Neuro-symbolic Approaches for Abstract and Relational Visual Reasoning Tasks

Zhiliang Xiang

School of Computer Science and Informatics

Cardiff University

MSc Advanced Computer Science

Supervisor: Dr Víctor Gutiérrez Basulto

September 2022

Table of Contents

List of Figures 2
1 Introduction 3
1.1 Background 3
1.2 Objectives and Contributions 4
1.3 Outline 6
2 Related Work 6
2.1 Computational Attempts on RPM Problem 6
2.2 Neuro-Symbolic Systems 8
2.3 Disentangled Representations 8
3 Problem Description 9
3.1 Raven Progressive Matrix 9
3.2 General Solving Scheme Used by Human 10
3.3 Problem Formulation 11
4 Preliminaries 12
4.1 Variational Auto-Encoder and Attend-Infer-Repeat 12
4.2 Answer Set Programming (ASP) 14
4.3 NeurASP 15
5 A NeurASP Approach to RPM 18
5.1 Observation 18
5.2 A Row-level Representation? 19
5.3 The Challenge 19
6 Methodology 20
6.1 A Finer-grained Representation at Row-level 20
6.2 Overview of the Architecture 21
6.3 Preprocessing for Disentangled Representation 22
6.4 Row Embedding Aggregation and Multi-task Rule Classification 23
6.5 Learning with Background Knowledge 25
6.6 Neuro-symbolic Inference 29
7 Experiments 30
7.1 Dataset 30

7.2 The AIR Module 32
7.3 The Main Pipeline 32
7.4 Results of the Main Pipeline 35
8 Future Work and A Potential Solution 37
8.1 Future Work 37
8.2 A Potential Solution 38
9 Conclusion 39
10 Reflection 39

List of Figures

1 RPM Example 5
2 Solving Procedure of Humans 10
3 Three Abstract Relations 12
4 Graphical model of VAE 13
5 Row-level Hierarchy 21
6 General Pipeline 22
7 Process of AIR 24
8 Multi-task Rule Classification Module 26
9 A Sample of the RAVEN Dataset 31
10 ELBO Loss 33
11 Qualitative results of AIR module 33
12 Potential Future Work 38

Abstract

It has been long regarded that abstract relational reasoning is fundamental to human intelligence and a critical component for the development of human-level AI systems. A well-studied and reliable estimation of human's capacities w.r.t. abstract reasoning is Raven Progressive Matrices (RPM) Test. Given that performances on the RPMs have predictive validity, understanding individual performances opens a gate to the essence of intelligence. To better analyse the performances, both symbolic and deep learning (DL) approaches have been proposed. However, existing proposals either *ignore the challenges raised by visual perception (symbolic)* or *suffer from statistical biases and lack of explainability (DL models)*. While recent developments of Neuro-Symbolic frameworks, e.g. NeurASP, have combined the strengths of both the symbolic system and DL models as complementaries, to what extent these frameworks can be applied to abstract reasoning still *remain unexplored*. To fill up this empty gap, we explore potential ways to apply NeurASP to solve the RPM task and develop a prototype framework as the first attempt. Although experiment results are not as competent as the state-of-the-arts, we pinpoint several assumptions of NeurASP to be relaxed for more suitable solutions to abstract reasoning. We also highlight the key challenges and future directions to develop NeurASP-based systems with the capacity of abstract reasoning, then blueprint one of the potential solutions. The prototype system and experiments have been open-sourced and available at: `https://tinyurl.com/diss-xzl`.

Keywords. *Abstract Reasoning*; *Artificial Intelligence (AI)*; *Neuro-Symbolic AI*; *Neural Probabilistic Logic Programming*.

Acknowledgements

I would like to express my greatest gratitude to my parents for the trusts and wholehearted supports they have been giving to me for pursuing further studies abroad. Without the supports from them, I would not have the opportunities to explore what I truly like to devote as career.

I am also really grateful to my supervisor Dr Víctor Gutiérrez Basulto for identifying such an interesting topic and all the guidance he has been giving. It's has been a fruitful year for my scientific growth, the inspirations that he has been sharing selflessly makes these little achievements possible.

Lastly, I would like to thank the ARCCA team for providing computational resources and technical supports during the development.

1 Introduction

1.1 Background

Abstract Reasoning and Raven Progressive Matrices. Abstract reasoning refers to the mechanism of human cognation for generalising about high-level concepts and relating variations of attributes above concrete objects. Such capabilities for abstraction and relational reasoning allow human to deal with novelty, i.e., to adapt one's thinking to a new cognitive problem. It has been long regarded by both cognitive psychology and artificial intelligence (AI) communities that abstract relational reasoning is fundamental to human intelligence (Carpenter et al., 1990; Meo et al., 2007; Falkenhainer et al., 1986; Hofstadter and Mitchell, 1994) and a critical component for the development of human-level AI systems (McCarthy et al., 2006).

A well-studied and reliable estimation of human's capacities w.r.t. abstract reasoning is *Raven Progressive Matrices (RPM) Test* (John and Raven, 2003). Simple yet effective, RPM has been extensively applied as IQ test by psychology community to a wide range of populations (Meo et al., 2007). In an RPM test, the test-taker is presented with a 3×3 matrix with the bottom right panel left blank as shown in Figure 1. The goal is to complete the matrix by choosing one of the 8 candidates listing in answer set, such that visual features of the figural elements occurring in each row/column align with the same "underlying patterns" of variation. This requires test-takers to identify the correspondences between among figural elements and reason about the underlying relations applied to them. For instance, all vertical elements in Figure 1 quantitatively follow a distribution of numbers distinctively picking from the set $\{1, 2, 3\}$ in a row-wise manner. However, difficulties of RPMs substantially raise as the visual complexities and number of underlying rules increased (Carpenter et al., 1990; Meo et al., 2007). Such characteristics make RPM test a strong diagnosis of abstract and relational reasoning abilities, distinguishing even among highly educated subjects (Meo et al., 2007; Santoro et al., 2018).

Computational Approaches to RPMs. Given that performances on the RPMs have predictive validity, understanding the causal relationship between individual differences and test performances opens a gate to the understanding of the essence of intelligence (Meo et al., 2007). Symbolic computational simulation models were therefore proposed for a better analysis of the nature of individual differences (Carpenter et al., 1990; Lovett et al., 2007; Lovett and Forbus, 2017). While symbolic systems are explainable and can achieve human-like performances, they rely heavily on explicitly encoded rules and *oversimplified assumptions w.r.t. visual inputs* (Lovett and Forbus, 2017; Mitchell, 2021). Nevertheless, the rule taxonomy identified by (Carpenter et al., 1990) inspired a wealth of research tackling automatic RPM problem

generation systems (Matzen et al., 2010; Wang and Su, 2015; Santoro et al., 2018; Zhang et al., 2019a), providing abundant high-quality RPMs for both psychological practice and the development of AI agents for abstract reasoning.

With the large amount of generated data available and the paradigm shift of AI approaches, there have recently been numerous attempts to get deep learning (DL) models to learn relation abstraction via the RPM test (Santoro et al., 2018; Zhang et al., 2019a,b; Wang et al., 2020; Hu et al., 2021; Yu et al., 2021; Zhang et al., 2021). Although the capacities w.r.t. low-level perception (Steenbrugge et al., 2018; van Steenkiste et al., 2019; Spratley et al., 2020) and generalising abstract relations (Santoro et al., 2018; Hu et al., 2021; Yu et al., 2021) of DL models seem fascinating, they come with *costs of interpretability and data efficiency* (Manhaeve et al., 2021b; Mitchell, 2021). More importantly, training on procedurally generated data examples allows DL models to exploit potentially *statistical biases*, resulting to *unfaithful performances* (Santoro et al., 2018; Spratley et al., 2020; Hu et al., 2021; Mitchell, 2021). It therefore remains unclear what DL models have learnt hold faithfully the abilities of abstract and relational reasoning.

Neuro-Symbolic Frameworks. In another venue, there has been an increasing interest in the development of Neuro-Symbolic systems that take into account the strengths of both the DL models, e.g. visual perception, with those of symbolic systems, e.g. structural reasoning as complementaries (d'Avila Garcez et al., 2019; Manhaeve et al., 2021b). Two prominent examples of Neuro-symbolic systems are NeurASP (Yang et al., 2020) and DeepProbLog (Manhaeve et al., 2021a), that is the formalism of neural probabilistic logic programming (NPLP). Being able to make full use of both the worlds of symbolic systems and DL models, NPLP *enables symbolic systems the ability of visual perception* and *provides DL models the capabilities of explainable logic reasoning*. Moreover, the intuition of combining neural perception and logic reasoning aligns well with the "thinking fast and slow" cognitive model (Kahneman, 2011; Booch et al., 2021) , offering a well-suited test bed for abstract visual reasoning. Surprisingly, studies regarding the abilities of abstract reasoning of existing NPLP frameworks remain *to-date an empty gap*.

