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

Michael Hersche1,2, Mustafa Zeqiri2, Luca Benini2, Abu Sebastian1,2 & Abbas Rahimi1,2

Neither deep neural networks nor symbolic artificial intelligence (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 Raven’s progressive matrices datasets. Compared with 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 with 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.

Human fluid intelligence is the ability to think and reason abstractly, and make inferences in a novel domain. The Raven’s progressive matrices (RPM) test has been a widely used assessment of fluid intelligence and abstract reasoning. The RPM is a non-verbal test that involves perceiving pattern continuation and elemental abstraction, and finding relations between abstract elements on the basis of 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 on the basis of 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 (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 the two interact varies greatly across the spectrum of artificial intelligence (AI) architectures. At one end of the spectrum, in deep learning, perception has primacy and reasoning is more likely to adapt to the representations than vice versa. At the other end of the spectrum, in classical symbolic AI, the perceptual representations are pre-engineered to, for example, emphasize relations, rather than 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. Neuro-symbolic architectures further enrich the spectrum of possibilities by utilizing techniques from both ends: they combine subsymbolic (for example, neural network) with symbolic AI approaches, aiming to reach human-level generalization. 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, for example, visual question answering, causal video reasoning and solving RPM. However, the resulting neuro-symbolic architectures are not immune to the potential problems of their...
individual ingredients (that is, the neuro and symbolic parts), which are explained in the following.

We first explain the binding problem 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. When we consider the fully local representations, an item of any complexity level can be represented by a single unit, for example, 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, distributed representations can provide enough capacity to represent a combinatorially growing number of compositional items. However, they face another issue known as the ‘superposition catastrophe’. Let us consider four atomic items: red, blue, square, and triangle. For representing two composite objects, for example, 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 patterns of all four atomic items are co-activated, which causes ‘ghost’ or ‘spurious’ memory for unavailable objects such as red triangle or blue square. As a common practice, to bypass this problem, neuro-symbolic architectures employ two complementary networks to unambiguously extract the item attributes from multiple objects in an image that results in an increase in the number of weights. The second ingredient of neuro-symbolic architectures is symbolic reasoning. For solving the RPM tests, as an upgrade of pure logical reasoning, it is implemented as a probabilistic abduction reasoning that can be viewed as searching for a solution in a space defined by prior background knowledge, which is represented in symbolic form by describing all possible rule realizations that could govern the RPM tests. The reasoning process can then be executed via an exhaustive search over these symbols to abduce the probability distribution of the rules. However, the computational complexity of the exhaustive search rapidly increases with the number of objects in the RPM panels. In effect, this computationally expensive search hinders its utilization for end-to-end training and real-time inference.

We aim to address the two aforementioned problems by exploiting vector-symbolic architectures (VSAs)\textsuperscript{11–18}. 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. In a VSA, all representations—from atomic to composite structures—are high-dimensional distributed 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 (that is, additive superposition), permutations and associative memory. The compositionality and transparency enabled the use of VSAs in inductive logical reasoning\textsuperscript{14–18}, but these inspiring works lacked a perception module to process the raw sensory inputs. Instead, they assumed a perception system, for example, a symbolic parser, to provide the symbolic representations that support the reasoning. Further, their inductive logical reasoning has eluded the uncertainty in perception, and supported only one type of rule in RPM: the progression rule.\textsuperscript{13,18}

In this Article, we propose a neuro-vector-symbolic architecture (NVSA) in which two powerful machineries, namely deep neural networks and VSAs, are synergistically combined to co-design 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 non-linear transformations of a neural network as a flexible means of representation learning over the dictionary should be able to provide such appropriate representations by expressing attributes per objects, which, in turn, fulfills the reasoning demand. For the NVSA frontend, we use VSA whose vector entries are restricted to being dense bipolar. The dictionary can be constructed by the application of the VSA operations to 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, for example, one codebook for the colours (red, blue) and another codebook for the shapes (square, 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 \(w\) can be computed by binding (\(\odot\)) two vectors, one drawn from each of the codebooks (Fig. 1b). For example, a red square object is represented as \(w = x_{\text{red}} \odot x_{\text{square}}\). The random construction of the atomic vectors and the properties of multiplicative binding yield a unique \(w\) that is quasi-orthogonal (that is, dissimilar) to the VSA representations of all other attributes and their combinations (for example, \(x_{\text{blue}} \odot x_{\text{triangle}}\) or \(x_{\text{blue}} \odot x_{\text{square}}\) or \(x_{\text{red}} \odot x_{\text{triangle}}\)). 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 co-activated, it results in minimal interference such that each object can be recovered (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 (\(\oplus\)) together their object vectors: \(s = (x_{\text{red}} \odot x_{\text{square}}) \oplus (x_{\text{blue}} \odot x_{\text{triangle}})\). The bundling operation creates an equally weighted superposition of multiple objects, and preserves similarity; hence, the bundled vector \(s\) 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

**NVSA frontend: perception**

The NVSA frontend consists of a trainable neural network and a VSA whose architecture allows the construction of appropriate perceptual representations suitable for reasoning demands. The frontend design 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, for example, 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 to transduce the raw image of the scene to the VSA representations (Fig. 2b). This merger of the data-driven representation of the scene with its descriptive symbolic representation facilitates downstream reasoning tasks, without exploding the representation dimensionality, or facing the superposition catastrophe. In the following, we describe the steps involved in the NVSA frontend.

