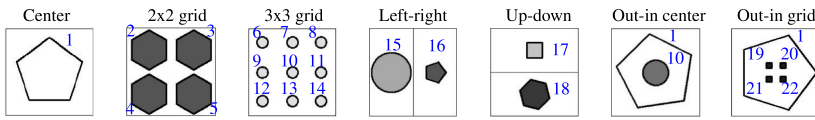| Method	Avg	Center	2x2 grid	3x3 grid	L-R	U-D	O-IC	O-IG
WReN[4]	14.7	13.1	28.6	28.3	7.5	6.3	8.4	10.6
ResNet[7]	53.4	52.8	41.8	44.3	58.8	60.2	63.2	53.1
ResNet+DRT[7]	59.6	58.1	46.5	50.4	65.8	67.1	69.1	60.1
Shah et al.[24]	67.5	94.6	53.1	33.9	85.0	89.1	89.8	31.9
LEN[5]	78.3	82.3	58.5	64.3	87.0	85.5	88.9	81.9
CoPINet[6]	91.4	95.1	77.5	78.9	99.1	99.7	98.5	91.4
DCNet[47]	93.6	97.8	81.7	86.65	99.8	99.8	99.0	91.5
PrAE[23]	$60.3^{\pm2.7}$	$70.6^{\pm5.3}$	$83.5^{\pm2.8}$	$30.5^{\pm1.7}$	$88.5^{\pm4.2}$	$89.2^{\pm4.1}$	$37.3^{\pm3.1}$	$22.7^{\pm2.2}$
MRNet[10]	$74.7^{\pm3.3}$	$96.2^{\pm4.3}$	$49.1^{\pm4.7}$	$45.9^{\pm6.0}$	$93.7^{\pm2.1}$	$94.2^{\pm2.1}$	$92.5^{\pm1.1}$	$51.3^{\pm11.5}$
SCL[44]	$87.2^{\pm0.9}$	$99.9^{\pm0.0}$	$76.6^{\pm0.8}$	$63.5^{\pm4.7}$	$96.8^{\pm5.6}$	$98.4^{\pm3.0}$	$96.5^{\pm3.6}$	$78.5^{\pm2.5}$
NVSA (end-to-end tr.)	$87.7^{\pm0.5}$	$99.7^{\pm0.4}$	$93.5^{\pm2.9}$	$57.1^{\pm3.3}$	$99.8^{\pm0.1}$	$99.7^{\pm0.2}$	$98.6^{\pm1.6}$	$65.4^{\pm0.6}$
NVSA (attribute label tr.)	$98.5^{\pm0.1}$	$100^{\pm0.0}$	$99.4^{\pm0.0}$	$96.3^{\pm0.6}$	$100^{\pm0.0}$	$100^{\pm0.0}$	$100^{\pm0.0}$	$93.9^{\pm0.0}$
Human[7]	84.4	95.5	81.8	79.6	86.4	81.8	86.4	81.8

TABLE III: End-to-end model accuracy (%) on the I-RAVEN test set. The upper part of the table shows the results reported in the literature, while 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. The NVSA was either trained end-to-end (see equation (19)) or with the auxiliary visual attribute labels (see Supplementary equation (3)).

