sciendo

Int. J. Appl. Math. Comput. Sci., 2024, Vol. 34, No. 2, 309–321
DOI: 10.61822/amcs-2024-0022

amcs

LEARNING ABSTRACT VISUAL REASONING VIA TASK DECOMPOSITION: A CASE STUDY IN RAVEN PROGRESSIVE MATRICES

JAKUB KWIATKOWSKI [a,*], KRZYSZTOF KRAWIEC [a]

[a] Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznań, Poland
e-mail: jakub.k.kwiatkowski@doctorate.put.poznan.pl,
krawiec@cs.put.poznan.pl

Learning to perform abstract reasoning often requires decomposing the task in question into intermediate subgoals that are not specified upfront, but need to be autonomously devised by the learner. In Raven progressive matrices (RPMs), the task is to choose one of the available answers given a context, where both the context and answers are composite images featuring multiple objects in various spatial arrangements. As this high-level goal is the only guidance available, learning to solve RPMs is challenging. In this study, we propose a deep learning architecture based on the transformer blueprint which, rather than directly making the above choice, addresses the subgoal of predicting the visual properties of individual objects and their arrangements. The multidimensional predictions obtained in this way are then directly juxtaposed to choose the answer. We consider a few ways in which the model parses the visual input into tokens and several regimes of masking parts of the input in self-supervised training. In experimental assessment, the models not only outperform state-of-the-art methods but also provide interesting insights and partial explanations about the inference. The design of the method also makes it immune to biases that are known to be present in some RPM benchmarks.

Keywords: abstract visual reasoning, Raven progressive matrices, machine learning, problem decomposition.

1. Introduction

One of the key attributes of general intelligence is abstract reasoning, which, among others, subsumes the capacity to reason about and complete sequential patterns. To quantify such capabilities in humans and diagnose related deficiencies, John C. Raven devised in the 1930s a visual test contemporarily known as Raven progressive matrices (RPMs) (Raven, 1936). An RPM problem (Fig. 1) is a 3×3 grid of eight *context panels* containing arrangements of 2D geometric objects. The objects adhere to rules that govern the relationships between the panels in rows, e.g., progression of the number of objects, a logical operation concerning their presence, etc. The task is to complete the puzzle by replacing the lower-right *query panel* with the correct image from the eight provided *answer panels*. Solving the task requires 'disentangling' the rules corresponding to rows and columns and capturing the analogies between the observed patterns.

RPMs have been more recently adopted in the AI community for assessing similar capabilities in artificial intelligent systems, along with other benchmarks like Bongard problems (Bongard, 1970) and Hofstadter's analogies (Hofstadter, 1995). Recent advances in machine learning have accelerated this trend, with deep learning becoming the primary paradigm of choice for designing such systems (Małkiński and Mańdziuk, 2022a).

The original collection of RPM problems, standard progressive matrices (Raven, 1936), comprised just 60 tasks, which is not enough to effectively train data-hungry machine learning models. Therefore, several larger datasets and task generators have been devised, among them RAVEN (Zhang *et al*., 2019a) and I-RAVEN (Hu *et al*., 2021). In this process, it became evident that designing a representative, diverse, and varying in difficulty collection of RPM tasks is nontrivial. The key challenge is that one needs to generate seven incorrect answer panels such that it is impossible to deduce the correct answer panel from them. Unfortunately, most

*Corresponding author

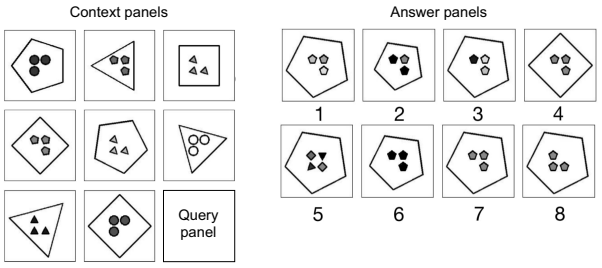

Fig. 1. Example of an RPM task.

tasks in RAVEN do not meet this requirement: the correct answer panel can be selected by identifying the most frequent attributes across all eight answer panels (shape, size, etc.) and picking the answer panel with those properties. This flaw can be easily exploited, which was epitomized with the so-called *context-blind* methods (Wu *et al*., 2020) that achieve almost perfect scores on RAVEN by disregarding the context entirely and making decisions based on answer panels only. This problem has been termed *biased answer set*, and we illustrate it in Section SM9 of Supplementary Material (SM).

In this study, we circumvent the above problem by decomposing RPM tasks into two stages: (i) prediction of properties of the query panel, and (ii) identifying the answer panel with properties that match those predicted ones the best. To this aim, we use the abstract properties available in the RAVEN benchmarks and design a bespoke deep learning architecture based on the transformer blueprint (Vaswani *et al*., 2017). The resulting approach, which we dub the abstract compositional transformer, is not only more transparent than end-to-end neural architectures but also immune to biased answer sets and capable of surpassing the state-of-the-art performance. More specifically, the property prediction stage is immune to biases because it does not involve the answer panels, while the second stage does not entail learning and thus by definition cannot be biased by the content of a training set. Also, to the best of our knowledge, this is the first attempt to predict symbolic descriptors of RPM puzzles and the first study on self-supervised learning for RPMs. The two-stage approach and model architecture (Section 2), a bespoke training procedure (Section 3) and an extensive empirical analysis concerning property prediction (Section 5) and problem solving (Section 6) form our main contributions.

2. Proposed approach

Rather than training models that choose answer panels in RPMs, we propose to rely on *property prediction*, in which models generate an abstract, structured representation of the missing panel (Fig. 2). To this aim, we rely on the RAVEN dataset (Zhang *et al*., 2019a), in which tasks have been generated from symbolic specifications expressed in an image description grammar that captures visual concepts such as the position, type of shape, color, number of objects, inside-outside, etc. The objective of the model is to predict these properties for the query panel and for the answer panels. A trained property prediction model is then subsequently used to choose the answer panel.

Our model comprises three modules: the image tokenizer, transformer, and property predictor; we describe them in subsections that follow (see Section SM1 in Supplementary Material for technical details). Even though RPM problems have an inherent 2D structure, we rely on *sequence-to-sequence* transformers (Vaswani *et al*., 2017) and demonstrate, analogously to prior work on applying such models to 2D images (Dosovitskiy *et al*., 2020), that effective RPM solvers can be obtained without explicitly presenting the spatial structure of the problem to the model.