**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 reasoning demands. For instance, when solving RPMs, the reasoning process requires the PMFs 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. For the NVSA frontend, we use VSA whose vector entries are restricted to being dense bipolar. The dictionary can be constructed by the application of the VSA operations to 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, for example, one codebook for the colours (red, blue) and another codebook for the shapes (square, 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 \(w\) can be computed by binding (\(\odot\)) two vectors, one drawn from each of the codebooks (Fig. 1b). For example, a red square object is represented as \(w = x_{\text{red}} \odot x_{\text{square}}\). The random construction of the atomic vectors and the properties of multiplicative binding yield a unique \(w\) that is quasi-orthogonal (that is, dissimilar) to the VSA representations of all other attributes and their combinations (for example, \(x_{\text{blue}} \odot x_{\text{triangle}}\) or \(x_{\text{blue}} \odot x_{\text{square}}\) or \(x_{\text{red}} \odot x_{\text{triangle}}\)). 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 co-activated, it results in minimal interference such that each object can be recovered (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 (\(\oplus\)) together their object vectors: \(s = (x_{\text{red}} \odot x_{\text{square}}) \oplus (x_{\text{blue}} \odot x_{\text{triangle}})\). The bundling operation creates an equally weighted superposition of multiple objects, and preserves similarity; hence, the bundled vector \(s\) 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 constituents’ object vectors, which avoids the superposition catastrophe by design (for example, $\text{sim}(s, x_{\text{red}} \otimes x_{\text{triangle}}) \approx 0$). 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 two-attributes objects to arbitrary number of attributes suitable for solving RPM (an RPM test example is shown in Fig. 2a). Similarly, we randomly generate a set of compact codebooks for each available attribute. Using these codebooks, a quasi-orthogonal vector for every possible combination of a single object as the Hadamard product of its attribute vectors is computed and stored in a dictionary $\mathbf{W}$ in $[-1, +1]^{w_d}$ containing all $m$ possible single object combinations. The dictionary $\mathbf{W}$ is generated once, on the basis of the initialization of codebook vectors, and is kept frozen during training. For more details on the application to the RAVEN dataset, see Methods. 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.

**Neural network representation learning over VSA**

To avoid the pitfalls of pure symbolic approaches, 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 to the corresponding VSA representations using a mapping $f_\theta$ with learnable parameters $\theta$. To do so, we propose using a ResNet-18, motivated by its good performance, 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, 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 requiring at least two separate neural networks (for example, a Mask R-CNN followed by a ResNet-34) that increase the trainable parameters by $\cdot 6 \times$ compared with our NVSA frontend.

In a fully supervised setting in which the labels of the visual attributes are given, the NVSA frontend can be trained independently of the backend. For an image panel $\mathbf{X}$ containing $k$ objects with $k$ target indices $\{y_i\}_{i=1}^k$, the optimization maximizes the similarity between the output query $q = f_\theta(\mathbf{X})$ of ResNet-18 and the bundled vector $w_{y_1} \oplus ... \oplus w_{y_k}$ using a novel additive cross-entropy loss together with batched gradient descent. We provide the details of this loss, and show its superiority compared over other loss functions and perception methods (Supplementary Note 1a). We also analyze the generalization of the NVSA frontend to unseen combinations of attribute values in a novel object. 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% (Supplementary Note 1b). The multiplicative binding-based encoding, however, generalizes well to unseen combinations of multiple objects (Supplementary Note 1c). It also results in learning powerful perceptual representations in readiness for solving visual analogy tasks (Supplementary Note 2).

On the basis of the cosine similarities between a query $\mathbf{q}$ and all elements in the dictionary $\mathbf{W}$, we derive the probabilistic scene representation containing five PMFs of each PMF that represent the position, number, type, size and colour distribution of the entire panel. These PMFs...
are denoted by $P = \left( p_{\text{num}}, p_{\text{type}}, p_{\text{size}}, p_{\text{colour}} \right)$ (Methods). 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 $P^{0}$ 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 $P^{3,3}$ for the missing panel.

**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 solving larger sized problems compared with the 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 allows the application of VSA operators to implement the rules such as addition of the attribute values, or subtraction, distribution and more (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.

**VSA representations of PMFs**

The RAVEN dataset applies an individual rule to each of the five attributes (position, number, colour, size and type), which is either constant, progression, arithmetic or distribute three (Methods). The rules are applied row-wise across the context matrix. On the basis of 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 colour 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 the NVSA backend, we switch from the previously used bipolar dense representations to binary sparse block codes (5, 6). This VSA framework, with the help of fractional power encoding (3), permits the representation of continuous PMFs. For more details on binary sparse block codes and how to create the codebooks for discrete and continuous attributes, see Methods.

Each PMF is represented by the weighted superposition whereby the values ($p^{i,j}[k]$) in the PMF are used as weights and the corresponding codewords ($b_k$) as basis vectors (Fig. 3a): $a^{i,j} := g(p^{i,j}) = \sum_{k=1}^{n} p^{i,j}[k] \cdot b_k$. (1)

Every attribute PMF is transformed separately into its corresponding VSA representation, for example, the PMF of the attribute number is transformed to $a_{\text{num}}^{i,j} := g(p_{\text{num}}^{i,j})$. (Methods)

**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 $r_i$ vectors for the first and second row using the first two panels:

$$r_i = a^{i,1} \odot a^{i,2} \odot a^{i,3}, \quad i \in \{1, 2\}. \quad (2)$$

To better understand equation (2), let us assume that the distribution of the PMFs of the context panels is maximally compact, that is, the values in $p_{\text{num}}^{i,j}$ are ‘1’ at the correct number of objects and ‘0’ elsewhere. Then, the bound vector of the first row can be formulated as $r_1^+ = e_{p}^1 \odot e_{p}^2 = e_{p}^{1+2}$, where $e_{p}^1$ and $e_{p}^2$ are the number of objects in the first and second panel. If the rule applies, that is $e_{p}^1 = e_{p}^1 + e_{p}^2$, we expect the bound vector $r_1^+$ to be identical to the VSA representation of the last panel in the row $a^{3,3} = e_{p}^3 = e_{p}^{1+2}$, thanks to the properties of fractional power encoding.