1.2 Objectives and Contributions

Given that the NPLP paradigm has not only well-aligned intuition with cognitive models but also the advantages in combining both symbolic systems and DL models, we are interested in its capabilities w.r.t. abstract visual reasoning. Specifically, the main aim of this project is to explore to what extend the Neuro-Symbolic frameworks (Yang et al., 2020) can be applied to solve abstract visual reasoning tasks such as

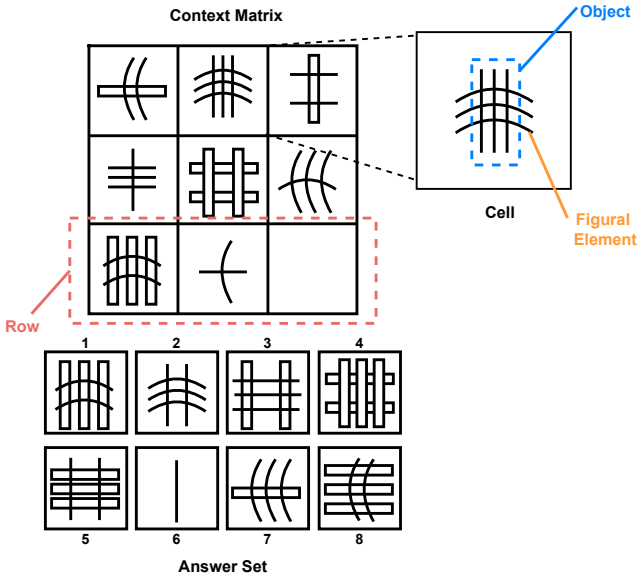


Figure. 1: An example problem of Raven's Progressive Matrix(Carpenter et al., 1990) (5 is the correct answer)

Raven's Progressive Matrices. To this aim, we first observe the pitfalls w.r.t. scalability and expressiveness of NeurASP led by the assumptions on output of neural network (NN) and the nature of symbolic systems. We explore potential ways to overcome the problems and propose a prototype framework using NeurASP by introducing *scalable and expressible representations under restricted assumptions*.

In particular, we capture the low-level variations of figural elements and relations between cells by row-level embeddings, lifting the innumerable combinations of attributive values of figural elements. While the row-level representations may lack of low-level information of individual elements, we utilise a pretrained AIR module (Eslami et al., 2016) to provide NNs with a scene-disentangled object-centric bias (Spratley et al., 2020). To get around the inability of NeurASP w.r.t. accepting mutually non-exclusive probabilistic input, we introduce a component-attribute-wise symbolic representation using ASP encoding, then establish one-to-one correspondences between row embeddings and underlying rules via a multi-task classification network (Zhang et al., 2014). To examine the effectiveness of our approach, we experiment the prototype system on the I-RAVEN dataset (Hu et al., 2021), comparing with two state-of-the-art models. Although results shown are not as competent, we analyse possible reasons with respect to both the proposed methodology and restrictions of NeurASP framework. On top of the reasons, we discuss the key challenges and future directions to develop AI systems with the capacity of abstract relational reasoning.

1.3 Outline

The remainder of this report is organised as follows. The next section reviews existing works on symbolic and DL approaches to the RPM task, and surveys relevant techniques to be adopted. Section 3 gives a detailed descriptions of the RPM problem with formal definitions. Section 4 introduces briefly the key concepts will be used in methodology. Section 5 discusses the preliminary observations and underlines potential pitfalls of applying NeurASP to the RPM task. Based on the observations, Section 6 proposes an intermediate representation of the problem and specifies the pipeline of the prototype system. Section 7- 8 gives details of the evaluations and analyses potential reasons contribute to the results. Finally, Section 9 concludes the report and highlights challenges for the future work, then enclosed by a scratch design for one possible solution.

2 Related Work

2.1 Computational Attempts on RPM Problem

Provided the extensive use of the RPM test and its centric role in measuring abstract reasoning abilities, to simulate and understand better the cognitive processes (Meo et al., 2007), many efforts have been taken for the development of automatic RPM solvers. Computational models in this regard can be roughly categorised into symbolic approaches and deep learning approaches.

Symbolic Approaches. Symbolic methods typically represent visual of figural elements into a set of symbolic representations and then infer corresponding rules by instantiating explicitly encoded rule formulae (Carpenter et al., 1990; Lovett et al., 2007; Lovett and Forbus, 2017). In a key study on this direction, Carpenter et al. (1990) identified a rule taxonomy and proposed theoretical models of the cognitive processes used in solving RPM test. Based on the taxonomy, computational simulation systems were also developed where symbolic representations of figural elements were directly encoded by human. Indeed, predetermining hand-coded representations of visual input is an oversimplification considering what information in each cell should be or should not be perceived is ignored (Lovett et al., 2007; Lovett and Forbus, 2017). To address the limitations regarding symbolic visual mapping, the Structure-Mapping Engine (SME) (Falkenhainer et al., 1986) is used in (Lovett and Forbus, 2017). Empowered by SME, the system can effectively capture underlying rules implied through comparing the set of predicate logic descriptions of each manually segmented objects. Yet still, low-level visual perception was not addressed at all (Mitchell, 2021). Nevertheless, the rule taxonomy identified by Carpenter et al. (1990) inspired a wealth of research tackling automatic RPM problem generation systems (Matzen et al., 2010; Wang and

Su, 2015; Santoro et al., 2018; Zhang et al., 2019a). Notably, Wang and Su (2015) introduced an abstract representation of RPMs formalised in first-order logic (FOL). By taking advantage the expressive power of FOL, the proposed formulation can not only capture the irregularly varied patterns of RPMs, but also formal notions of well-formedness of RPMs. By contrast, symbolic formulation of our work borrows the notions of RPM well-formedness in (Wang and Su, 2015), but the notions are formulated in answer set semantics (Calimeri et al., 2020) rather than full FOL.

Deep Learning (DL) Approaches. In general, DL approaches excel in incorporating information from low-level visual inputs (Manhaeve et al., 2021b). Task-specific NNs are often adopted to introduce inductive bias, informing NNs about the abstract relations (Santoro et al., 2018; Zhang et al., 2019a; Spratley et al., 2020; Hu et al., 2021). As a pioneer to explore solving advanced RPMs via DL models, Santoro et al. (2018) proposed a variant of the relational neural model (Santoro et al., 2017). The proposed model learns to generalise the relations based upon convolutional representations of a problem matrix and an individual answer choice pair. While it appears fascinating the abilities to learn abstract relations, the use of DL models comes at the costs of lacking interpretability and low data efficiency (Manhaeve et al., 2021b; Mitchell, 2021). To meet the demands of large-size training data, Santoro et al. (2018) took inspirations from (Wang and Su, 2015), synthesised a large-scale RPM-like dataset, namely the Procedurally Generated Matrices (PGM) dataset. Similarly, Zhang et al. (2019a) introduced the RAVEN dataset based on Attributed Stochastic Image Grammar (A-SIG). Compared to PGM, RAVEN is claimed to be visually less complex but more complex w.r.t. underlying abstract relations (Zhang et al., 2019a; Spratley et al., 2020). The establishment of these two major datasets encourages many following studies to investigate neural models' abilities in the abstract reasoning task (Zhang et al., 2019b; Zheng et al., 2019; Hu et al., 2021; Yu et al., 2021). Although attempts to model hierarchical relations of RPMs using neural embeddings have given some performance boosts, DL methods still show inadequacies when underlying abstract relations get more complex (Hu et al., 2021; Yu et al., 2021). Moreover, DL models potentially suffer from subtle statistical bias residing in procedurally generated data (Mitchell, 2021). Both (Hu et al., 2021) and Spratley et al. (2020) have reported that due to the defect of answer set generation of RAVEN dataset, DL models can generalise to solve the RPMs without looking at answer sets, such statistical shortcut can lead to unfaithful performances. Although a fairer version (Hu et al., 2021) of RAVEN dataset is proposed, it still remains unclear what DL models have learnt provided the lack of transparency (Mitchell, 2021).