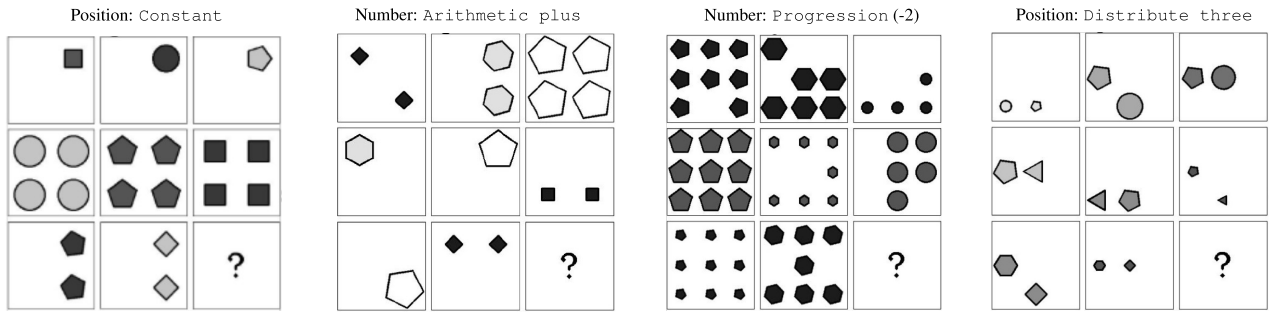| Method	Avg	Center	2x2 grid	3x3 grid	L-R	U-D	O-IC	O-IG
WReN[4]	23.8	29.4	26.8	23.5	21.9	21.4	22.5	21.5
ResNet[7]	40.3	44.7	29.3	27.9	51.2	47.4	46.2	35.8
ResNet+DRT[7]	40.4	46.5	28.8	27.3	50.1	49.8	46.0	34.2
SRAN[8]	60.8	78.2	50.1	42.4	70.1	70.3	68.2	46.3
LEN[5]	41.4	56.4	31.7	29.7	44.2	44.2	52.1	31.7
CoPINet[6]	46.1	54.4	36.8	31.9	51.9	52.5	52.2	42.8
DCNet[47]	49.36	57.8	34.1	35.5	58.5	60.0	57.0	42.9
PrAE[23]	$71.1^{\pm2.1}$	$83.8^{\pm3.4}$	$82.9^{\pm3.3}$	$47.4^{\pm3.2}$	$94.8^{\pm2.1}$	$94.8^{\pm2.1}$	$56.6^{\pm3.0}$	$37.4^{\pm1.7}$
MRNet[10]	$75.0^{\pm1.4}$	$96.8^{\pm3.7}$	$45.6^{\pm3.3}$	$39.6^{\pm1.8}$	$95.7^{\pm1.5}$	$95.9^{\pm1.8}$	$95.6^{\pm1.5}$	$55.5^{\pm4.7}$
SCL[44]	$84.3^{\pm1.1}$	$99.9^{\pm0.0}$	$68.9^{\pm1.9}$	$43.0^{\pm6.2}$	$98.5^{\pm2.9}$	$99.1^{\pm1.5}$	$97.7^{\pm1.3}$	$82.6^{\pm2.5}$
NVSA (end-to-end tr.)	$88.1^{\pm0.4}$	$99.8^{\pm0.2}$	$96.2^{\pm0.8}$	$54.3^{\pm3.2}$	$100^{\pm0.1}$	$99.9^{\pm0.1}$	$99.6^{\pm0.5}$	$67.1^{\pm0.4}$
NVSA (attribute label tr.)	$99.0^{\pm0.3}$	$100^{\pm0.0}$	$99.5^{\pm0.0}$	$97.1^{\pm1.8}$	$100^{\pm0.0}$	$100^{\pm0.0}$	$100^{\pm0.0}$	$96.4^{\pm0.0}$

SUPPLEMENTARY FIGURES

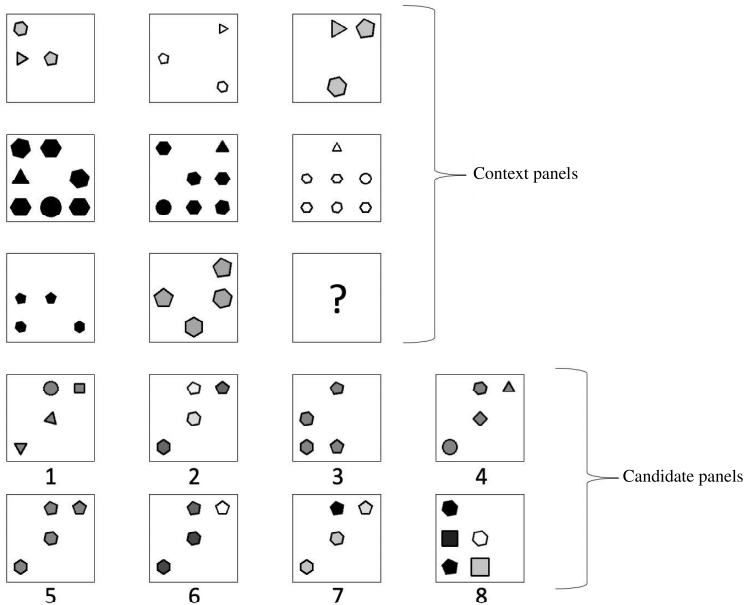**Supplementary Figure 1: Details on the RAVEN dataset.**

(a) Example for each constellation in RAVEN

(b) Example for each rule in RAVEN

(c) Example RPM test

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:

$$\theta^* = \underset{\theta}{\arg\max} \, \mathrm{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_1} \oplus ... \oplus \mathbf{w}_{y_k}\right) \tag{1}$$

$$\approx \underset{\theta}{\arg\max} \, \mathrm{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_1}\right) + ... + \mathrm{sim}\left(f_\theta(\mathbf{X}), \mathbf{w}_{y_k}\right). \tag{2}$$

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

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

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 $\mathbf{z} = \mathbf{Wq}$, 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[23]. 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[23], 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 $9\times$ 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[23] 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.