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

$$u[\text{arithmetic plus}] = \sim(r_1^+, a^{1,3}) \cdot \sim(r_2^+, a^{2,3}) \cdot h_a(a^{1,3}, a^{2,3}), \quad (3)$$

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

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

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 that 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:

$$\hat{p}_{\text{num}}^{3,3} = \text{norm}\left([\sim(\hat{a}^{1,3}, b_1), \sim(\hat{a}^{1,3}, b_2), ..., \sim(\hat{a}^{1,3}, b_n)]\right). \quad (5)$$

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 $3 \times 3$ grid (Fig. 3c). The position is described with a 9-bit code where a ‘1’ indicates that the position is occupied inside the $3 \times 3$ 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:

$$r_i = a^{i,1} \odot a^{i,2} \odot a^{i,3}, \quad i \in \{1, 2\}. \quad (6)$$

$$c_{ij} = a^{i,j} \odot a^{j,i} \odot a^{i,j}, \quad i, j \in \{1, 2\}. \quad (7)$$

Equations (6) and (7) describe a VSA-based conjunctive formula grounded over the row and column being considered, respectively.
Solving the arithmetic plus rule on number attribute with vector-symbolic reasoning

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, Illustration of solving the arithmetic plus rule. The PMF of each panel \( p(i,j) \) 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 \( r_1 \) and \( r_2 \). The rule probability is computed by multiplying the similarities between the row vectors \( r \) and the last panel of the first two rows, and an additional constraint \( h_u \). 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 search computes the similarities between the predicted vector and all atomic vectors to determine the PMF \( \hat{p}(3,3) \).

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 \( r_1 \) and \( r_2 \). The rule probability is the product of the similarity between the two row vectors and the similarity between the first two column representations \( c_1 \) and \( c_2 \). 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, for example, \( r_2 \).
For example, given a set of arbitrary PMFs in a row, the resulting product vector \( \mathbf{v} \) 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 \texttt{distribute three} rule applies by simply checking if the product vectors are similar among rows and columns, that is, \( \sim(\mathbf{r}_1, \mathbf{r}_2) \gg 0 \) and \( \sim(\mathbf{c}_1, \mathbf{c}_2) \gg 0 \), and combine them together to estimate the rule probability

\[
\begin{align*}
\mathbf{u}[\texttt{distribute three}] &= \sim(\mathbf{r}_1, \mathbf{r}_2), \\
&\quad \sim(\mathbf{c}_1, \mathbf{c}_2) \cdot \mathbf{h}_d(\mathbf{a}^{1,1}, \mathbf{a}^{1,2}, \ldots, \mathbf{a}^{1,3}), \\
&\sim(\mathbf{r}_1, \mathbf{r}_2) \cdot \mathbf{h}_d(\mathbf{a}^{2,1}, \mathbf{a}^{2,2}, \ldots, \mathbf{a}^{2,3}), \quad \text{(8)}
\end{align*}
\]

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

The associative memory search is limited to only the \( n \) atomic vectors in the codebook \( \mathbf{B} \); hence, our NVSA backend requires \( O(n) \) in time and space. This is a substantial reduction compared with pure symbolic search-based reasoning approaches, which search exhaustively through all possible rule implementations demanding up to \( O(n^3) \) in time and space. For example, the previously described \texttt{distribute three} rule on the attribute position would have \( n = 511 \cdot 12 \geq 6 \cdot 10^7 \) different rule implementations in the \( 3 \times 3 \) grid constellation which is prohibitive to compute. This exhaustive rule search forces the neuro-symbolic approach by Zhang et al.\(^{29}\) 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 NVSA is mainly due to performing search-in-superposition. For example, by comparing the VSA representations of the first and second row, we can sum up all possible combinations in superposition:

\[
\begin{align*}
\sim(\mathbf{r}_1, \mathbf{r}_2) &= \sim(a^{1,1} \odot a^{1,2} \odot a^{1,3}, a^{2,1} \odot a^{2,2} \odot a^{2,3}) \\
&= \sim \left( \left( \sum_{k=1}^{n} p^{1,1}(k) \cdot b_k \right) \odot \left( \sum_{k=1}^{n} p^{1,2}(k) \cdot b_k \right) \right) \odot \left( \sum_{k=1}^{n} p^{1,3}(k) \cdot b_k \right) \odot a^{2,1} \odot a^{2,2} \odot a^{2,3}. \quad \text{(9)}
\end{align*}
\]

Without the VSA representations, one would need to compute the maximally expanded version.

For details about our implementation of the arithmetic minus, the progression and the constant rule, see Supplementary Note 3.

Finally, we train our NVSA frontend and backend end-to-end by mutually minimizing a REINFORCE\(^{23}\) and an additional auxiliary loss (Methods). Note that only the neural part of the frontend (that is, ResNet-18) has trainable parameters, while the dictionary (\( \mathbf{W} \)) and all the parameters in the backend (for example, rule representations) are frozen.

Results

We evaluate NVSA on the RAVEN\(^7\), I-RAVEN\(^6\) and PGM\(^4\) datasets (Methods), and compare it with the state-of-the-art models in both deep neural networks (MRNet\(^4\) and scattering compositional learner (SCL)\(^{48}\)) and neuro-symbolic AI (PrAE\(^{29}\)). First, we consider the more diverse RAVEN and I-RAVEN datasets. We retrained the state-of-the-art 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. Table 1 compares the classification accuracy on the RAVEN dataset. NVSA achieves an average accuracy of 87.7%, outperforming MRNet by 13%, SCL by 0.5% and PrAE by 27.4%.

As described in Methods, there is a shortcut solution when employing the RAVEN dataset related to its answer choices. Therefore, it is recommended to use the I-RAVEN dataset, which provides unbiased fair answer panels\(^{28,43}\). NVSA achieves the highest accuracy on the I-RAVEN dataset too (88.1%) on average, while the majority of deep learning approaches\(^{4,24,44}\) face a large accuracy drop, showing <50% accuracy on average. Our NVSA does not exhibit an accuracy drop when switching from RAVEN to I-RAVEN because it cannot rely on such a shortcut by design: on the basis of the most probable rules, it predicts the PMFs of the empty panel before individually comparing them with each PMF of the answer panel. The controllability and explainability of NVSA is a great advantage for problems that require them. NVSA also substantially outperforms MRNet by 13.1%, SCL by 4.2% and PrAE by 17.0% on I-RAVEN, on average (Table 2).