2.2 Neuro-Symbolic Systems

With the recent raising concerns about the interpretability and accountability of neural models in AI community, there are increasing interests on Neuro-Symbolic AI. This paradigm brings the advantages of learning in neural networks, reasoning and intepretability of symbolic representation together (Besold et al., 2021; d'Avila Garcez et al., 2019). While there are different formalisms within the paradigm, prominent approaches include symbolic constraints to regularise neural models (Xu et al., 2018), templating neural networks using logic (Rocktäschel and Riedel, 2017; Wang et al., 2019). Notably, by bridging DL models and probabilistic logic programming (PLP) (Dries et al., 2015; Lee et al., 2017), the neural probabilistic logical programming (NPLP) formalism (Manhaeve et al., 2021a; Yang et al., 2020) treats the output of DL models as the probability distribution over atomic facts in PLP, exploiting the full strength of DL models and symbolic systems. This intuition of taking the strength from both DL models, e.g. low-level visual perception, and those of symbolic systems, e.g. structural reasoning as complementaries aligns well with the "thinking fast and slow" cognitive model (Kahneman, 2011; Booch et al., 2021). With the rich background knowledge provided by logic programs, DL models in NPLP frameworks have shown great potentials in learning visual reasoning task such as Sudoku (Manhaeve et al., 2021a; Yang et al., 2020) and common-sense reasoning on image scenes (Yang et al., 2020). Given the well-suited test beds of abstract visual reasoning offered by NPLP frameworks, it is a surprise that the topic remains unexplored to-date.

2.3 Disentangled Representations

While learning good representations of high-dimensional perceptual data has been a fundamental goal of DL (Bengio et al., 2013), there has recently been research attentions focused on learning representations that are *disentangled* (Bengio et al., 2013; Kingma and Welling, 2014; Eslami et al., 2016; Steenbrugge et al., 2018; van Steenkiste et al., 2019). The intuition is to model lower-dimensional neural representations as the explanatory factors of variations in data, while keeping each distinct factor in isolation (Steenbrugge et al., 2018). It is often expected that disentangled representations could provide more effective representations of the data and thus help to learn more data-efficient and robust representations (van Steenkiste et al., 2019). Particularly, variational auto-encoder (VAE) based methods (Kingma and Welling, 2014; Eslami et al., 2016) have been shown effective on visual relational reasoning tasks (Steenbrugge et al., 2018; van Steenkiste et al., 2019; Spratley et al., 2020). These methods are able to learn compact latent vectors of visual sensory data, explicitly capturing the attributive variations of figural elements in visual relational reasoning tasks (Steenbrugge et al., 2018; van Steenkiste

et al., 2019). Notably, Spratley et al. (2020) leverages the "*Attend-Infer-Repeat*" (AIR) framework (Eslami et al., 2016) to disentangle figural elements in RPMs, informing DL models with objectness and spatial information as inductive biases. Indeed, spatial and attributive information of figural elements are crucial to induce the underlying rules (Lovett et al., 2007). In light of this, our work will adopt an AIR module as a preprocessing component prior to the main pipeline.

3 Problem Description

3.1 Raven Progressive Matrix

An RPM problem consists of a 3×3 *problem matrix*, in which the bottom right *entry* is absent, test takers are required to choose one of the eight candidates in the *answer set* below the problem matrix for the empty entry. Each of the entries consists of one to five figural elements, for example shapes, shadings and lines etc. In order to determine the empty entry, test takers would need to be able to correctly identify the underlying relations among entries and rows depending solely on visual analogy, then reason about which candidate could satisfy the rules that both of the first two rows/columns follow.

Example 1. Figure 1 shows an RPM problem matrix that contains entries of 3 types of figures (line, curve and rectangle), to identify the corresponding rules one may need to make several assumptions and try them out one by one. For example, one may first heuristically hypothesise that figures in the same shape are governed by a same rule. Once the rule has been tried out, it became obvious that both the number and orientation of the figures vary unsystematically. In this case, one must keep trying other possible assumptions until the correct correspondences are found. In fact, horizontal and vertical figures in each row all follow two rules, that are, the distribution of numbers of identical figures (one, two and three) and the distribution of distinct shapes (line, curve and rectangle) present horizontally/vertically.
As such, the difficulty of solving a RPM problem varies in accordance with the complexity of identifying corresponding rules and the number of underlying rules. A study conducted by Meo et al. (2007) showed that as the difficulty raises, the percentage of correct attempts falls significantly from 85% to 30% among 80 university students. The RPM test is therefore, a strongly diagnostics in terms of abstraction and reasoning abilities, discriminating even among test takers from higher education background Carpenter et al. (1990); Meo et al. (2007). Solving RPM problems has therefore been a long-standing challenging task in the AI community to the degrees of the difficulty in developing a machine with the ability to reason abstractly as well as the complexity of the problem itself (Hu et al., 2021).

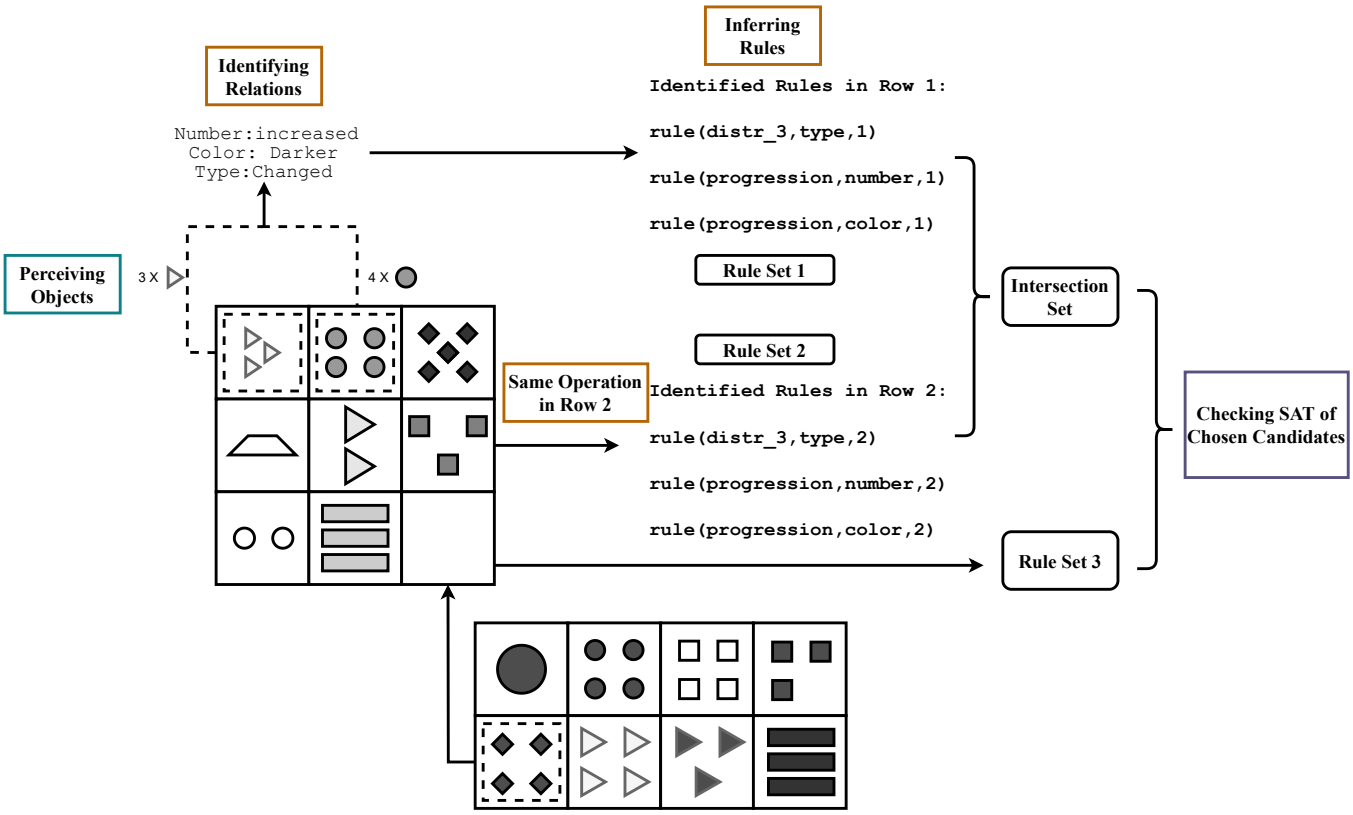


Figure. 2: General outline of incremental, reiterative solving procedure of human test-takers

3.2 General Solving Scheme Used by Human

As revealed in Carpenter et al. (1990), one typical strategy adopted by test-takers to solve RPM problems in the experiment is so called incremental, reiterative representation and rule induction scheme. The general outline of the solving procedure is shown as Figure 2, the figural objects in the first row are first perceived cell by cell, attributes of corresponding objects are then compared in order to identify relations between each cell. After the patterns of similarities and differences are compared, dominant patterns in the first row could be derived as instances of rules. The second row is processed in an akin way, and the set of rules identified in the second row are subsequently taken intersection with the rules showing in the first row. The intersections are then applied to the third row with a chosen candidates filling in the missing cell. Candidates who are failed to contribute to a third row that satisfies all the dominant rules from the first two rows will be weed out. The *goal* of our method is to build an automatic RPM solving model via Neuro-symbolic technique to simulate this abstract reasoning strategy used by human.