2.1. Image tokenizer. The tokenizer maps the 2D raster representation of an RPM problem to a sequence of abstract tokens using a convolutional neural network (CNN) that gradually contracts the input image to lower spatial resolutions in consecutive layers, while increasing the number of channels. The multi-channel superpixels produced by the last layer form the tokens, i.e., a token is the vector of channels produced by the CNN at a given location. In experiments, our CNN is EfficientNetV2B0 (Tan and Le, 2021) pretrained on the ImageNet database (Deng *et al*., 2009). We consider the three following types of tokenizers that vary in how they perceive the panel rasters (when necessary, the single-channel monochrome RPM image is broadcast to RGB input channels of the CNN).

Panel tokenizer. In this variant, the raster image representing each panel is tokenized independently. The CNN is applied to each raster (84×84 pixels in RAVEN) and produces a 3×3 array of 128-channel superpixels, which is then flattened row-wise into a sequence of nine 128-dimensional tokens. This is repeated for all nine panels of the puzzle, and the resulting sequences are concatenated, producing 81 tokens in total.

Task tokenizer. In this variant, the raster image of the entire RPM task is tokenized with a single invocation of the CNN. For RAVEN, this means applying the CNN to a 252×252 pixel image, which results in an 8×8 array of 128-channel superpixels, then flattened row-wise, leading to a sequence of 64 128-dimensional tokens.

Row tokenizer. In this variant, the rasters representing individual panels in each row are first stacked channel-wise, resulting in three 3-channel 84×84 images corresponding to the top, middle, and bottom

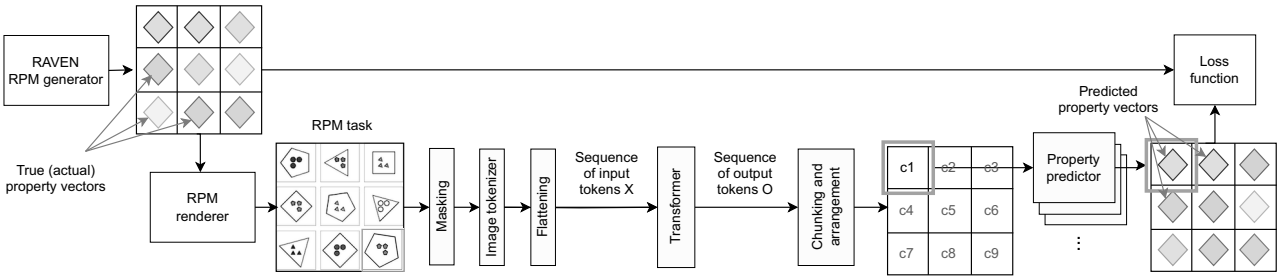

Fig. 2. Architecture of the model and its training process, guided by the loss function that compares the predicted and actual properties of RPM panels. The model learns from completed RPM tasks, with one of the panels (context or query panel) masked out, and predicts the properties of all panels.

rows of the puzzle. The CNN is queried on each of those images independently and produces a 3×3 array of superpixels in each query, which are then flattened row-wise, resulting in nine 128-dimensional tokens. Finally, the subsequences for the top, middle, and bottom RPM rows are concatenated, resulting in 27 tokens.

By stacking the panel images, we directly juxtapose them in input channels, thus allowing the CNN to form low-level features that capture the differences between the individual images. The RPM images from the left, central, and right columns end up being interpreted by the CNN as, respectively, the red, green, and blue channels. The pretrained CNN is trained alongside the entire model, and can thus adapt to the characteristics of the RPM.

The relatively small sizes of rasters, combined with 18 convolutional layers of the CNN (cf. Tan and Le, 2019, Table 1), cause the receptive fields of units in the last layer to span the entire input image. Therefore, for all tokenizers, each token may in principle depend on the entire input raster (a panel raster for Panel and Row tokenizers, and a task raster for the Task tokenizer). Also, only the Panel tokenizer is guaranteed to ensure some degree of selective correspondence between RPM panels and tokens. In the Task tokenizer, the representation is more entangled, as the receptive fields of the CNN are allowed to span *multiple* neighboring panels. In the Row tokenizer, the consecutive groups of nine tokens form an entangled representation of the top, middle and bottom row of panels. However, the degree of entanglement depends on the characteristics of features acquired in training, and the actual *effective receptive fields* can be smaller (see, e.g., Luo *et al*., 2017).

2.2. Sequence-to-sequence transformer. The transformer processes the one-dimensional sequences of tokens X produced by an image tokenizer by first encoding each token independently using the encoder,

$$Z = map(Encoder_{\theta_E}, X), \tag{1}$$

which is implemented as a dense linear layer, primarily meant to match the dimensionality of the tokens to the input dimensionality of the transformer. Then, the transformer maps the sequence of encoded tokens Z to a sequence of output tokens O of the same length:

$$O = Transformer(Z; \theta_T). \tag{2}$$

The model is equipped with a *learnable positional encoding*, applied to the input tokens in Z. As the number of tokens is constant, we encode the *absolute* positions of tokens in Z, which can be achieved with a fixed-size learnable embedding. There is a single entry in the embedding for each position in the input sequence, and thus 81, 64, and 27 entries for respectively the Panel, Task, and Row tokenizers. The embedding vectors are added to respective tokens in Z before passing them to the transformer.

Internally, the transformer is a stack of transformer blocks, each of them consisting of a multi-head attention mechanism $Attn(\theta_A)$, normalization layers $LayerNorm$, a feed-forward network $f(\theta_f)$ and skip connections. The processing for the i-th token can be summarized as

$$\begin{aligned} a_i &= Attn(LayerNorm(z_i; \theta_N); \theta_A), \\ m_i &= a_i + z_i, \\ o_i &= f(LayerNorm(m_i; \theta_{N'}); \theta_f) + m_i. \end{aligned} \tag{3}$$

The resulting tokens o_i form the output sequence O. For a detailed description of transformer and multi-head attention, see the work of Vaswani *et al*. (2017).

2.3. Property predictor. The property predictor maps the sequence of tokens O produced by the transformer to the properties of individual panels as follows:

1. the sequence is sliced into nine *chunks* of equal length;

2. the tokens in each chunk are concatenated to form a single vector;

3. each vector is independently mapped to a *property vector* (Sec. 2.4) using a dense subnetwork.

For the Panel tokenizer, which produces 81 tokens, the chunk length is 9; for the Row tokenizer, which produces 27 tokens, the chunk length is 3; for the Task tokenizer, producing 64 tokens, each chunk comprises seven tokens, and the last token is discarded.