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 3, the PrAE reasoning backend reaches relatively lower accuracies (94.21–95.68%) in the \( 2 \times 2 \) grid, the \( 3 \times 3 \) grid and the out-in grid compared with 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 \( 3 \times 3 \) grid, requiring 15,408 min (10.7 days) instead of the previous 648 min (10.8 h) in the PrAE with restricted search for solving 2,000 RPM tests. Note that we run the experiments on the central processing units (CPUs) as the unrestricted PrAE demands more than 53 GB memory that could not fit the graphics processing unit (GPU) providing 32 GB memory (Methods). However, our NVSA reasoning backend effectively resolves this bottleneck: it reduces the computation time on the \( 3 \times 3 \) grid to 63.2 min, which is 244× faster than the unrestricted PrAE, and the memory demand to <10 GB, while maintaining the high accuracy (96.89% versus 97.50%). Moreover, we demonstrate that the frontend and backend of our NVSA can be trained end-to-end, in practically any constellation, and it provides real-time inference for solving the RPM tests (Supplementary Video 1).

We also evaluate NVSA on the PGM dataset\(^4\), as the first neuro-symbolic approach targeting this dataset, while other neuro-symbolic works\(^{21,23,24}\) targeted only RAVEN/I-RAVEN. NVSA achieves an average accuracy of 68.3% and is highly competitive with the reproduced state-of-the-art MRNet (Supplementary Note 5).

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 when it has been trained on examples containing all of the attribute–rule pairs except the specific target one. Our NVSA outperforms the baselines (LEN\(^1\) and CoPiNet\(^1\)) by a large margin in all unseen attribute–rule pairs (Supplementary Note 4).

Discussion

The NVSA frontend allows expression of many more object combinations than dimensions in the vector space. However, it requires one 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 a dynamical system based on the VSA operators. A powerful example of this would be the resonator networks\(^4,49\) and their stochastic non-linear variants\(^{29}\) 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 this essential search, one could resort to analogue in-memory computing. It has recently been shown that associative memory can be realized using analogue in-memory computing based on crossbar arrays of emerging non-volatile memories. Besides improving the computational density and energy efficiency, this paves the way for reducing the computational time complexity of associative memory search to \( \mathcal{O}(1) \). Other frequent computational primitives such as binding and bundling can also benefit from low-power hardware realization.

By accurately and efficiently solving the RPM tests, we have demonstrated that NVSA enhances the aspects of both perception and reasoning by adding a distinctive flavour to them that 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 substantially lower than the combinatorial attributes. These structured representations were used as target vectors to train the deep neural network. When the attribute labels are available, the training can be done by the additive cross-entropy loss; otherwise it can be done end-to-end. The resulting deep representation learning over VSA permitted the simultaneous inference of multiple attributes of multiple objects that obviated exploding the representation dimensionality or facing the superposition catastrophe. In the NVSA backend, we proposed a computationally efficient and differentiable reasoning where the PMFs of discrete or continuous attributes are expressed as the VSA representations. This permitted the use of VSA operators to efficiently implement the rules that substantially save the computational cost thanks to the distributivity and computing-in-superposition of VSA. As a result, the time/space computational cost grows linearly with the number of objects and attributes.

### Table 1 | Model accuracy (%) on the RAVEN test set

| Method          | Average | Centre | 2×2 grid | 3×3 grid | L-R | U-D | O-IC | O-IG |
|-----------------|---------|--------|----------|----------|-----|-----|------|------|
| WReN4           | 14.7    | 13.1   | 28.6     | 28.3     | 7.5 | 6.3 | 8.4  | 10.6 |
| ResNet7         | 53.4    | 52.8   | 41.8     | 44.3     | 58.8| 60.2| 63.2 | 53.1 |
| ResNet+DRT7     | 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 |
| LENA²           | 78.3    | 82.3   | 58.5     | 64.3     | 87.0| 85.5| 88.9 | 81.9 |
| CoPINet8        | 91.4    | 95.1   | 77.5     | 78.9     | 99.1| 99.7| 98.5 | 91.4 |
| DCNet44         | 93.6    | 97.8   | 81.7     | 86.65    | 99.8| 99.8| 99.0 | 91.5 |
| PrAE²³          | 60.9²¹ | 70.6³³ | 83.6³⁴ | 30.5³² | 88.5³⁵ | 89.2³⁴ | 37.3³³ | 22.9³² |
| MRNet²⁵         | 74.7³¹³ | 96.2⁴²³ | 49.1⁴⁷ | 45.9⁶⁰ | 93.7²¹ | 94.2⁵¹ | 92.5⁴¹ | 51.3³¹³ |
| SCL³             | 87.2⁰³⁰ | 99.9⁰⁰⁰ | 76.6⁰⁸ | 63.5⁴⁷ | 96.8⁰⁶ | 98.4⁴³⁰ | 96.5³⁶⁵ | 78.5²⁵³ |
| NVSA (end-to-end tr.) | 87.³⁰⁵ | 99.³⁰⁴ | 93.⁵³⁴ | 57.³³³ | 99.³⁰¹ | 99.³⁰² | 98.³⁰⁶ | 65.³⁰⁶ |
| NVSA (attribute label tr.) | 98.³⁰⁰ | 100³⁰⁰ | 99.⁴⁰⁰ | 96.³⁰⁶ | 100³⁰⁰ | 100³⁰⁰ | 100³⁰⁰ | 93.³⁰⁰ |
| Human⁷          | 84.⁴    | 95.⁵   | 81.⁸    | 79.⁶    | 86.⁴ | 81.⁸ | 86.⁴ | 81.⁸ |

The upper part of the table shows the results reported in the literature, while in the lower part, we report the average accuracy ± standard deviation over five runs with different seeds for our NVSA and the reproduced state-of-the-art deep neural networks (MRNet and SCL) and neuro-symbolic (PrAE) approaches. The NVSA was either trained end-to-end (equation (19)) or with the auxiliary visual attribute labels (supplementary equation (3)).