3.3 Problem Formulation

Formally, following the FOL formulation presented by Wang and Su (2015), we denote *objects* and *attributes* finite sets of constants $\mathcal{O}$ and $\mathcal{A}$ respectively. We shall make clear distinctions between *object* and *figural element*. While a figural element is a single visual pattern that picked from an alphabet, objects can be considered as a set of multisets with elements from the alphabet whose attributive variations together contribute to a rule. Clearly, it is non-trivial that to identify which figural elements should be "grouped" together by comparison, but we assume that has been given as we are operating at a higher level (see section 5). For every pair of (o, α) where $o \in \mathcal{O}$ and $\alpha \in \mathcal{A}$, there exists a function $v : (\mathcal{O}, \mathcal{A}) \rightarrow \mathcal{D}$, where $\mathcal{D}$ is a finite set of value domain, that is, $v(o, \alpha)$ indicates attribute α of the figural element o is assigned with a value picked from the value domain $\mathcal{D}$. We follow a conventional notation R/k to indicate a predicate R has arity k. By a tri-ary predicate $p(o, i, j)$, we represent an object o locates on the i-th row and j-th column of the *context matrix* of an RPM problem, where $i, j \in \{1, 2, 3\}$.

Definition 1. *Denote the generalisation of underlying variation pattern with the ternary predicate R, an underlying **rule** to a row of the context matrix of an RPM is defined as following:*

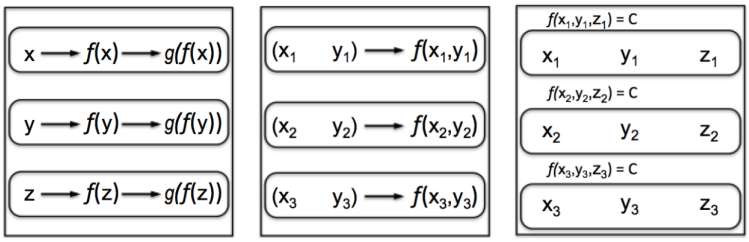
$$\exists a \in \mathcal{A}, \forall i, \exists o_1, o_2, o_3 \in \mathcal{O}.(p(o_1, i, 1) \wedge p(o_2, i, 2) \wedge p(o_3, i, 3) \wedge R(v(o_1, \alpha), v(o_2, \alpha), v(o_3, \alpha)) \quad (1)$$

$$\begin{aligned} \forall a, o_1, o_2, o_3.(R(v(o_1, a), v(o_2, a), v(o_3, a)) \leftrightarrow \\ unary(v(o_1, a), v(o_2, a), v(o_3, a)) \vee binary(v(o_1, a), v(o_2, a), v(o_3, a)) \vee \\ ternary(v(o_1, a), v(o_2, a), v(o_3, a)). \end{aligned} \quad (2)$$

While formula 1 constrains the occurrence of a rule R to be dependent on the assignments of a common attribute of objects in each cell of a row, formula 2 specifies what type of relation R could be. Namely, a rule R holds for an attribute α of objects $\{o_1, o_2, o_3\}$ from each cell of a row i if and only if the value assignments satisfy one or several of $\{unary, binary, ternary\}$ relations. Figure 3 illustrates how do the ternary relations interact with attributive values across a row, instantiations of these relations will be discussed in section 6.

Definition 2. *Denote the set of Rs that hold for in the first 2 row of a context matrix, that is, $R_1 \cap R_2$, we consider an **RPM constraint** if $\phi \in R_1 \cap R_2$ satisfied the following:*

$$\begin{aligned} \exists o_1, o_2, o' \in \mathcal{O}, i \in \{1, ..., 8\}. \\ (p(o_1, 3, 1) \wedge p(o_2, 3, 2) \wedge ans(o', i) \wedge \exists \alpha.\phi(v(o_1, \alpha), v(o_2, \alpha), v(o', \alpha))), \end{aligned} \quad (3)$$



(a) Unary (b) Binary (c) Ternary

Figure. 3: Illustration of the 3 abstract relations (Wang and Su, 2015)

where the predicate $ans/2$ *indicate objects occurring at the* i*-th position of an* answer set.

As a result, formula 3 filters out a set of rules ϕ that hold for value assignments of an attribute α of objects from the first 2 rows, but do not assert true for α of objects in a third row completed by choosing the candidate i as an answer. We denote the set of RPM constraints in an RPM problem by C_R, the aim of an RPM problem is therefore picking a candidate from answer set that contributes to a third row where the most of relations in C_R hold. For simplicity, we denote

$$Ans(i) = \{\phi(v(o_1, \alpha), v(o_2, \alpha), v(o', \alpha)))|(p(o_1, 3, 1), p(o_2, 3, 2), ans(o', i), \alpha \in \mathcal{A}, \phi \in C_R\}, \quad (4)$$

namely, $Ans(i)$ is the subset of C_R that contains the set of RPM constraints satisfied also by the third row when selecting the i-th candidate as an answer. Therefore, for a given RPM problem, we are aiming at identifying the i such that $Ans(i) \not\subseteq Ans(j)$ holds for every $Ans(j)$ where $i, j \in \{1, ..., 8\}$ and $i \neq j$.

4 Preliminaries

Here we briefly introduce key concepts relevant to the AIR model (Eslami et al., 2016), answer set programming (ASP) (Gebser et al., 2012) and NeurASP (Yang et al., 2020) as preliminaries to understand the proposed method , we refer the reader to the references for more details.

4.1 Variational Auto-Encoder and Attend-Infer-Repeat

Variational Autoencoder (VAE) (Kingma and Welling, 2014) consists of an *encoder network* and a *decoder network*. Given an encoder network parameterised by ϕ is trained to produce vectors for the mean μ and the standard deviation σ of the distribution function for sampling lower-dimensional latent factors $\boldsymbol{z}$ given input $\boldsymbol{x}$, namely $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Then the decoder network parameterised by θ could be characterised as

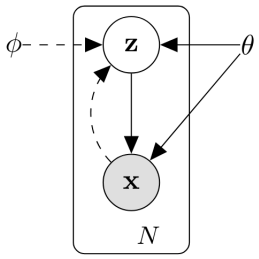


Figure. 4: Graphical Model of VAE (Kingma and Welling, 2014)

a generative model that learns to reconstruct $\boldsymbol{x}$ for each given $\boldsymbol{z}$, that is, $p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$. Learning objective of such models is to maximise the *evidence lower bound (ELBO)*, formulated as follow:

$$\mathcal{L}(\boldsymbol{x};\theta,\phi) = \mathbb{E}_{q_\phi(\boldsymbol{z})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - KL[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))],$$

Intuitively, the ELBO loss contributes to the minimisation of two factors: 1) the reconstruction loss, modelled by the decoder network (left-hand side of subtraction); 2) the KL divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and a hypothesised, e.g. Gaussian, distribution $p(\mathbf{z})$ for sampling latent variables (right-hand side of subtraction). In this way, the latent space is encouraged to be an information-rich latent representation that allows for smooth interpolation between samples (Spratley et al., 2020).

Since the latent representations produced by VAEs are generally unstructured and lack of interpretability, the Attend-Infer-Repeat framework (AIR) (Eslami et al., 2016) is proposed to follow the intuition of structured perception of human vision system to improve VAE. For an input scene image, AIR attends an object within the scene using attention windows constructed by spatial transformers, and encoding it into a structured embedding (representing the what the object is, and the position and scale of the object), which is subsequently fed to a decoder network to reconstruct the image scene by adding the decoded object into an empty scene for each iteration, until all the objects are assembled. As such, the learnt representations of AIR are able to further disentangle each object along with its crucial properties in a scene image. Inspired by Spratley et al. (2020), we will leverage such virtue of AIR to preprocess our dataset, in this way latent representations of position and scale of figural objects in each RPM image can be obtained.

4.2 Answer Set Programming (ASP)

ASP is a form of the stable model (answer set) semantics based logic programming which aims at solving difficult, primarily NP-hard, search problem by computing answer sets of logic programs (Lifschitz, 2008).