The nine property vectors obtained in this way are assumed to correspond to RPM panels, traversed row-wise and left-to-right in each row (eight context panels and the query panel). Associating the chunks with panels requires the transformer to both *combine and disentangle* the information carried by the input tokens. The combining is necessitated by the task, which requires detecting the patterns adhered to by the context panels. The disentanglement, on the other hand, is necessary for the Task and Row tokenizers, which do not derive tokens from individual context panels *independently*, but aggregate information from multiple panels.

2.4. Property vectors. Following the RAVEN family of benchmarks (Zhang *et al*., 2019a; Hu *et al*., 2021; Benny *et al*., 2021), we assume the panels to be composed of *objects* that can appear in one of seven *spatial arrangements*, each containing at least one object, with the maximum number of objects as follows: *center-single* (1), *distribute-four* (4), *distribute-nine* (9), *in-center-single-out-center-single* (2), *in-distribute-four-out-center-single* (5), *left-center-single-right-center-single* (2), *up-center-single-down-center-single* (2). An RPM panel is represented as a property vector of a fixed dimensionality comprising 'slots' for all objects in every arrangement; there are thus 25 slots in total. Each object is characterized by three *appearance properties* with the following admissible values:

- color: 255, 224, 196, 168, 140, 112, 84, 56, 28, 0 (10 values, rendered as colors in this paper),
- size: 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 (6 values),
- type: triangle, square, pentagon, hexagon, circle (5 values).

Another appearance property used by Zhang *et al*. (2019a) is the object's rotation angle. However, the angle is drawn at random when generating a panel, i.e., it does not influence the reasoning rules. A successful solver should disregard this characteristic, hence we do not include it in our representation.

In total, a property vector comprises thus 101 variables:

- the identifier of the arrangement (1 variable),
- $present_i$: a group of 25 binary variables, each indicating the presence/absence in the i-th object slot;
- 75 appearance properties for the objects in all slots.

Relevance of properties. An element of a property vector may deem some other elements *relevant* or *irrelevant*. For instance, if *arrangement* = *distribute-four*, only $present_i$ for four indices i matter; if then only two of them are set to 1, only the corresponding $2 \times 3 = 6$ appearance properties describing the two indicated objects are relevant. The number of relevant properties varies from 5 for *center-single* (1 *arrangement* + $present_i$ for a single object + 3 appearance properties) to 37 for *distribute-nine* (1 *arrangement* + 9 $present_i$ properties + 9×3 appearance properties).

The distinction between relevant and irrelevant panel properties is essential; in particular, the loss function and the metrics are calculated from relevant properties only. When confronting two property vectors, one of them serves as the *source of relevance*; this depends on the use case (more details in Section SM2 and in the experimental part).

Encoding of property vectors. To make properties amenable to differentiable loss functions, we represent all variables with one-hot-encoding, which results in the low-level representation of a panel being a binary vector of $7 + 25 + 25 \times (10 + 6 + 5) = 557$ dimensions. Respectively, models produce for each panel a 557-dimensional vector with values in $[0, 1]$, which is assured by forcing the dimensions representing a given categorical variable through the softmax activation function; for instance, the first seven dimensions represent the probability distribution over arrangements. To calculate the loss, the distribution predicted by the model is confronted with the corresponding one-hot target distribution using categorical entropy. The entropies so obtained for particular variables are multiplied by weights tuned and fixed in the early stages of this project (see Section SM3 of Supplementary Material). The loss is calculated only for the distributions of relevant properties, with the target property vector acting as the source of relevance. The overall loss on a given RPM task is a weighted sum of the losses for individual panels, where weights depend on the type of masking applied in training (Section 3).

3. Model training

As in natural language processing, our models are trained via self-supervision, i.e., they are tasked to predict a masked-out element of the input sequence, given the context of the visible elements. While masking usually concerns tokens, RPMs require making decisions about panels; therefore, in each training step, the tokenizer is applied to the task with a single context panel masked out.

In an RPM, one can make predictions in both the directions of the sequence of context panels, due to the

nature of the underlying patterns (e.g., the progression of the number of objects). Thus it seems desirable to mask randomly chosen panels to facilitate the learning of patterns *across the entire RPM puzzle*, and prospectively use that knowledge for making decisions about the query panel. The query panel is empty by definition, so it seems natural to treat it as a masked-out one, too. However, masking out more than one panel at once would be inconsistent with test-time querying (when only the query panel is masked) and could lead to ambiguity, i.e., multiple answer panels being correct. Therefore, we split training into two phases as follows:

1. **Random masking phase.** Each puzzle is completed with the correct answer from the answer panels, and a randomly chosen panel (one of nine) is masked out. This is realized 'on the fly', so the same task has different panels masked out in different epochs.

2. **Query masking phase.** The RPM tasks are presented as-is, with the query panel masked out, and no additional masking is applied. The weight of the loss related to the query panel is multiplied by 0.01, in order for this phase to act as fine-tuning after Phase 1. The loss function is not applied to the non-masked panels, i.e., the model is not penalized for making predictions there.

By staging training into these phases, we allow the model to first learn the patterns across the entire RPM board, and only then require it to focus on the query panel.

Masking requires replacing a panel with a 'neutral' image; initially, we considered empty (zeroed) panels and random noise images. Ultimately, the *trainable masks* performed best. The masking image is initialized with random noise and considered a parameter of the model, i.e., it is continuously updated in training, ultimately expressing the cumulative input that the model 'expects' at masked panels

4. Related work

RPMs have been long considered an interesting benchmark for abstract reasoning systems, along with Bongard problems (Bongard, 1970), Hofstadter's analogies (Hofstadter, 1995), Numbo (Defays, 1995), or Sudoku, to name a few. The advent of deep learning has only intensified this interest, with an outpour of studies proposing various architectures and learning approaches. A recent survey (Małkiński and Mańdziuk, 2022a) cites at least 34 papers, most of them published within the last five years; Tables 8 and 9 cite those of them that achieved the best performances on the RAVEN (Zhang *et al*., 2019a) and I-RAVEN (Hu *et al*., 2021) benchmarks. Of those, the model that bears the most architectural similarity to the approach proposed in this paper is the attention relation network (ARNe) (Hahne *et al*., 2019), which engages a transformer to facilitate spatial abstract reasoning. However, like almost all other studies that we are aware of, ARNe is trained to choose answer panels, rather than to predict panel properties, and thus uses the transformer blueprint in a very different way.