### Table 2 | Model accuracy (%) on the I-RAVEN test set as a revised unbiased version of the RAVEN dataset

| Method          | Average | Centre | 2×2 grid | 3×3 grid | L-R | U-D | O-IC | O-IG |
|-----------------|---------|--------|----------|----------|-----|-----|------|------|
| WReN4           | 23.8    | 29.4   | 26.8     | 23.5     | 21.9| 21.4| 22.5 | 21.5 |
| ResNet7         | 40.3    | 44.7   | 29.3     | 27.9     | 51.2| 47.4| 46.2 | 35.8 |
| ResNet+DRT7     | 40.4    | 46.5   | 28.8     | 27.3     | 50.1| 49.8| 46.0 | 34.2 |
| SRAN⁴           | 60.8    | 78.2   | 50.1     | 42.4     | 70.1| 70.3| 68.2 | 46.3 |
| LENA³           | 41.4    | 56.4   | 31.7     | 29.7     | 44.2| 44.2| 52.1 | 31.7 |
| CoPINet⁶        | 46.1    | 54.4   | 36.8     | 31.9     | 51.9| 52.5| 52.2 | 42.8 |
| DCNet⁴⁴         | 49.3⁶   | 57.8   | 34.1     | 35.5     | 58.5| 60.0| 57.0 | 42.9 |
| PrAE²³          | 71.²⁰⁷ | 83.³⁶⁴ | 82.²³³ | 47.⁴³³ | 94.⁸²³ | 94.⁸³³ | 56.³²³ | 37.⁴³² |
| MRNet²⁵         | 75.⁰³⁴ | 96.⁸⁵⁷ | 45.⁶³³ | 39.⁶³⁸ | 95.⁷³⁵ | 95.⁹³⁸ | 95.⁶³⁵ | 55.⁴³⁷ |
| SCL³             | 84.³⁰¹ | 99.⁵⁰⁰ | 68.⁹¹⁹ | 43.⁰³³ | 98.⁵³⁵ | 99.¹³³ | 97.³³³ | 82.³²³ |
| NVSA (end-to-end tr.) | 88.³⁰⁴ | 99.⁸⁰² | 96.²⁰⁸ | 54.³³² | 100³⁰³ | 99.⁹³¹ | 99.⁶³³ | 67.¹³³ |
| NVSA (attribute label tr.) | 99.⁵⁰³ | 100³⁰⁰ | 99.⁵⁰⁰ | 97.¹³⁸ | 100³⁰⁰ | 100³⁰⁰ | 100³⁰⁰ | 96.⁴³⁰ |

The upper part of the table shows the results reported in the literature, while in the lower part, we report the average accuracy ± the standard deviation over five runs with different seeds for our NVSA and the reproduced state-of-the-art approaches. The NVSA was either trained end-to-end (equation (19)) or with the auxiliary visual attribute labels (supplementary equation (3)).
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. In fact, it has been shown that the shortcut pattern can be leveraged by deep neural networks; for example, CoPINet achieved a higher accuracy when trained and tested exclusively on the answer panels without considering the context panels at all (context blind). To this end, the impartial RAVEN (I-RAVEN) 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.

**Methods**

**RAVEN and I-RAVEN dataset containing RPM tests**

The RAVEN dataset consists of a rich set of RPM tests. Every RPM test consists of an incomplete $3 \times 3$ 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 centre, $2 \times 2$ grid, $3 \times 3$ grid, left–right, up–down, out–in centre 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 to 6) and ten colors in the form of shadings (enumerated from 1 to 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, for example, the $2 \times 2$ grid contains maximally four objects. The position attribute describes the occupancy of the objects inside the panel. Its range is constellation dependent too; for example, the $2 \times 2$ grid has 15 different position constellations. Nine panels are arranged to a $3 \times 3$ 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.

| Method | Accuracy (%) | CPU compute time (min) |
|--------|-------------|------------------------|
| PrAE   | 94.67       | 94.21                  |
|        | 95.68       | 1.0                    |
|        | 1.4         | 648.1                  |
| O-IG   | 98.82       | 97.50                  |
|        | 99.22       | 11                     |
|        | 15,408.5    | 2.2                    |

NVSa backend

| Method | Accuracy (%) | CPU compute time (min) |
|--------|-------------|------------------------|
|        | 99.19       | 96.89                  |
|        | 99.55       | 12.6                   |
|        | 63.2        | 18.5                   |

Experiments were conducted with Intel Xeon E5-2640 cores running at 2.4 GHz. The NVSA backend was configured with a vector dimension $d = 1,024$ and $k = 4$ blocks.

**Table 3**

| Method | Accuracy (%) | CPU compute time (min) |
|--------|-------------|------------------------|
| 2×2    | 98.82       | 97.50                  |
| 3×3    | 94.67       | 94.21                  |
| O-IG   | 95.68       | 1.0                    |
|        | 648.1       | 1.4                    |

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. In fact, it has been shown that the shortcut pattern can be leveraged by deep neural networks; for example, CoPINet achieved a higher accuracy when trained and tested exclusively on the answer panels without considering the context panels at all (context blind). To this end, the impartial RAVEN (I-RAVEN) 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.