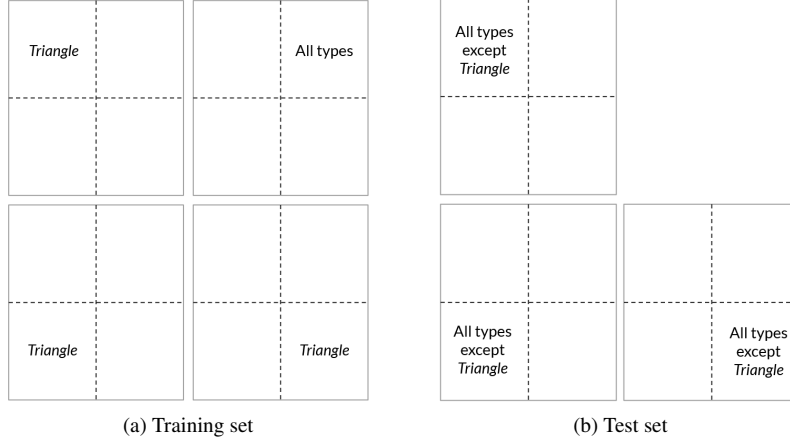† A universal model on all seven constellations was trained and tested.

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[56] 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 $\mathbf{z}$ 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 $\mathbf{z}$

(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[57]. 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 **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 encoding can generalize to unseen combinations of multiple objects. We train the NVSA frontend (hereafter, we simply omit the

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

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 $k_{train} \in \{1, 2\}$, where k_{train} is the number of available objects in the panel. In the training setting $k_{train} = 1$, we only train the NVSA frontend using the basis training set with a single object, whereas in the $k_{train} = 2$ 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 $k_{test} \in \{2, 3, 4\}$, where in each setting, the trained models are tested on 28,800 panels containing a fixed number of k_{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 k_{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 $k_{train} \in \{1, 2, 3, 4\}$, where in $k_{train} = 1$ we only use the basis training set to train the NVSA frontend. In the settings where $k_{train} \geq 2$, we augment the training set which is used in the setting $k_{train} - 1$ by 21,600 panels containing exactly k_{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 $k_{test} \in \{2, 3, 4, 5, 6, 7, 8, 9\}$, where in each setting, the trained models are tested on 64,800 panels containing a fixed number of k_{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 $k_{train} = 1$ setting, the NVSA frontend is already able to correctly predict the majority of the panels in the $k_{test} \in \{2, 3\}$ 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 $k_{test} = 4$, achieving 19.3%. However, training with the 2 object combinations (i.e., $k_{train} = 2$) achieves an average panel accuracy of 97.1% in all testing settings. Considering the out-of-distribution (OOD) testing cases, i.e., $k_{test} \in \{3, 4\}$, 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 $k_{train} = 1$ setting, where it correctly predicts 70.6% of the panels containing 2 objects. However, in the testing settings $k_{test} \in \{5, 6, 7, 8, 9\}$ the model is overwhelmed by the presence of too many objects at the same time. In $k_{train} = 2$, it is able to obtain non-zero panel accuracy in all testing settings except the most complex

one ($k_{\text{test}} = 9$). 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 $k_{\text{train}} \in \{1, 2, 3, 4\}$ and $k_{\text{test}} \in \{2, 3, 4, ..., 9\}$. 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., $\mathbf{o}_\beta = \mathrm{argmax}_{\mathbf{w} \in \mathbf{W}} \mathrm{sim}(\hat{\mathbf{o}}_\beta, \mathbf{w})$. 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