In terms of the taxonomy proposed by Małkiński and Mańdziuk (2022a), our approach could be classified as a relational reasoning network, as a part of the model is delegated to learn relations between context panels. A notable representative of this class is the wild relation network (WReN) (Barrett *et al*., 2018), in which a relation network is used to score the answer panels. By contrast, our approach does not model the relations between panels explicitly, but delegates relational learning to the transformer, while encoding the spatial characteristics of the task as a sequence of tokens.

Our approach bears resemblance to some past works in the hierarchical networks category delineated by Małkiński and Mańdziuk (2022a). More specifically, the way in which our Row tokenizer stacks panels channel-wise is analogous to the design of 'perceptual backbones' in, e.g., the stratified rule-aware network (SRAN) (Hu *et al.*, 2021; Małkiński and Mańdziuk, 2022a, Fig. 8). Notice, however, that the panel stacking used in the Row tokenizer is the only way in which we explicitly reveal the relationships between panels to the model. All remaining logic about the correspondence, succession, progression, etc. of patterns in the panels needs to be autonomously learned by juxtaposing input tokens. This mitigates manual modeling of relationships and, consequently, human biases.

Last but not least, there were several works in which the models, apart from choosing the right answer panel, were required to make predictions about the *rules* that govern the generation of RPM tasks. This has been attempted via auxiliary terms in the loss function by Zhang *et al*. (2019a), but the models were not benefitting from this extension, or even underperforming when compared with the reference architecture (Section 6.4 therein). Similar negative results have been reported by, among others, Hu *et al*. (2021), Zhang *et al*. (2019b) and several follow-up studies (see Małkiński and Mańdziuk, 2022, Section 3.2). Preliminary encouraging results in addressing these challenges were presented by Małkiński and Mańdziuk (2022b).

5. Results: Property prediction

In this section, we cover the experimental results for the property prediction tasks; in Section 6, we use the trained models to solve the choice tasks. Implementation details are available in Supplementary Material.

Following Zhang *et al*. (2019a), we use the original

division of the 70,000 tasks from the RAVEN database[1] (7 spatial arrangements $\times$10,000 tasks) into training, validation and test sets of, respectively, 42,000, 14,000, and 14,000 tasks. We train nine models in total, using the three types of image tokenizers and three masking regimes in training (Section 3): **Combined**, comprising 200 epochs of random masking and up to 30 epochs of query masking, and two ablative regimes: **Query**-only (query masking for 200 epochs) and **Random**-only (random masking for 200 epochs). In the first phase of masking (whether followed by a second phase or not), the weight of the loss for the masked panel is multiplied by 2, to emphasize its importance (this multiplier value was found beneficial in preliminary experiments). Validation takes place after each epoch, and the model with the lowest validation error is selected.

To assess the models' capacity to predict panel properties, we devise a range of test-set metrics that are calculated on the relevant properties only, with the target property vector acting as the source of relevance, i.e., determining the properties that are deemed relevant for a given task (Sections 2.4 and SM2):

- **Correct**: The primary metric in the further discussion. Amounts to 1 if *all* relevant properties of the query panel have been correctly predicted; otherwise 0. Averaged across all tasks in the testing set.
- **PropRate**: The fraction of correctly predicted relevant properties, across all test tasks.
- **AvgProp**: The fraction of relevant properties correctly predicted, averaged over tasks. For a given task, it amounts to 1 if all relevant properties have been predicted correctly, and 0 if none. Because the number of relevant properties varies by task and panel, AvgProp is not equivalent to PropRate.
- **AvgH**: The Hamming distance between the predictions and the target on the relevant properties, averaged over tasks. For a given task, the best attainable value of this metric is obviously 0, while the worst one corresponds to the scenario with nine objects (*distribute-nine* arrangement) and amounts to 37 (1 for the incorrect identifier of the arrangement, plus 9 for the incorrect setting of the nine corresponding presence/absence variables, plus $9 \times 3 = 27$ incorrect values of the color, size and type of an object).

Results. In the *Property prediction* part of Table 1, we report the performance of particular models. Both the type of tokenizer and the masking scheme are strong determinants of model capabilities. The models that tokenize each panel separately (Panel) fare the worst on all metrics. Tokenizing the entire task at once (Task) leads to much better predictive accuracy. Nevertheless, the model that involves the Row tokenizer systematically fares best when juxtaposed with the others, which suggests that superimposing panels as separate image channels facilitates inferring relevant patterns.

The observed differences might be partly due to the number of tokens used in particular architectures (81, 64, and 27 for Panel, Task and Row). However, the Row tokenizer uses the *fewest* tokens, so it is in principle most likely to suffer from the 'information bottleneck'; nevertheless, it outperforms the other two types of models. This suggests that the way a sequence of input tokens 'folds' the task image is more important than its length.

Concerning the masking schemes, masking only the query panel (Query) throughout the entire training process turns out to be very ineffective. By contrast, Random masking performs much better. This may seem paradoxical, as making predictions about the query panel is less demanding for the learner: as it is located at the end of the third row and the third column of the RPM grid, predicting its properties requires only *extrapolation* of the properties observed in the other rows and columns. On the other hand, masking random panels involves also making predictions about the middle panels (requiring *interpolation*) and about the first panels (requiring extrapolation in the opposite direction). A model trained in the Random mode has to master all these skills, yet it proves better when tested only at the query panel. This shows that forcing the transformer to detect and reason about patterns observed across the entire puzzle helps it generalize better.

While training in the Random mode outperforms the query masking mode by a large margin, Table 1 suggests that even better predictive accuracy can be attained when the former is followed by the latter in training (Combined mode). Focusing on the query panel in the later stages of training is thus beneficial. The learning curves presented in Section SM8 of Supplementary Material align with this conclusion: the metrics tend to saturate towards the end of the Random phase and experience increase once training switches to the Query phase.

To corroborate these observations, in the *Classification* part of Table 1 we report the values of selected metrics *calculated for the context panels*. As these panels are not masked out, the model can directly observe them, and predicting their properties is much easier as they do not need to be inferred from the logical rules that govern the puzzle. The metrics are thus much better, with some models attaining almost perfect values. The tokenizer type has an opposite impact on classification compared to prediction: the Panel models perform the best, presumably because separate tokenization reduces the 'cross-talk' between panels. The complete set of metrics for classification are given in Table SM2.

[1] https://github.com/WellyZhang/RAVEN.

Table 1. Comparison of configurations on the property prediction task.