**VSA background**

VSA is a family of similar systems that represent data by using random high-dimensional vectors. There are different VSA variants (for a review, see refs. 54,55). In this work, the NVSA frontend uses the dense bipolar representation while the NVSA backend uses binary sparse block codes. We begin with the explanation of VSAs whose vector entries are restricted to being dense bipolar. Initially, one or multiple codebooks are defined as $X = \{x_i\}_{i=1}^{m}$, where the elements of each atomic $d$-dimensional vector $x_i \in \{-1, +1\}$ are randomly drawn from a Rademacher distribution (that is, 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; 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 ($\otimes$), 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 $x_i \in X$ is its own inverse with respect to the binding operation, that is, $x_i \odot x_i = 1$, where $I$ is the $d$-dimensional all-1 vector. Hence, the individual factors in a product can be exactly retrieved by unbinding: $x_i \odot (x_i \odot x_j) = (x_i \odot x_i) \odot x_j = x_j$.

As an alternative VSA representation, binary sparse block codes use $d$-dimensional, binary-valued basis vectors with $k$ non-zero elements. More specifically, the vectors are divided into $k$ distinct blocks that 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 inner product normalized by the number of non-zero elements. The bundling of two or more vectors is computed through the element-wise addition. Optionally, the bundled vector could be sparsified to have only one non-zero element per block again. Since this results in loss of information, the bundling is performed without sparsification in this work.

**Codebook generation in the NVSA frontend**

We randomly generate a set of compact codebooks for the available attributes in the RAVEN dataset as $T := \{t_{l}^{\text{in}}\}, S := \{s_{l}^{\text{in}}\}, C := \{c_{l}^{\text{in}}\}$ and $L := \{l_{l}^{\text{in}}\}$, which respectively represent the type, size, color and position of a single object, considering the cross-configuration equivalent positions with the same proportions (Supplementary Fig. 1). We set $d = 512$, which 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 ($m = 6,600$) 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 (that is, a four-way multiplicative binding). These $d$-dimensional vectors are stored in a dictionary $W \in \{-1, +1\}^m$ that 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, for example, an RPM panel as the union of its distinct objects. The dictionary $W$ is generated once, on the basis of the initialization of codebook vectors, and is kept frozen during training.

**Inferring PMFs for object attributes and panel attributes from VSA representations**

We describe the last step of the frontend here. Given an RPM panel, our trained ResNet-18 generates a VSA query vector that can be decomposed into the constituent object vectors, each derived from a unique combination of attributes. The decomposition performs a matrix–vector multiplication between the normalized dictionary matrix and the normalized query vector, to obtain the cosine similarity scores. Since the structure of the dictionary matrix is known, we can infer the attributes position, colour, size and type from the detected indices. On the basis of the similarity scores, we derive the PMFs for every object attribute. After inferring these PMFs of the object attributes, we infer the PMFs of the panel attributes. We combine all object PMFs to five PMFs that represent the position, number, type, size and colour distribution of the entire panel. These PMFs are denoted by $P = (p_{pos}, p_{num}, p_{type}, p_{size}, p_{colour})$.

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

$$v_{\text{type}}^{(k)}[j] = \sum_{s \in \{1, \ldots, 6\}} \sum_{c \in \{1, \ldots, 10\}} \text{ReLU}(\text{sim}(q, l_k \circ t_j \circ s_c \circ c)),$$

where $\text{ReLU}(\cdot)$ is the rectified linear unit activation. Similarly, the sum is computed for the object PMF of attribute colour and size. For marginalizing for object existence ($v_{\text{exist}}^{(k)}[0]$), 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 $(f_k)$, which indicates that no object is present at position $k$:

$$v_{\text{exist}}^{(k)}[0] = \sum_{t \in \{1, \ldots, 5\}} \sum_{s \in \{1, \ldots, 6\}} \sum_{c \in \{1, \ldots, 10\}} \text{sim}(q, l_k \circ t \circ s_c \circ c).$$

In this case, we marginalize over all cosine similarities (that is, no $\text{ReLU}$ activation is applied) as the $v_{\text{exist}}^{(k)}[1]$ 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 that sums up to 1, we apply a scaled softmax non-linearity on the sum vector. More precisely, for each attribute $a$ as either type, colour, size or existence, and the panel position $k$, we compute

$$v_a^{(k)} = \text{softmax}(s_a \cdot v_a^{(k)}),$$

where $s_a$ 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 (that is, position and 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, for example, $I_p = \{1 \}$ represents the case where only the first object is occupied, and $I_p = \{1, 2, 3, \ldots, 9\}$ the case where all objects are occupied in a $3 \times 3$ grid. The position PMF is derived by

$$p_{\text{pos}}[j] = \prod_{k \in I_p} v_{\text{exist}}^{(k)}[0] \prod_{k \in \{1, \ldots, 9\} \setminus I_p} v_{\text{exist}}^{(k)}[1].$$

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

$$p_{\text{type}}[n_{\text{type} + 1}] = 1 - \sum_{j=1}^{n_{\text{type}}} p_{\text{type}}[j].$$

**Mapping PMFs of discrete and continuous attributes to VSA representations**

The PMF of a discrete attribute is represented by a vector space that is spanned by unrelated basis vectors $B := \{b_i\}_{i=1}^n$. Each basis vector $b_i \in \{0, 1\}^k$ is a $d$-dimensional vector with non-zero elements whose index is drawn from a uniform distribution. For representing the PMF of a continuous attribute, we use a vector space that is spanned by a basis taken from the fractional power encoding $\mathbb{F}_2$. Building the basis of the fractional power encoding begins with randomly initializing one single unitary basis vector $e \in \{0, 1\}^k$. The basis vector corresponding to any arbitrary attribute value $v$ is defined by exponentiating the basis vector $e$ with the value as its exponent. For example, the basis vector corresponding to the value $3$ is $e^3 = e \circ e \circ e$. 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 $k$ non-zero elements. In RPMs, however, we exclusively encounter attributes with integer values, for example, the size attribute of an object is enumerated from 1 to $n = 6$. Thus, the underlying codebook is $k$-sparse, of finite size and defined as $B := \{e_i\}_{i=1}^n$. Using this codebook, a PMF of a given continuous attribute is then transformed into 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. Every training RPM example provides eight context panels ($X^{(1,1)}, \ldots, X^{(3,2)}$), eight candidate answer panels ($X^{(0,1)}, \ldots, X^{(0,6)}$), the ground-truth answer $y_{\text{ask}}$, and the ground-truth rule $y_{\text{rule}}$. First, we pass the context panels through the NVSA frontend and infer $p_{\text{pos}}$; similarly,
we infer \( P^0 \) for the candidate answer panels. Using \( P^{(k)} \), we compute the rule belief \( \langle u \rangle \) per attribute using the NVSA backend (for example, using equations (3) and (8) for arithmetic plus and distribute three). On the basis of the distribution of the rule beliefs, we then sample one action \( \langle a \rangle \) per attribute and execute it, yielding five probability distributions for the attributes \( P^{(3,3)} \). Finally, we compute for each candidate answer panel the Jensen–Shannon divergence between each of the five probability distributions in \( P^{(k)} \) and \( P^{(3)} \), and sum up the five Jensen–Shannon divergence values to obtain a score for the answer panel.