Basic Notion. In this report, we consider extended logic programs allowing for disjunctive in heads of rules (Gebser et al., 2012). A *rule* r is of the following form:

$$H \leftarrow B_1, ..., B_m, \sim B_{m+1}, ..., \sim B_n. \tag{5}$$

By $head(r) = H$ and $body(r) = \{B_1, ..., B_m, \sim B_{m+1}, ..., \sim B_n\}$, we denote the *head* and *body* of r, where "$\sim$" stands for default negation. The head H is a disjunction $a_1 \vee ... \vee a_k$ over *atoms* $a_1, ..., a_k$, belonging to some alphabet $\mathcal{A}$. Each body component B_i is either an atom or a `#sum` *(or weight) constraint* of the form $L\#\texttt{sum}[l_1 = w_1, ..., l_k = w_k]U$. In the latter $l_i = a_i$ or $l_i = \sim a_i$ is a *literal* and w_i a *non-negative* integer *weight* for $a_i \in \mathcal{A}$ and $1 \leq i \leq k$; L and U are integers providing a lower and an upper bound. Intuitively, a rule specifies the head H holds true, if all the conjunctions of body literals appearing on the right-hand-side of $\leftarrow$ are true. An *integrity constraint* is a rule r such that $head(r) = \bot$. If $body(r) = \emptyset$, r is called a *fact*. Sticking to formula 5, we let $head(r)^+ = \{a_1, ..., a_k\}$, $body(r)^+ = \{B_1, ..., B_m\}$, and $(L\#\texttt{sum}[l_1 = w_1, ..., l_k = w_k]U)^+ = [l_i = w_i | 1 \leq i \leq k, l_i \in \mathcal{A}]$. A special case of `#sum` constraint is the *cardinality constraint*, where the default weights 1 is omitted, that is, $L\#\texttt{sum}[l_1, ..., l_k]U$. For simplicity, we use the form $L\{l_1, ..., l_k\}U$ when referring a cardinality constraint. Specially, *choice rules* are those rules with cardinality constraints as heads and of the form

$$L\{a_1, ..., a_k\}U \leftarrow B_1, ..., B_m, \sim B_{m+1}, ..., \sim B_n, \tag{6}$$

expressing that the number of selections of the disjunctive heads is subject to the associated lower and upper bound. Another variant of the $\#\texttt{sum}$ constraint is the $\#\texttt{max}$ aggregate where literals with the greatest weights is derived. Choice rules and the $\#\texttt{max}$ aggregate are in fact transformed from normal rules defined as formula 5 detailed in (Gebser et al., 2012).

Intepretation and Stable Model. A (Herbrand) *interpretation* is represented by the set $X \subseteq \mathcal{A}$ of its entailed atoms. The satisfaction relation "$\models$" on rules r is defined as follows:

- $X \models \sim B$ if $X \not\models B$,

- $X \models (a_1 \vee ... \vee a_k)$ if $\{a_1, ..., a_k\} \cap X \neq \emptyset$,
- $X \models (L\#\mathtt{sum}[l_1 = w_1, ..., l_k = w_k]U)$ if $L \leq \sum_{1 \leq i \leq k, X \models l_i} w_i \leq U$,
- $body(r)$ if $X \models l$ for all $l \in body(r)$, and
- $X \models r$ if $X \models head(r)$ or $X \not\models body(r)$.

A *logic program* Π is a set of rules r, and X is a *model* of Π if $X \models r$ for every $r \in \Pi$. The reduct of the head H of r w.r.t. X is $H^X = \{a_1 \vee ... \vee a_k\}$ if $H = a_1 \vee ... \vee a_k$, and $H^X = atom(H^+) \cap X$ if $H = (L\#\mathtt{sum}[l_1 = w_1, ..., l_k = w_k]U)$. Furthermore, the reduct of some (positive) body element $B \in body(r)^+$ is $B^X = B$ if $B \in \mathcal{A}$, and $B^X = (L - \sum_{1 \leq i \leq k, l_i = \sim a_i, a_i \notin X} w_i)\#\mathtt{sum}B^+$ if $B = (L\#\mathtt{sum}[l_1 = w_1, ..., l_k = w_k]U)$. The *reduct* of program Π w.r.t. interpretation X is the following:

$$\Pi^X = \{H \leftarrow B_1^X, ..., B_m^X | r \in \Pi, X \models body(r), H \in head(r)^X, body(r)^+\}. \tag{7}$$

X is an *answer set* (or *stable model*) of Π s.t. no proper subset of X is a model of Π^X, that is, an answer set is a $\subseteq$*-minimal* model of its own reduct.

4.3 NeurASP

NeurASP is a simple and effective extension that bridges NNs and PLP by treating the output of neural networks as the probability distribution over atomic facts in answer set programs. It has been successfully used in solving visual reasoning task such as Sudoku solving and common-sense reasoning on image scenes etc.

Syntax. For a given neural network M that takes an arbitrary tensor input $\boldsymbol{x}$, we consider the output $M(\boldsymbol{x})$ a $\mathbb{R}^{e \times n}$ matrix that contains the probabilities of predicted results output by M, where e denotes the number of random events while n is the possible outcomes of each of the random events. The entry $M(\boldsymbol{x})[i, j](i \in \{1, ..., e\}, j \in \{1, ..., n\})$ is the probability of the j-th outcome of the i-th event. To express M in answer set programs, *neural atoms* are defined as form

$$nn(m(e, \boldsymbol{t}), [v_1, ..., v_n]). \tag{8}$$

where i) the reserved keyword nn indicates the presence of a neural atom; ii) m is the symbolic predicate that identifies the given neural network M; iii) $[v_1, ..., v_n]$ are the possible predicted results for each input tensor of M; iv) $\boldsymbol{t}$ is a list of *terms (functions or constants)* mapped to an input tensor of M via an externally defined mapping $\mathbf{D}$, i.e. $\mathbf{D}(\boldsymbol{t}) = \boldsymbol{x}$. Each neural atom introduces ground atoms of the form

$c = v$, where $c \in \{m_1(\boldsymbol{t}), ..., m_e(\boldsymbol{t})\}$ and $v \in \{v_1, ..., v_n\}$. In the cases where an input tensor $\boldsymbol{x}$ is indexed by several tuples $\boldsymbol{t}$s, rules with neural atoms as heads are allowed, that is,

$$nn(m(e, \boldsymbol{T}), [v_1, ..., v_n]) \leftarrow B(\boldsymbol{T}). \tag{9}$$

Here, $B(\boldsymbol{T})$ indicates the conjunctions of body literals with a list $\boldsymbol{T}$ of variables to be assigned by $\boldsymbol{t}$s.

A *NeurASP program* Π is the union of the set of answer set programs Π^{asp} (specified in subsection 4.2), and the set of neural atoms Π^{nn}. Denote the set of all ground atoms derived from Π^{nn} with σ^{nn}, a NeurASP program requires that atoms in σ^{nn} only appear in body of any rules in Π^{asp}.

Semantics. For any NeurASP program Π, each neural atom of Π^{nn} is replaced with the set of choice rules

$$1\{m_i(\boldsymbol{t}) = v_1, ..., m_i(\boldsymbol{t}) = v_n\}1. \text{ for } i \in \{1, ..., e\}\ , \tag{10}$$

where both lower and upper bounds are 1, meaning only one of the bracketed atoms is picked. Rules having neural atoms as heads in Π^{nn} are expanded analogously. An ASP counterpart Π' of Π can be obtained by replacing all the neural atoms with the set of ground atoms σ^{nn} introduced by neural atoms. Consequently, *stable models* of a NeurASP program Π can be defined as the stable models of its ASP counterpart Π'.

To compute the probability of a stable model of Π, we consider first the probability of an atom $m_i(\boldsymbol{t}) = v_j$ in σ^{nn}, that is,

$$P_{\Pi}(m_i(\boldsymbol{t}) = v_j) = M(\mathbf{D}(\boldsymbol{t}))[i, j]. \tag{11}$$

Recall that $\mathbf{D}(\boldsymbol{t})$ is an externally defined mapping that links the term $\boldsymbol{t}$ to an input tensor $\boldsymbol{x}$, Equation 11 can therefore be seen as the probability of an input tensor $\mathbf{D}(\boldsymbol{t})$ is labelled to v_j for a random event i. Let $I|_{\sigma^{nn}}$ be the projection of an given interpretation I to σ^{nn}, then the probability of a stable model I of Π can be defined as following:

$$P_{\Pi}(I) = \begin{cases} \frac{\prod_{c=v \in I|_{\sigma^{nn}}} P_{\Pi}(c=v)}{Num(I|_{\sigma^{nn}, \Pi})}, & \text{if } I \text{ is a stable model of } \Pi; \\ 0, & \text{otherwise,} \end{cases} \tag{12}$$

where $\mathit{Num}(I|_{\sigma^{nn},\Pi})$ is the total number of derivable stable models of Π agree with $I|_{\sigma^{nn}}$ on σ^{nn}. Equation 12 defines probability of a stable model I of Π is the product of the probability of each atom in σ^{nn}.