Tokenizer	Masking	Property prediction				Classification	
		Correct	PropRate	AvgProp	AvgH	Correct	AvgProp
Panel	Query	1.53	62.23	58.08	4.55	97.54	99.50
	Random	20.82	82.52	80.34	2.23	98.38	99.44
	Combined	22.17	83.33	81.29	2.14	98.28	99.49
Task	Query	18.91	81.23	79.25	2.41	89.69	98.35
	Random	72.63	95.80	95.25	0.65	88.44	98.00
	Combined	75.63	96.15	95.64	0.61	88.33	97.98
Row	Query	20.56	79.96	78.15	2.66	84.03	97.68
	Random	75.44	96.17	95.69	0.60	87.64	98.22
	Combined	77.58	96.47	95.99	0.56	87.85	98.25

Table 2. Sizes and querying costs for particular models.

Tokenizer	#Parameters [M]		MFLOPS	
	Transformer	Total	Transformer	Total
Panel	2.65	11.45	87.77	2326.16
Task	2.65	11.19	52.84	2002.38
Row	2.64	10.68	29.02	793.97

Table 3. Sizes and querying costs for dense models.

Dense	#Parameters [M]		MFLOPS	
layer size	Transformer	Total	Transformer	Total
336	2.67	10.70	5.34	770.23
512	4.33	12.37	8.67	773.57

In Table 2, we characterize the sizes and computational requirements of particular models. The Row model that excels at predictive accuracy is also the smallest and cheapest at querying. The slight differences in the number of parameters of transformers result from the number of tokens, which determines the that of entries in the learnable embedding used for positional encoding. Relative differences in the total number of parameters are somewhat larger, and stem from the combined dimensionalities of the chunks of output tokens; a chunk is mapped to a property vector using a dense layer and thus its size impacts the number of parameters. Despite these differences in the number of parameters being moderate, the computational cost of querying the Row model is several times lower than for Panel and Task tokenizers. This is due to the larger number of tokens processed by transformers in those models, which leads to quadratically more query-key interactions.

Does transformer matter? One of the research questions of this study concerns the importance of the transformer blueprint, i.e., whether learning to model direct interactions between tokens representing parts of the input brings any advantage compared to more straightforward approaches. To verify this hypothesis, we consider baseline *denseformers* architectures in which the transformer is replaced with a dense subnetwork: the tokens produced by the tokenizer are concatenated into a vector and fed into a subnetwork comprising five dense layers of the same size. The output of the last dense layer is then passed to the property predictor and undergoes further processing as in our model, i.e., it is sliced into nine chunks used to predict the properties of individual panels (Section 2.3). There are no other differences between denseformers and our models.

Given that the Row tokenizer proved most capable (Table 1), we design two comparable denseformer variants, with dense layer size 336 and 512, so that the total number of parameters and cost of querying are similar (Tables 3 and 2). Each variant is trained in the Random masking mode, once with and once without regularization consisting of layer normalization (Lei Ba *et al.*, 2016) and dropout.

Table 4 summarizes the performance of denseformers in terms of metrics from Table 1. The densformers are clearly inferior to the transformers: except for the AvgH metric, none of them attains even the worst value of the corresponding metric for the transformers. While layer normalization (Lei Ba *et al.*, 2016) has a positive impact on predictive accuracy, increasing the layer size from 336 to 512 improves the accuracy only slightly, which suggests that boosting it further, beyond 512 units, is unlikely to lead to significant improvements. We thus conclude that the 'cross-talk' between tokens representing parts of the RPM task, facilitated by the transformer architecture, brings significant added value, and perhaps is even essential for this kind of tasks.

Table 4. Comparison of densformer models.

Dense size	Layer normal.	CorrRate	PropRate	AvgProp	AvgH
336	No	0.24	48.41	43.08	6.03
	Yes	0.45	55.19	51.24	5.43
512	No	0.28	49.76	44.76	5.88
	Yes	0.88	55.92	52.03	5.36

Structure of errors. We encode the appearance properties as unordered categorical variables, but in fact they are ordinal. In Section SM7, we show that models are much more likely to commit small errors on these properties than large ones, which implies that they correctly discovered the ordinal nature of attributes, even though it was not engraved in their architectures nor conveyed to them explicitly in training. Such insights are not available in approaches that directly learn to choose answer panels.

6. Results: Choice tasks

In this section, we use the models trained for property prediction in Section 5 for solving RPM tasks. To this aim, we devise the direct choice maker (DCM) algorithm that makes decisions by comparing the *prediction* for the masked query panel (given context panels) with the *classification* of individual answer panels in the same context. Given a trained model P and a task T, the DCM proceeds as follows:

1. P is queried on T as in property prediction, i.e., on the context panels of T, with the query panel masked out. The 9th property vector p produced in response, corresponding to the query panel, is the model's **prediction** of the answer to the task.

2. For each of the eight answer panels, $i = 1, \ldots, 8$, P is queried on T with the query panel replaced with the i-th answer panel. In each of those queries, the 9th property vector is stored as p_i. This will be referred to as **classification** of a panel (in terms of its properties).

3. A distance function d is applied to the pairs (p, p_i), and the answer panel with the minimal $d(p, p_i)$ is returned as the solution to T. The distance functions (explained in the following) take into account only the relevant properties, where their relevance is determined by p_i (see Section SM2).

   We use p_i as the source of relevance when calculating $d(p, p_i)$, because classification is in general easier than prediction (cf. Table 1), so p_i's are less likely to make mistakes in determining the relevance of properties.

We devise three performance metrics, each calculating the percentage of tasks for which the DCM selects the correct answer panel. The metrics vary in the type of property vectors (categorical or encoded) and in d. **AccUnique** uses the DCM with categorical property vectors and the Hamming distance as d. A tie ($d(p, p_i)$ being minimized by two or more answer panels) counts as a failure. **AccTop** operates like AccUnique, except that a tie on the closest matches counts as a success if one of them points to the correct answer. Finally, **AccProb** applies the DCM to the encoded property vectors, i.e., probability distributions produced by the model, and uses binary cross-entropy for vector elements corresponding to binary properties (e.g., object presence) and categorical cross-entropy for multi-valued properties (e.g., object size), summing them in d over all relevant properties. Similarly to loss functions and metrics used in Section 5, all metrics are calculated on the test set.