Inspired by PrAE\(^{23}\), we update the trainable parameters \( \theta \) by mutually minimizing a loss based on REINFORCE\(^{19}\) and an additional auxiliary loss using the following gradient update

\[
\theta \leftarrow \theta - \beta \left( \nabla_{\theta} \log(\pi(X, y_{\text{task}})) - \nabla_{\theta} \log(\pi(y_{\text{task}})) \right), \tag{19}
\]

where \( \beta \) is the learning rate. The first update term minimizes the categorical cross-entropy loss \( (\log(\pi(X, y_{\text{task}}))) \), whereby the second term, based on REINFORCE, operates on the sampled action log-probability. The third term corresponds to the auxiliary loss, which sums up the negative log-likelihood of the ground-truth rules. 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.

**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, we set the dimension of the bipolar vectors to \( d = 512 \) in the NVSA frontend, while in the NVSA backend we set the dimension of the binary sparse block codes to \( d = 1,024 \) and \( k = 4 \).

We train a separate NVSA, consisting of the frontend (that is, a trainable ResNet-18 with a frozen W) and the backend, per constellation. Motivated by ref. \(^{11}\), the ResNet-18 was pre-trained on the ILSVRC2012 ImageNet-1k dataset. We also modified the first convolutional block of ResNet-18 by reducing its stride from 2 to 1 and removing the maxpooling layer, 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 \( 10^{-4} \) and a constant learning rate of \( 9.5 \times 10^{-5} \). All the training hyperparameters are determined on the basis of 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 the only exception, the \( 3 \times 3 \) grid constellation had to be trained with a batch size 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-based computer 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.

**Data availability**

All three datasets used in this work are openly available. RAVEN is accessible at [http://wellyzhang.github.io/project/raven.html#dataset](http://wellyzhang.github.io/project/raven.html#dataset) and PGM at [https://console.cloud.google.com/storage/browser/ravens-matrices](https://console.cloud.google.com/storage/browser/ravens-matrices). I-RAVEN can be generated using the code provided at [https://github.com/husheng12345/SRAN](https://github.com/husheng12345/SRAN).

**Code availability**

The code used to generate the results of this study is available in the accompanying GitHub repository\(^5\).

**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. *Psychol. Rev.* **97**, 404–431 (1990).
3. Bilker, W. B. et al. Development of abbreviated nine-item forms of the Raven’s standard progressive matrices test. *Assessment* [https://doi.org/10.1177/1073191124466555](https://doi.org/10.1177/1073191124466555) (2012).
4. Barrett, D. G. T., Hill, F., Santoro, A., Morris, A. S. & Lillicrap, T. Measuring abstract reasoning in neural networks. In *Proc. International Conference on Machine Learning (ICML)* (eds Dy, J. & Krause, A.) (PMLR, 2018).
5. Zheng, K., Zha, Z.-J. & Wei, W. Abstract reasoning with distracting features. In *Advances in Neural Information Processing Systems (NeurIPS)* (eds Wallach, H. et al.) (Curran Associates Inc., 2019).
6. Zhang, C. et al. Learning perceptual inference by contrasting. In *Advances in Neural Information Processing Systems (NeurIPS)* (eds Wallach, H. et al.) (Curran Associates Inc., 2019).
7. Zhang, C., Gao, F., Jia, B., Zhu, Y. & Zhu, S.-C. RAVEN: a dataset for relational and analogical visual reasoning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2019).
8. Hu, S., Ma, Y., Liu, X., Wei, Y. & Bai, S. Stratified rule-aware network for abstract visual reasoning. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2021).
9. Jahrens, M. & Martinez, T. Solving Raven’s progressive matrices with multi-layer relation networks. In *2020 International Joint Conference on Neural Networks (IJCNN)* (IEEE, 2020).
10. Benny, Y., Pekar, N. & Wolf, L. Scale-localized abstract reasoning. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2021).
11. Zhuo, T. & Kankanhalli, M. Solving Raven’s progressive matrices with neural networks. Preprint at [arXiv](https://doi.org/10.48550/arXiv.2002.01646) (2020).
12. Zhuo, T., Huang, Q. & Kankanhalli, M. Unsupervised abstract reasoning for Raven’s problem matrices. *IEEE Trans. Image Processing*, **30**, 8332–8341 (2021).
13. Chalmers, D. J., French, R. M. & Hofstadter, D. R. High-level perception, representation, and analogy: a critique of artificial intelligence methodology. *J. Exp. Theor. Artif. Intell.*, **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, 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)* (eds Bengio, S. et al.) (Curran Associates Inc., 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)* (OpenReview.net, 2019).
20. Han, C., Mao, J., Gan, C., Tenenbaum, J. & Wu, J. Visual concept-metaconcept learning. In *Advances in Neural Information Processing Systems (NeurIPS)* (eds Wallach, H. et al.) (Curran Associates Inc., 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) (OpenReview.net, 2022).