For a set of integrity constraints (recall subsection 4.2) O that are used to specify observable facts, e.g. ground truth labels of data samples, of for an input tensor are referred to *observation*, the probability of an observation O is satisfied by Π is defined as

$$P_{\Pi}(O) = \sum_{I \models O} P_{\Pi}(I).$$

Finally, we consider the probability of a set of observations $\mathbf{O} = \{O_1, ..., O_o\}$, calculated as the product of the probability of each $O_i \in \mathbf{O}$, that is,

$$P_{\Pi}(\mathbf{O}) = \prod_{O_i \in \mathbf{O}} P_{\Pi}(O_i).$$

Learning in NeurASP. The objective of learning in a NeurASP program Π is to find the set of parameters $\boldsymbol{\theta}$ in the associated neural network, such that derived probabilistic facts given by $\boldsymbol{\theta}$ maximise the log-likelihood of being satisfiable to the set of observations $\mathbf{O}$ under Π, i.e.

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} \sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O)),$$

where $\hat{\boldsymbol{\theta}}$ denotes the best set of parameters, $P_{\Pi(\boldsymbol{\theta})}(O)$ refers to the probability of the provided interpretation from the set of parameters $\boldsymbol{\theta}$ is satisfiable observation $O \in \mathbf{O}$.

As the probabilities of atoms in σ^{nn} are the output of a neural network, one can indeed compute the gradient of the NeurASP program Π with regard to $\boldsymbol{\theta}$ via back-propagation. Denoting the output probability distribution of neural network M with $\boldsymbol{p}$, then the gradient of $\sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O))$ with regard to $\boldsymbol{\theta}$ can be computed as follows:

$$\frac{\partial \sum_{O \in \mathbf{O}} log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial \boldsymbol{\theta}} = \sum_{O \in \mathbf{O}} \frac{\partial log(P_{\Pi(\boldsymbol{\theta})}(O))}{\partial \boldsymbol{p}} \times \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{\theta}}.$$

Inference in NeurASP. Inference in NeurASP can be considered as a process of computing the most possible stable models based on the ground probabilistic facts derived from neural atoms with domain terms corresponding to labelling classes, and the probabilities corresponding to the distribution of an input tensor is given a each of the classes of a neural network.

5 A NeurASP Approach to RPM

Intuitively, the most natural solution to RPM problems using NeurASP (Yang et al., 2020) would be training a neural network to learn low-level perception for given RPM problems. That is, recognising all the figural elements in each cell of a problem matrix (see blue box in Figure 2), and assigning each of them categorical labels that describe their attributes. For example, the *shape* attribute of horizontal figural element in the first cell of Figure 1 can be labelled to *curve*. Subsequently, we gather all the labels and the corresponding probabilities output by the NNs as the input domain for deriving ground atoms to be used in NeurASP inference. In this case, NNs are responsible for only low-level visual perception, LP programs derive the most possible answer of a problem instance, making full use of both DL models and symbolic reasoners. However, after carefully examining the implementation details of NeurASP, we observed that the NeurASP framework might not be scalable to solve RPM problems due to either the nature of the problem or the scalability of the framework itself.

5.1 Observation

To begin with, we argue that *assumptions on perceptual output of the NeurASP framework do not align with the non-verbal nature of visual abstract reasoning problem*. In fact, the non-verbal nature of RPM indicates that only visual sensory is mandantory during tests (Carpenter et al., 1990; Meo et al., 2007). Specifically, although test-takers need to figure out the underlying relations at different levels (figural elements, cells and rows) when solving an RPM, it is not necessary that one needs to perceive the exact attributive values of figural elements. Rather, relations are inferred through visual comparison only (Lovett et al., 2007; Lovett and Forbus, 2017). Therefore, *exact labels should be considered unknown when solving RPMs*. However, assumptions of NeurASP on the output of DL models imply that symbolic labels to be *explicitly given* and *exhaustively enumerable*. Take example the hand-written digits addition reasoning task in (Manhaeve et al., 2021a; Yang et al., 2020), where a neural network is responsible for recognising hand-written digits and additions are inferred by logic programs. In the case, output of the neural network is explicit and enumerable, considering numbers are verbally explicit and ranging from a small domain of $\{0, ..., 9\}$. In contrast, *attributes of figural elements are much more complicated to be explicitly captured by labels, e.g. how do we label an object in terms of position?* Indeed, one might argue such problem is trivial if unsupervised techniques, such as multi-attribute clustering (Sim et al., 2009), are used to obtain distinguishable clusters as symbolic labels to be associated to attributes. Nevertheless, the problem of innumerable domain will rise when trying to list all the possible output of neural networks as implied by

the domain assumption in (Yang et al., 2020). Clearly, foreseeing all combinations of irregularly varied attributive values like those occurring in RPMs is an infeasible task (Agarwal et al., 2021).

5.2 A Row-level Representation?

To address the problem above, we seek for an alternative solution where output representations of neural network are finite in size and enumerable. Inspired by Hu et al. (2021), where the proposed DL model extracts embedding vectors of underlying rules leveraging a hierarchical representation of an RPM, we consider an intermediate symbolic representation of underlying rules at the row level. Since each underlying rule applies to only one of the attributes of objects in each row (Figure 2), the number of rule-attribute pairs in RPM problems are finite. In light of this, instead of using neural network to identify attributes as exact representations of objects in RPM problems (object-level perception), capturing descriptive information of attributes with embedding vectors at the row-level seems more sensible. Therefore, we assume that the types of rules underlined by Carpenter et al. (1990) in RPMs to be known by the system. While it is indeed an strong assumption to be made, acknowledging the system with rules will make possible the enumeration of predicted outcomes of the neural network. To this end, we consider encoding RPMs at a row-level using an intermediate embedding representation that aggregates the descriptive information of figural elements in cells across a row. Consequently, for each input row embedding vector from an RPM, we can formulate the neural network as a multi-task classification problem w.r.t. the underlying rules of the row. Since the range of underlying rules is an actually enumerable and scalable domain, neural atoms could now capture the variations in a row.

5.3 The Challenge

Although an intermediate representation of rows helps neural network overcome the challenge posed by inenumerable predicted outcomes, another two potential problems are still in the way. First, as using row-level representation captures only variations at the cell-level and row-level, information at the object-level could be missing. It has been identified that information w.r.t. both *spatial relation and shape comparison* (Lovett et al., 2007) are crucial to discover the correspondences between objects. Second, taking per output of a neural network as a random event implies that probabilities assigned to each ground atom derived from a neural atom is *mutually exclusive*. However, due to the fact that different groups of objects could contribute to different variations on the same attribute across a row, the classification of a row-level representation to type of rules should be *mutually non-exclusive*. For example, colour of some objects might follow a progression rule and get darker (unary relation of increment), whereas others might follow

a distribute of 3 particular colours (ternary relation). In fact, the problem of computing the probabilities facts in mutually non-exclusive settings has been known as the *disjoint-sum-problem* (Raedt and Kimmig, 2015) in the field of logic programming. Thus, to address issues above a *more object-centric neural representation* and a *finer-grained symbolic representation* are needed, details will be discussed in the following sections.

6 Methodology

6.1 A Finer-grained Representation at Row-level

As mentioned, one potential pitfall of using NeurASP to establish a mapping between row and rules is its assumption on a categorical output of neural network, which restricts the system's ability to capture mutually non-exclusive random events. However, as suggested by (Raedt and Kimmig, 2015), one possible treatment to this problem is to split the formulation into mutually exclusive ones. Thus, we shall consider representations with a finer granularity, such that predicted outcomes of the neural network reflect to mutually exclusive events. That is, to find a representation that gives one-to-one correspondences to rules at a row-level.

Actually, objects that share the same variation patterns in each cell of a row can be grouped together by underlying rules (Zhang et al., 2019a). In line with the formulation of a rule of an RPM Formula 1, we define the notion of a *component* in RPMs.