Optimistic bounds. We first estimate the informal optimistic performance bounds, i.e., the test-set metrics that the DCM would attain *if the true property vectors (classifications) were known for answer panels*. These vectors are provided in the RAVEN database, so we use them as p_is in Step 2 of the DCM (and let them determine the relevance of properties), rather than querying the model. For AccProb, this implies comparing the continuously-valued probabilities produced by the model with one-hot vectors representing the categorical values of true properties. Table 5 presents the resulting estimates. As expected, AccUnique is the most demanding metric, as it requires the predicted property vector to be strictly closest to the property vector for exactly one of the answer panels. By contrast, AccTop treats ties as successes and thus reports significantly better scores. However, this metric does not reflect the model's capability of pointing to a unique solution among the answer panels. AccProb is the most pragmatic metric, due to the low likelihood of ties between answer panels and sensitivity to nuanced, continuously-valued responses of the model, so we focus on this metric in the following.

The relations between the models in Table 5 correlate with the quality of property prediction (Table 1), with the Row tokenizer being on average better than Task and Panel, the Combined masking mode slightly outperforming the Random mode, and the latter one in turn being much better than the Query-only mode. Expectedly, high accuracy of property prediction implies better choice making.

Accuracy. Table 6 presents the actual metrics summarizing models' capability of solving RPM choice tasks, i.e., with p_i's resulting from the classification of

Table 5. Optimistic bounds, with the DCM relying on *target* property vectors.

Tokenizer	Masking	AccProb	AccTop	AccUnique
Panel	Query	19.00	39.48	6.16
	Random	55.66	70.87	37.47
	Combined	57.57	72.34	39.27
Task	Query	65.09	74.18	38.18
	Random	94.84	96.89	90.70
	Combined	95.39	97.23	92.69
Row	Query	66.45	72.04	36.10
	Random	95.49	96.99	92.62
	Combined	95.90	97.48	94.04

answer panels. Except for two models, AccProb is noticeably worse than in Table 5, which was expected because the classifications p_i of answer panels can now diverge from the true vectors. Indeed, we calculated also the Correct metric (used in Section 5 for assessing the accuracy of property prediction) on classification alone in this setting, and it amounted to 75.95% and 58.77% respectively for Task and Row. This difference is likely the main factor that makes the former model fare much better in Table 6. We hypothesize that the root cause for this difference is that querying the Row models in classification means replacing an entire input channel of the input image (corresponding to the query panel) with an answer panel, while in training that panel was continuously providing the 'neutral' values from a learnable mask. For the Task tokenizer, this affects only $1/9$ of the input raster of the entire task (recall that tokens' receptive fields capture the entire input image in all tokenizers).

The models trained in the Query masking mode fare the worst again; clearly, the low capacity of predicting properties (Table 1) prevents them from choosing the right panels with the DCM. For the Panel and Task tokenizers, the pattern is consistent with previous tables: the Combined mode performs better than Random.

Due to the high computational cost of training, a single model was trained per configuration. To establish statistical significance, we conducted additional runs for the best-performing configurations and report the resulting averages as well as .95-confidence intervals for sample size 3 obtained in this way in the $\text{AccProb}_{(n=3)}$ column of Table 6. The figures largely confirm our earlier observations.

In testing, the models are queried on 'completed' tasks, with all nine panels present and no panel masked out. In training, they perform classification for the eight unmasked panels and prediction for the single masked panel, but they are never asked to perform classification for all panels. This may be particularly relevant for the Task tokenizer, which in training observes a single panel being masked *in each invocation* (in contrast to Panel and Row); in particular, in the Query-only mode, it is always the lower-right panel. This may explain the particularly bad performance of that model, with AccProb below the 12.5% achievable with choosing answer panels at random.[2]

Table 7 summarizes the performance of the same models on the testing part of the I-RAVEN benchmark (Hu *et al.*, 2021),[3] which features the same tasks as RAVEN, although with answer panels generated in an unbiased way. Apart from the Task+Query and Task+Combined models that fared best on RAVEN and observe slight deterioration, all remaining models perform better on I-RAVEN. Because the context panels and the *correct* answer panels are identical in both benchmarks, so must be the predictions of properties made by the models for them. Therefore, the differences between Tables 6 and 7 can be only due to the classifications of the *incorrect* answer panels. Apparently, the unbiased answer panels from I-RAVEN are less likely to result in property vectors that distort the assessment of relative similarities of the answer panels to the predicted answer.

Comparison with state-of-the-art. Following a recent survey (Małkiński and Mańdziuk, 2022a), in Table 8 we reproduce the test-set accuracy of five RPM solvers reported in past literature on the topic, which attain the best performance on the test part of the RAVEN collection. Table 9 presents analogous top results for the I-RAVEN benchmark (see the survey for the performance of other, less capable methods). The reported figures should be juxtaposed with the AccProb metric from previous tables. For reference, we quote also the estimated accuracy of the human performance.

Compared with these approaches, the performance of several variants of our models is very good, with two of them equipped with the Task tokenizer outperforming not only the reported human accuracy on RAVEN (Zhang *et al.*, 2019a), but also *all previously reported methods on this benchmark*, the best of which attained 94.1% (column AccProb in Table 6 vs. Table 8). For I-RAVEN, our method beats all-but-one of the SotA methods (Table 7 vs. Table 9).

Examples. Figure 3 compares visually the behavior of the Task and the Row model for two tasks from the RAVEN test set (rotation angle is fixed when rendering models' predictions and classifications). In the first example (Fig. 3(a)), both models produce perfect predictions and similar, though imperfect, classifications of answer panels. However, the Row model fails to choose the correct

[2]Notice, however, that this explanation ignores the cross-talk between tokens that takes place in further processing by the transformer.

[3]Earlier published under the name Balanced-RAVEN (Hu *et al.*, 2020).

Table 6. Accuracy on choice tasks, based on *predicted* property vectors.

Tokenizer	Masking	AccProb	AccProb$_{(n=3)}$	AccTop	AccUnique
Panel	Query	17.79	—	39.13	6.33
	Random	41.39	—	59.75	25.65
	Combined	46.85	—	63.82	30.74
Task	Query	5.44	—	42.72	0.96
	Random	96.33	95.81±1.47	96.22	83.00
	Combined	96.97	96.45±1.13	96.57	86.30
Row	Query	30.97	—	63.27	5.32
	Random	79.23	80.58±5.67	94.68	25.47
	Combined	82.84	84.66±6.28	95.43	33.53

Table 7. Accuracy on the test set of the I-RAVEN benchmark.