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

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

$$\mathbf{o}_\beta = \mathbf{o}_{\beta_1} \oplus ... \oplus \mathbf{o}_{\beta_k}. \tag{16}$$
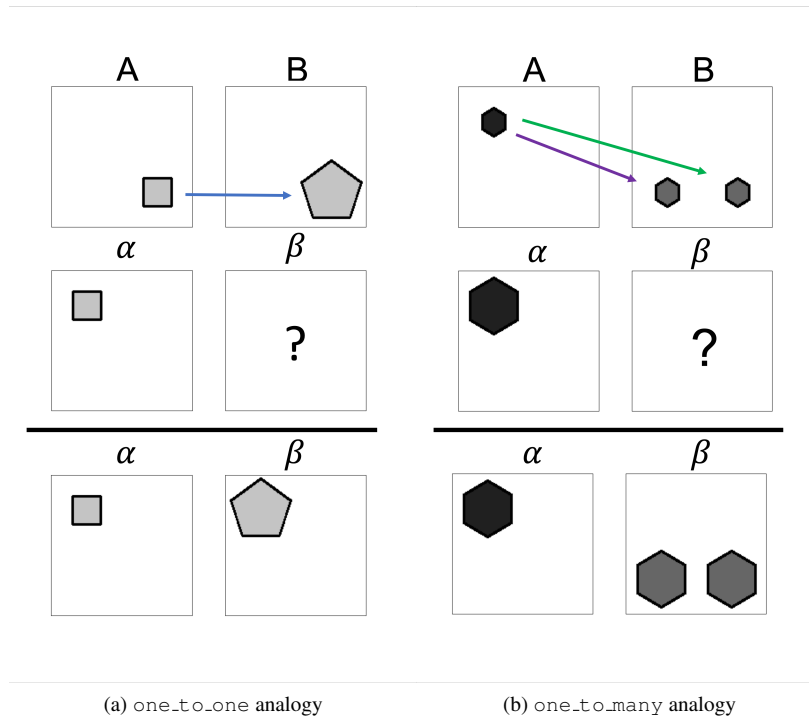
The trained ResNet-18 generates the perceptual representations for the three panels: $\mathbf{q}_A$, $\mathbf{q}_B$, $\mathbf{q}_\alpha$. Similar to Supplementary Note 2a, binding $\mathbf{q}_A$ and $\mathbf{q}_B$ captures the relationship set in $\mathbf{r}_{A:B}$. This is because, in VSA, multiplication (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:

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

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 $\mathbf{u}[\text{rule}]$ 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)[32]. 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[32], 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:
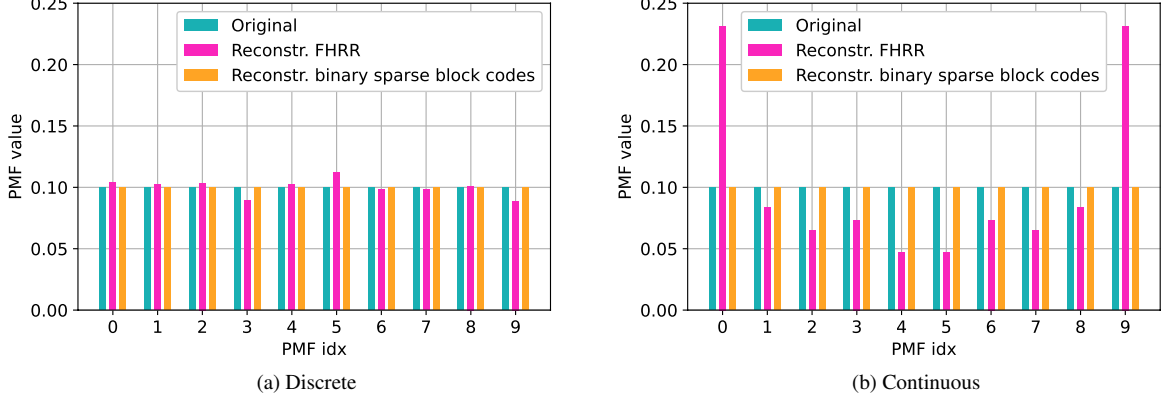
$$\mathbf{a}_{\text{FHRR}}^{(i,j)} := g_{\text{FHRR}}(\mathbf{p}^{(i,j)}) = \text{cnorm}\left(\sum_{k=1}^n \mathbf{p}^{(i,j)}[k] \cdot \mathbf{b}_k \right), \tag{20}$$

where $\text{cnorm}(\cdot)$ 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 $\mathrm{sim}(\mathbf{r}_i^+, \mathbf{a}^{(i,3)}) \gg 0$. Let us have a closer look at the similarity expression for the first row:

$$\mathrm{sim}(\mathbf{r}_i^+, \mathbf{a}^{(1,3)}) = \mathrm{sim}(\mathbf{a}^{(1,1)} \odot \mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) \tag{22}$$