Definition 3. *We say sets of objects O_1, O_2, O_3 distribute in columns $\{1, 2, 3\}$ respectively contribute to a **component**, if the following is satisfied*

$$\forall O_1, O_2, O_3, \alpha, i.(comp(i, \alpha, O_1, O_2, O_3) \leftrightarrow \forall o_1 \in O_1, o_2 \in O_2, o_3 \in O_3 \exists R. \\ (R(v(o_1, \alpha), v(o_2, \alpha), v(o_3, \alpha)) \wedge p(o_1, i, 1) \wedge p(o_2, i, 2) \wedge p(o_3, i, 3))). \tag{13}$$

In light of this, an one-to-one mapping from an attribute of a component to a rule can be established at a row-level, since a component contains all the objects that share the same rule on a particular attribute in a row. An illustration of the hierarchy from row to rule can be seen in Figure 5. While there might be figural elements irrelevant that serve as distractions and do not contribute to any rules (Santoro et al., 2018; Zhang et al., 2019a), our focus is only on components/objects since the goal is to establish the mapping between rules and a given row. Low-level variations of components, however, would be captured by neural representations as discussed.

Having the idea of representing low-level details by neural representations, we assume the maximal number of components allowed in RPMs is known by n, a component can be therefore identified by a simpler tuple of the form $(\boldsymbol{r}_i, c, \alpha)$ where $i \in \{1, 2, 3\}, c \in \{1, ..., n\}$ and $\boldsymbol{r}_i$ is the embedding vector represents the i-th row of an RPM. Denote $rule(\boldsymbol{r}_i, c, \alpha, R)$ the 4-ary predicate, meaning that a component c of the i-th row has an attribute α follows the rule R. Following the rule taxonomy in (Carpenter et al., 1990), we label the underlying rules in RPMs as $\mathcal{R} = \{\texttt{distr\_3}, \texttt{distr\_2}, \texttt{prog}, \texttt{const}, \texttt{add\_sub}\}$, namely, the *distribution of three/two values*, *quantitative progression*, *constant in a row* and *addition or subtraction* respectively. In fact, the relations introduced in Formula 2 can be instantiated by $\mathcal{R}$ (Wang and Su, 2015). We use a subscripted $\boldsymbol{r}_3^{(j)}$ when referring the third row constructed by choosing the j-th candidate as an answer. Then the set 4 can be represented by

$$Ans(i) = \{rule(\boldsymbol{r}_3^{(j)}, c, \alpha, \phi) | (j \in 1, ..., |ans|, \alpha \in \mathcal{A}, \phi \in C_R\}, \tag{14}$$

while keeping the objective stated in Section 3.3 the same. Given the new representation, we will specify in the following sections, how the row embedding $\boldsymbol{r}_i$ is learnt and how is NeurASP program encoded.

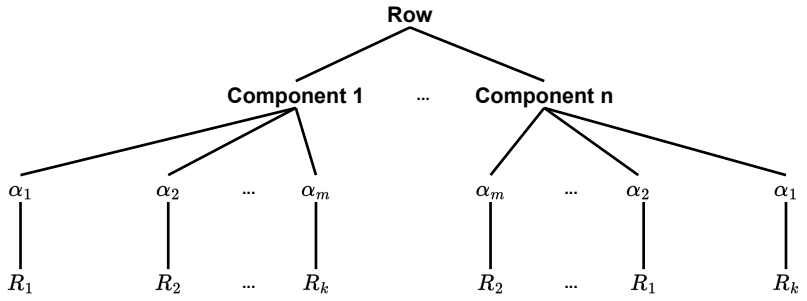


Figure. 5: Row-level Hierarchy

6.2 Overview of the Architecture

Generally, as shown in Figure 6, the proposed architecture consists of three stages which simulate the commonly adopted human-solving scheme mentioned in Section 3.2. As the first step, we obtain object-level latent representations by preprocessing RPMs with an AIR model, introducing an object-centric inductive bias (Spratley et al., 2020) to address the challenge w.r.t. lack of object-level information. Second, we aggregate the latent representations of objects and fuse them into an objectness-informed row-level embeddings. By utilising a multi-task DNN (MLTNN) module, we establish the mappings between the

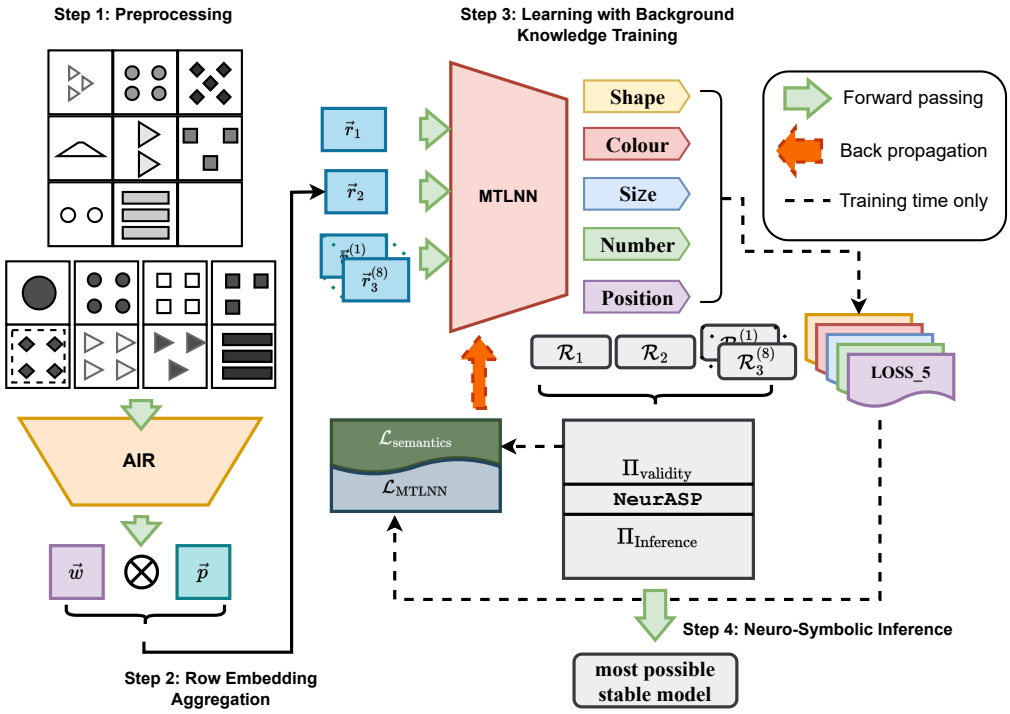


Figure. 6: General Pipeline

row-level embeddings and infer the underlying rules. Subsequently, we formulate a *validity program* for to be used by the NeurASP learning engine as semantic regularisation, effectively informing the MLTNN module with the background knowledge w.r.t. validity of RPMs. Consequently, a trained MLTNN is incorporated with a *inference program* in NeurASP to reason about the correct answers.

6.3 Preprocessing for Disentangled Representation

To identify relations between cells, object-centric observations to individual figural elements among cells are crucial (Carpenter et al., 1990; Lovett et al., 2007). Similar to Spratley et al. (2020), we exploit the fast scene decomposition ability of the AIR (Eslami et al., 2016) module to disentangle figural objects in each cell in an RPM as a preprocessing stage, then obtain latent vectors encoded information w.r.t. individual objects and their positions.

To achieve this, we train an AIR module takes each cell in both problem matrices and answer sets of RPMs as input. As seen in Figure 7, the module attends individual objects in a cell while inferring latent factors encoding the appearance, position and presence of the objects by an encoder network and recovering the latent factors to image tensors by a decoder network each by each during training. Specifically,

consider a set of RPMs $\mathcal{X}$, we extract the set all cells in $\mathcal{X}$ as the dataset X_{cell} for training. We denote each cell in X_{cell} with $\boldsymbol{x}_j$, where $j \in \{1, ..., |X| \times 16\}$. We use S to denote an AIR module with an encoder network E_ϕ and a decoder network D_θ, E_ϕ and D_θ are trained jointly to minimise the ELBO loss, that is,

$$\arg\min \sum_{j \in \{1,\ldots,|X_{\text{cell}}|\}} ELBO|\boldsymbol{x}_j - D_\theta(E_\phi(\boldsymbol{x}_j))|. \tag{15}$$