Tokenizer	Masking	AccProb	AccTop	AccUnique
Panel	Query	45.44	64.89	27.81
	Random	53.59	70.22	34.61
	Combined	60.16	74.08	41.24
Task	Query	12.34	51.68	2.50
	Random	94.90	95.42	86.43
	Combined	95.39	95.61	88.95
Row	Query	51.35	76.74	14.30
	Random	86.35	95.10	42.41
	Combined	88.52	95.55	52.09

Table 8. State-of-the-art results on the RAVEN test set (source: Małkiński and Mańdziuk, 2022a).

Method	Accuracy
Rel-Base (Spratley *et al*., 2020)	91.7
CoPINet + AL (Kim *et al*., 2020)	93.5
DCNet (Zhuo and Kankanhalli, 2021)	93.6
CoPINet + ACL (Kim *et al*., 2020)	93.7
Rel-AIR (Spratley *et al*., 2020)	94.1
Ours	97.0
Human (Zhang *et al*., 2019a)	84.4

Table 9. State-of-the-art results on the I-RAVEN test set (source: Małkiński and Mańdziuk, 2022a).

Method	Accuracy
SRAN (Hu *et al*., 2021)	60.8
SRAN MLCL+DA (Małkiński and Mańdziuk, 2022b)	73.3
MRNet (Benny *et al*., 2021)	86.8
SCL (Wu *et al*., 2020)	95.0
SCL MLCL+DA (Małkiński and Mańdziuk, 2022b)	96.8
Ours	95.4

answer, as it classifies the square in the answer panel as a triangle. As a consequence, p_8 is more similar to prediction p than p_7 (although p_7 and p_8 look identical when rendered as images, the raw outputs of models varied when assessed with cross-entropy that the DCM uses as the distance function d). The Task model produces a more faithful classification p_8 of the last answer panel and thus correctly points to p_7.

The second task (Fig. 3(b)) is harder by involving more objects and more complex rules. As a result, not only the classifications, but also the predictions are far from perfect, with imprecise predictions for sizes, colors, and occasionally even shapes of objects (object presence is always correctly predicted and reproduced). Nevertheless, the Task model is more consistent when predicting and classifying and thus chooses the correct answer.

More examples are provided in Section SM10 of Supplementary Material.

Discussion. The ultimate superiority of the Task models (Tables 6 and 7) suggests that it is desirable to tokenize tasks as they appear so that the transformer can detect and learn the RPM patterns on its own. Manual engineering of representation, attempted here by the Row tokenizer, which directly juxtaposes panels as image channels, does not necessarily help—even though Row models ranked top at predicting properties of masked panels, they underperformed at classifying answer panels.

This study demonstrated that rephrasing a learning task in a multi-dimensional, multi-label fashion can be beneficial for generalization. RPM tasks are in a sense 'closed', as the space of responses expected from the model is narrowed down to eight provided answer panels. Compared to this, learning to classify the properties of visible panels and to predict the properties of masked-out panels is more open-ended, and in a sense generative. It forces the model to derive more detailed patterns from the data and, consequently, leads to better generalization. Moreover, by predicting properties, one may mitigate the

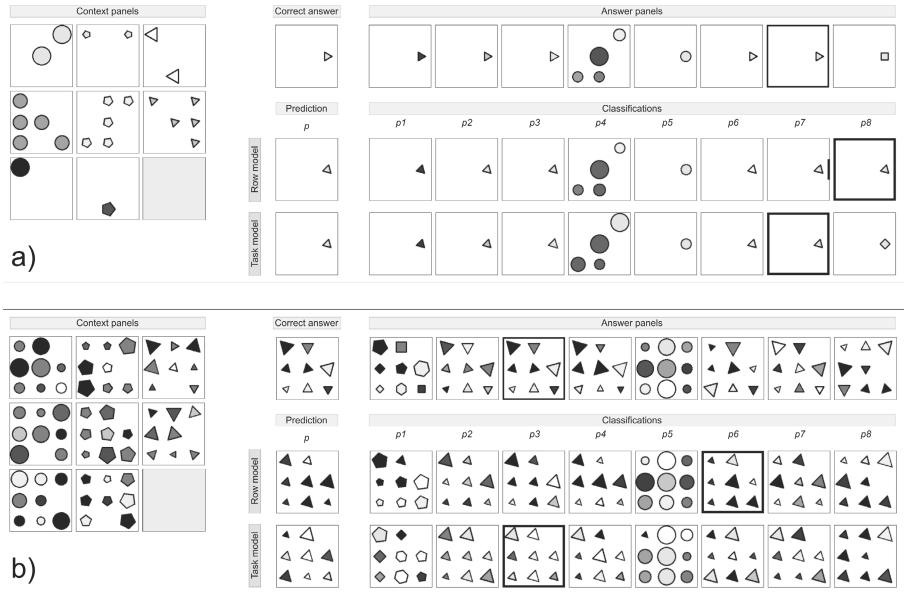

Fig. 3. Solving two RPM tasks (a) and (b) with the Task and Row models (both trained in the Combined masking mode). Left: the task. Middle: the correct answer and rendering of models' predictions p (Step 1 of the DCM) for the Row and Task models. Right: answer panels and the renderings of classifications p_i generated by the Row and Task models (Step 2 of the DCM). The panels corresponding to the most similar property vectors marked with thicker borders. Predictions and classifications are rendered from property vectors produced by the model while fixing rotation angles, as the angle was irrelevant in these tasks. To facilitate analysis, we render the *color* property using pseudocoloring (it is conventionally rendered in grayscale). See Figs. SM5–SM7 in Supplementary Material for more examples.

biases inadvertently introduced in benchmarks (Table 7). Last but not least, tracing the process of classifying and predicting properties provides interesting insights (Fig. 3).

In mapping an image to a sequence of tokens, the models considered here form an interesting middle ground between purely symbolic approaches and conventional deep learning, in a spirit similar to the vision transformer architecture proposed by Dosovitskiy *et al*. (2020) and neuro-symbolic systems. It is interesting to see that the transformer blueprint is helpful also when approaching a problem that is more abstract than conventional image classification. As evidenced in the presented results (in particular by the failure of the denseformer models; Section 5), explicit 'perceptual chunking' of representation provided by tokenization and the subsequent contextual reasoning realized with query-key interactions in the transformer allow learning the abstract patterns necessary to predict the missing panel and determine the right answer panel.