$$= \mathrm{sim}\left(\left(\sum_{k_1=1}^n \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{b}_{k_1} \right) \odot \left(\sum_{k_2=1}^n \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{b}_{k_2} \right), \left(\sum_{k_3=1}^n \mathbf{p}^{(1,3)}[k_3] \cdot \mathbf{b}_{k_3} \right) \right) \tag{23}$$

$$= \sum_{\substack{k_1,k_2,k_3 \\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{p}^{(1,3)}[k_3] + \sum_{\substack{k_1,k_2,k_3 \\ s.t. k_1+k_2 \neq k_3}} \mathrm{sim}(\mathbf{b}_{k_1} \odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3}) \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{p}^{(1,3)}[k_3] \tag{24}$$

$$= \sum_{\substack{k_1,k_2,k_3 \\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{p}^{(1,3)}[k_3] + \sum_{\substack{k_1,k_2,k_3 \\ s.t. k_1+k_2 \neq k_3}} n_{k_1,k_2,k_3} \cdot \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{p}^{(1,3)}[k_3] \tag{25}$$

$$\approx \sum_{\substack{k_1,k_2,k_3 \\ s.t. k_1+k_2=k_3}} \mathbf{p}^{(1,1)}[k_1] \cdot \mathbf{p}^{(1,2)}[k_2] \cdot \mathbf{p}^{(1,3)}[k_3]. \tag{26}$$

Equation (24) uses the linearity of the similarity and divides the sum into contributions that satisfy the arithmetic plus constraint (LHS), i.e., $\mathrm{sim}(\mathbf{b}_{k_1} \odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3}) = 1$, and contributions which do not satisfy the constraint (RHS). For the latter, we replace the similarity $\mathrm{sim}(\mathbf{b}_{k_1} \odot \mathbf{b}_{k_2}, \mathbf{b}_{k_3})$ 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[23]. Supplementary Table IV shows the relation between our NVSA backend and the PrAE backend[23] 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 $\mathrm{sim}(\mathbf{r}_i^+, \mathbf{a}^{(i,3)})$ 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 $0 \leq h_a \leq 1$ 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

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

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 $\mathbf{b}_n^2$):

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

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

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

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.,

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

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

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

where

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

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

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

Supplementary Table IV: Relation of rule probability computation between PrAE[23] and our NVSA backend.

Rule	PrAE	NVSA
Constant	$u = \sum_{r=1}^{2} \sum_{v=1}^{n} \prod_{c=1}^{3} \mathbf{p}^{(r,c)}[v] + \sum_{v=1}^{n} \prod_{c=1}^{2} \mathbf{p}^{(3,c)}[v]$	$u = \text{sim}(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}) \cdot \text{sim}(\mathbf{a}^{(1,2)}, \mathbf{a}^{(1,3)}) \cdot ... \cdot \text{sim}(\mathbf{a}^{(3,1)}, \mathbf{a}^{(3,2)})$
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	$u = \sum_{v^{(1,1)},...,v^{(3,2)} \in I_{d3}} \prod_{r=1}^{2} \prod_{c=1}^{3} \mathbf{p}^{(r,c)}[v^{(r,c)}] \prod_{c=1}^{2} \mathbf{p}^{(3,c)}[v^{(3,c)}]$	$u = \text{sim}(\mathbf{c}_1, \mathbf{c}_2) \cdot \text{sim}(\mathbf{r}_1, \mathbf{r}_2) \cdot h_d(\mathbf{a}^{(1,1)}, \mathbf{a}^{(1,2)}), ..., \mathbf{a}^{(2,3)})$

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.*[44]. 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[58].

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., $\mathbf{v}_{\text{exist}}^{(k)}$, $\mathbf{v}_{\text{type}}^{(k)}$, $\mathbf{v}_{\text{size}}^{(k)}$, $\mathbf{v}_{\text{color}}^{(k)}$ 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

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

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:

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

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[58]	33.0
LSTM[58]	35.8
Resnet-50[58]	42.0
Wild-ResNet[58]	48.0
CoPINet[6]	56.4
WReN[58]	62.8
LEN[5]	88.9
SCL[44]	88.9
MRNet[10]	93.4
PrAE[23]	N/A*
LEN[5]	N/A†
SCL[44]	N/A†
MRNet[10]	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:

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

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[44] (88.9%), and MRNet[10] (93.4%). The lower part of the table compares the accuracy of the reproduced methods with our NVSA. PrAE[23] 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[44] can only be applied to RAVEN, even though PGM results are reported in the corresponding works. Finally, MRNet[10], 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.