22. Yi, K. et al. Clever: collision events for video representation and reasoning. In International Conference on Learning Representations (ICLR) (OpenReview.net, 2020).

23. Zhang, C., Jia, B., Zhu, S.-C. & Zhu, Y. Abstract spatial–temporal reasoning via probabilistic abduction and execution. In Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (IEEE, 2021).

24. Shah, V. et al. Knowledge-based analogical reasoning in neuro-symbolic latent spaces. In 18th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy) (d'Avila Garcez, A. & Jiménez-Ruiz, E.) (CEUR-WS.org, 2022).

25. Rosenblatt, F. Principles of Neuropsychology: 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 Comput. 13, 411–452 (2001).

27. Malsburg, C. V. D. in Brain Theory (eds Palm, G. & Aertsen, A.) 161–176 (Springer, 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. in Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences (eds Holyoak, K. & Aikins, J.) 405 (1998).

30. Gayler, R. W. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In Joint International Conference on Cognitive Science (ICCS/ASCs) (Springer, 2003).

31. Plate, T. A. Holographic reduced representations. IEEE Trans. Neural Netw. 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. Cogn. Comput. 1, 139–159 (2009).

34. Kanerva, P. Large patterns make great symbols: an example of learning from example. In Proc. International Workshop on Hybrid Neural Systems (Springer, 1998).

35. Plate, T. A. Analogy retrieval and processing with distributed vector representations. Expert Syst. https://doi.org/10.1016/j.eswa.2006.04.001 (2000).

36. Gayler, R. W. & Levy, S. D. A distributed basis for analogical mapping. In New Frontiers in Analogy Research: Proc. Second International Analogy Conference-Analogy (eds Koninov, B. et al.) (New Bulgarian University Press, 2009).

37. Rasmussen, D. & Eliasmith, C. A neural model of rule generation in inductive reasoning. Top. Cogn. Sci. 3, 140–153 (2011).

38. 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) (IEEE, 2013).

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

40. Frady, E. P., Kleyko, D., Kymn, C. J., Olshausen, B. A. & Sommer, F. T. Computing on functions using randomized vector representations. Preprint at arXiv https://doi.org/10.48550/arXiv.2109.03429 (2021).

41. Wu, Y., Dong, H., Grosse, R. & Ba, J. The scattering compositional learner: discovering objects, attributes, relationships in analogical reasoning. Preprint at arXiv https://doi.org/10.48550/arXiv.2007.04212 (2020).

42. Maltiński, M. & Mańdziuk, J. Deep learning methods for abstract visual reasoning: a survey on Raven's progressive matrices. Preprint at arXiv https://doi.org/10.48550/arXiv.2201.12382 (2022).

43. Mitchell, M. Abstraction and analogy-making in artificial intelligence. Ann. N. Y. Acad. Sci. 1505, 79–101 (2021).

44. Zhuo, T. & Kankanhalli, M. Effective abstract reasoning with dual-contrast network. In International Conference on Learning Representations (ICLR) (OpenReview.net, 2021).

45. 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 Comput. 32, 2311–2331 (2020).

46. 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 Comput. 32, 2332–2388 (2020).

47. Langenegger, J. et al. In-memory factorization of holographic perceptual representations. In press Nat. Nanotechnol. (2023).

48. Sebastien, A., Le Gallo, M., Khaddam-Aljameh, R. & Eleftheriou, E. Memory devices and applications for in-memory computing. Nat. Nanotechnol. 15, 529–544 (2020).

49. Karunaratne, G. et al. In-memory hyperdimensional computing. Nat. Electron. 3, 327–337 (2020).

50. Karunaratne, G. et al. Robust high-dimensional memory-fragmented neural networks. Nat. Commun. 12, 2468 (2021).

51. Lin, H. et al. Implementation of highly reliable and energy efficient in-memory hamming distance computations in 1 kb 1-transistor-1-memristor arrays. Adv. Mater. Technol. 6, 2100745 (2021).

52. Li, H. et al. Memristive crossbar arrays for storage and computing applications. Adv. Intell. Syst. 3, 2100017 (2021).

53. Serb, A., Kobyzev, I., Wang, J. & Prodromakis, T. A semi-holographic hyperdimensional representation system for hardware-friendly cognitive computing. Philos. Trans. R. Soc. A 378, 20190162 (2020).

54. Kleyko, D., Rachkovskij, D. A., Osipov, E. & Rahimi, A. A survey on hyperdimensional computing aka vector symbolic architectures, part I: models and data transformations. ACM Comput. Surv. 55, 130 (2022).

55. 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. ACM Comput. Surv. 55, 175 (2022).

56. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8, 229–256 (1992).

57. Hersche, M., Zeqiri, M., Benini, L., Sebastian, A. & Rahimi, A. IBM/neuro-vector-symbolic-architectures. Zenodo https://doi.org/10.5281/zenodo.766149 (2023).

Acknowledgements
This work is supported by the Swiss National Science foundation (SNSF), grant 200800. We thank S. El Messoussi for helping with the generalization experiments, R. W. Gayler for insightful comments that contributed to the final shape of the manuscript, and L. Rudin for the careful proofreading. We also thank A. Gray, L. Horesh, K. Clarkson, I. Yunus Akhalwaya and M. Ernoult for fruitful discussions, and C. Apte and R. Haas for managerial support.

Author contributions
A.R. defined the research question and direction. M.H. and M.Z. conceived the methodology and performed the experiments. L.B., A.S. and A.R. supervised the project. M.H. and A.R. wrote the manuscript with input from all authors.
Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-023-00630-8.

Correspondence and requests for materials should be addressed to Abu Sebastian or Abbas Rahimi.

Peer review information Nature Machine Intelligence thanks Yixin Zhu, Tao Zhuo, Ari Morcos and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

© The Author(s), under exclusive licence to Springer Nature Limited 2023