7. Conclusion and future work

We have shown that the proposed approach of solving RPM tasks by learning to predict the properties of panels outperforms state-of-the-art models trained to choose answer panels and avoids the biases present in training data. The models fare well despite flattening the 2D structure of the puzzle and can be inspected to a greater extent than end-to-end neural models. In future research, we will consider making the choice makers trainable alongside the model, to allow them to adapt to the deficiencies of classification (identified in Section 6 and exemplified in Fig. 3) and so enable further improvements.

The explicit partitioning of the inference process into property prediction and choice of an answer panel with the DCM can be seen as a special case of *task decomposition*, with the properties predicted and classified in the first stage acting as subgoals. In this study, we exploited the subgoals available in the RAVEN benchmark. Prospectively, it would be interesting to synthesize subgoals automatically.

Acknowledgment

The authors acknowledge support by TAILOR, a project funded by the EU Horizon 2020 research and innovation program under the GA no. 952215 and by the Polish Ministry of Science and Higher Education under the grant no. 0311/SBAD/0740.

References

Barrett, D., Hill, F., Santoro, A., Morcos, A. and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks,

in J. Dy and A. Krause (Eds), *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 80, PMLR, Cambridge, pp. 511–520.

Benny, Y., Pekar, N. and Wolf, L. (2021). Scale-localized abstract reasoning, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, USA*, pp. 12557–12565.

Bongard, M. (1970). *Pattern Recognition*, Spartan Books, Baltimore.

Defays, D. (1995). Numbo: A study in cognition and recognition, https://www.researchgate.net/publication/262363566_Numbo_a_study_in_cognition_and_recognition.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database, *2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, USA*, pp. 248–255.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale, *arXiv:* 2010.11929.

Hahne, L., Lüddecke, T., Wörgötter, F. and Kappel, D. (2019). Attention on abstract visual reasoning, *CoRR:* abs/1911.05990.

Hofstadter, D.R. (1995). *Fluid Concepts & Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Basic Books, New York.

Hu, S., Ma, Y., Liu, X., Wei, Y. and Bai, S. (2020). Hierarchical rule induction network for abstract visual reasoning, https://www.researchgate.net/publication/339324056_Hierarchical_Rule_Induction_Network_for_Abstract_Visual_Reasoning.

Hu, S., Ma, Y., Liu, X., Wei, Y. and Bai, S. (2021). Stratified rule-aware network for abstract visual reasoning, *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1567–1574, (virtual).

Kim, Y., Shin, J., Yang, E. and Hwang, S.J. (2020). Few-shot visual reasoning with meta-analogical contrastive learning, *in* H. Larochelle *et al.* (Eds), *Advances in Neural Information Processing Systems*, Vol. 33, Curran Associates, Inc., Red Hook, pp. 16846–16856.

Lei Ba, J., Kiros, J.R. and Hinton, G.E. (2016). Layer normalization, *arXiv:* 1607.06450.

Luo, W., Li, Y., Urtasun, R. and Zemel, R. (2017). Understanding the effective receptive field in deep convolutional neural networks, *arXiv:* 1701.04128.

Małkiński, M. and Mańdziuk, J. (2022a). Deep learning methods for abstract visual reasoning: A survey on Raven's progressive matrices, *arXiv*: 2201.12382.

Małkiński, M. and Mańdziuk, J. (2022b). Multi-label contrastive learning for abstract visual reasoning, *IEEE Transactions on Neural Networks and Learning Systems* **35**(2): 1941–1953, DOI: 10.1109/TNNLS.2022.3185949.

Raven, J.C. (1936). *Mental Tests Used in Genetic, the Performance of Related Individuals on Tests Mainly Educative and Mainly Reproductive*, MSc thesis, University of London, London.

Spratley, S., Ehinger, K. and Miller, T. (2020). A closer look at generalisation in Raven, *Computer Vision, ECCV 2020: 16th European Conference, Glasgow, UK,* pp. 601–616, DOI: 10.1007/978-3-030-58583-9_36.

Tan, M. and Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks, *in* K. Chaudhuri and R. Salakhutdinov (Eds), *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 97, PMLR, Cambridge, pp. 6105–6114.

Tan, M. and Le, Q.V. (2021). EfficientNetV2: Smaller models and faster training, *in* M. Meila and T. Zhang (Eds), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, Proceedings of Machine Learning Research, Vol. 139, PMLR, Cambrige, pp. 10096–10106.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I. (2017). Attention is all you need, *in* I. Guyon *et al.* (Eds), *Advances in Neural Information Processing Systems*, Vol. 30, Curran Associates, Inc., Red Hook.

Wu, Y., Dong, H., Grosse, R.B. and Ba, J. (2020). The scattering compositional learner: Discovering objects, attributes, relationships in analogical reasoning, *CoRR:* abs/2007.04212.

Zhang, C., Gao, F., Jia, B., Zhu, Y. and Zhu, S.-C. (2019a). Raven: A dataset for relational and analogical visual reasoning, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, USA*, pp. 5312–5322.

Zhang, C., Jia, B., Gao, F., Zhu, Y., Lu, H. and Zhu, S.-C. (2019b). Learning perceptual inference by contrasting, *in* H. Wallach *et al.* (Eds), *Advances in Neural Information Processing Systems*, Vol. 32, Curran Associates, Inc., Red Hook.

Zhuo, T. and Kankanhalli, M.S. (2021). Effective abstract reasoning with dual-contrast network, *9th International Conference on Learning Representations, ICLR 2021,* (virtual).

Jakub Kwiatkowski is a PhD student of computer science at the Poznan University of Technology. He works in the ICT Security Department at the Poznan Supercomputing and Networking Center, where he explores the applicability of machine learning in cybersecurity. His research interest is focused on deep learning, computer vision, representation learning, abstract visual reasoning and explainable AI.

Krzysztof Krawiec holds PhD and habilitation degrees from the Poznan University of Technology, Poland. His main research areas are evolutionary and coevolutionary computation, genetic programming, neurosymbolic systems, and applications in medical imaging. He is an associate editor of *Genetic Programming and Evolvable Machines* and the author of *Behavioral Program Synthesis with Genetic Programming* (Springer, 2016). More details are available at www.cs.put.poznan.pl/kkrawiec.

Supplementary material

The supplementary material is available online at https://arxiv.org/abs/2308.06528. It covers the technical details of the method and its software implementation, a discussion of the loss function, the analysis of variance and stability of the results, the analysis of the structure of errors, learning curves, and more visualizations of models' responses and choices. It also includes a color version of Fig. 3.

Received: 11 August 2023
Revised: 7 December 2023
Accepted: 5 February 2024