Assume that every $\boldsymbol{x}_j \in X_{\text{cell}}$ contains at most n objects. For each of object $o_i \in \{o_1, ..., o_n\}$, its occurrence in $\boldsymbol{x}_j$ is determined the generative model D_θ w.r.t. three types latent factors: i) $\boldsymbol{w}_j^i$ refers to the appearance of o_i; ii) $\boldsymbol{p}_j^i$ indicates the position and scale of o_i in $\boldsymbol{x}_j$; iii) $\boldsymbol{z}_j^i$ encodes the presence of o_i, that is, both $\boldsymbol{w}_j^i$ and $\boldsymbol{p}_j^i$ will be set to padding vectors in the same shapes if $\boldsymbol{z}_j^i = 0$. While $\boldsymbol{w}_j^i$ and $\boldsymbol{p}_j^i$ are assumed to subject to a Gaussian prior $N(\mathbf{0}, \mathbf{1})$ (continuous), presences of n objects are considered drawn from a geometric prior $n \sim \text{Geom}(\rho)$ since whether an object o_i presents in $\boldsymbol{x}_j$ or not is a binary event. As a result, once S is trained, we utilise the trained encoder E_ϕ to encode each cell of per RPM in X as a preprocessing stage. Namely, for each data example $X \in \mathcal{X}$, we have the following,

$$\begin{aligned} &Z_{\text{code}}^m = E_\phi(\boldsymbol{x}_m) = \{(\boldsymbol{w}_m^1, \boldsymbol{p}_m^1), ..., (\boldsymbol{w}_m^n, \boldsymbol{p}_m^n)\}, \\ &\mathcal{Z} = \{Z_{\text{code}}^m | m\} \text{ where } m \in \{1, ..., 16\}. \end{aligned} \tag{16}$$

In this way, we include both the *object-centric appearance* and *spatial information* encoded into disentangled latent vectors to be used by MLTNN at the next stage. Different from Spratley et al. (2020), which serialises the figural object tensors cropped by attention windows to build a new dataset, we directly use the encoded latent embeddings in the next stage as we believe that the disentangled representations learnt by AIR module condense representative information (van Steenkiste et al., 2019) of each cell and have a much lower dimensionality compared to raw image tensors, the subsequent NN training could thus be more efficient in for above reasons.

6.4 Row Embedding Aggregation and Multi-task Rule Classification

Since a row-level one-to-one correspondences can be established between component and rules, we model the mapping from a given row to underlying rules as n categorical classification problems with the rule taxonomy as labels. To achieve this, we first aggregated for each RPM the set of object-level embeddings $\mathcal{Z}$ obtained from the last stage to produce row embeddings. Specifically, for a given set of object-level

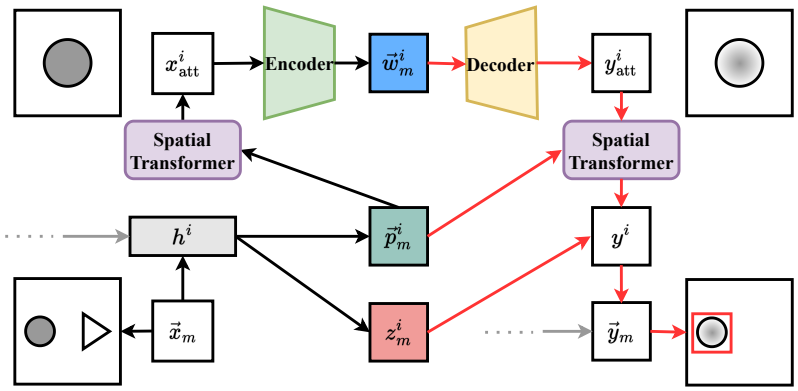


Figure. 7: Scene decomposition process of AIR. For each iteration, Spatial Transformer crops one object $\boldsymbol{x}^i_{\text{att}}$ from cell image $\boldsymbol{x}_m$ using the position and scale latent $\boldsymbol{p}^i_m$ inferred from the hidden state $\boldsymbol{h}^i$ from previous step, then $\boldsymbol{x}^i_{\text{att}}$ is encoded as object latent $\boldsymbol{w}^i_m$ by an Encoder network. Decoder network reconstructs the object image as $\boldsymbol{y}^i_{\text{att}}$, the same position and scale latent is used by the generative process to obtain shifted and scaled reconstruction $\boldsymbol{y}^i_m$, which will be added to scene image if the presence latent z^i_m was inferred to be true.

latents of an RPM, we first compute the i-th row embedding as following

$$\begin{aligned} &\textbf{Step 1: } \boldsymbol{Z}_{ij} = ELU\left(\begin{bmatrix} \boldsymbol{w}^1_{ij} \otimes \boldsymbol{p}^1_{ij} \\ \vdots \\ \boldsymbol{w}^n_{ij} \otimes \boldsymbol{p}^n_{ij} \end{bmatrix}\right), \boldsymbol{c}_{ij} = MLP(\boldsymbol{Z}_{ij}), \\ &\textbf{Step 2: } \boldsymbol{r}_i = MLP\left(\begin{bmatrix} \boldsymbol{c}_{i1} \\ \boldsymbol{c}_{i2} \\ \boldsymbol{c}_{i3} \end{bmatrix}\right). \end{aligned} \tag{17}$$

As visualised in Figure 8, for the first step, we unify the latent vectors for appearances $\boldsymbol{w}^o_{ij}$ and position $\boldsymbol{p}^o_{ij}$ with the element-wise bi-linear operation $\otimes$, then pass the unified linear combinations to an ELU [1] function to obtain the unified latents $\boldsymbol{Z}_{ij}$ of a cell. Aggregating all the objects latents via the fully connected layer MLP with $ReLU$[2] activation, the cell-level embedding $\boldsymbol{c}_{ij}$ can be obtained. For the second step, we stack all 3 cell-level embeddings of the i-th row and again use a MLP function to aggregate the information at cell level as the row embedding $\boldsymbol{r}_i$. Note that for every third row $\boldsymbol{r}_3$, the row embedding is

[1] Exponential Linear Unit is an activation function that tends to converge cost to 0 faster and produce more accurate results.

[2] Rectified Linear Units is the activation function formulated as $max\{0, x\}$.

constructed as $\begin{bmatrix} \boldsymbol{c}_{31} \\ \boldsymbol{c}_{32} \\ \boldsymbol{c}_{33}^{(q)} \end{bmatrix}$, where $\boldsymbol{c}_{33}^{(q)}$ is the cell embedding of the q chosen candidate from the answer set. Consequently, for each RPM, we obtain the set of row embeddings $H = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3^{(1)}, ..., \boldsymbol{r}_3^{(8)}\}$.

To build the mapping between components are rules, we consider classifying each attributive variation of the components of a row by the rule taxonomy. That is, we build a MTLNN M_{rule} such that for each input row embedding $\boldsymbol{r}_i$, we classify each of components $c_i^{(k)}$ w.r.t. attribute α into the category $\mathcal{R} = \{\texttt{distr\_3}, \texttt{distr\_2}, \texttt{prog}, \texttt{const}, \texttt{add\_sub}\}$. As different attributes of objects vary independently and all the classification task are equally important, the learning task can be viewed as a MTLNN using shared trunk (unified input representation to all tasks) with task-specific branches (Zhang et al., 2014). Also, each of the sub-tasks is a multi-class classification problem, it is reasonable to employ cross-entropy function as the loss functions. Thus, we formulate the learning objective as following

$$\underset{\{\boldsymbol{W}^{(t)}\}_{t=1}^{T}}{\arg\max} \sum_{i=1}^{N} \sum_{t=1}^{T} y_i^{(t)} \log(p(y_i^{(t)} | \boldsymbol{r}_i; \boldsymbol{W}^{(t)})) - \sum_{t=1}^{T} ||\boldsymbol{W}^{(t)}||^2, \tag{18}$$

where T denotes the total number of tasks, N is the input sample size. Since there are at most n components varying over 5 attributes, T is assigned by $5n$. $\boldsymbol{W}^{(t)}$ is the learnable parameters of the t-th task. p is the softmax function $p(y_i^{(t)} = r | \boldsymbol{r}_i) = \frac{\exp\{(\boldsymbol{W}_r^{(t)})^\intercal \boldsymbol{r}_i\}}{\sum_j^{|\mathcal{R}|} \exp\{(\boldsymbol{W}_j^{(t)})^\intercal \boldsymbol{r}_i\}}$, produces the probability distribution over rule classes $\mathcal{R}$ per attribute α of a component $c_i^{(k)}$. To penalise large weights, the second term of Equation 18 is added as regularisation.

As a result, for each input row representation $\boldsymbol{r}_i \in H$, M_{rule} output a matrix $P^{(i)}$ in $\mathbb{R}^{n \times |\mathcal{A}| \times |\mathcal{R}|}$. Each row of P represents the distribution over $\mathcal{R}$ produced by one of the $5n$ sub-task classifiers, i.e. the probability distribution of how a particular component $c_i^{(k)}$ of the row $\boldsymbol{r}_i$ is classified over the rule taxonomy $\mathcal{R}$. To gather the probabilities for all of H per RPM, we sequentially feed the 10 row embeddings to M_{rule}. We will show in the next step, how to integrate the probability matrix $P^{(i)}$ to NeurASP using neural atoms and facilitate the MTLNN training with background knowledge encoded as answer set programs.

6.5 Learning with Background Knowledge

One of the promising features of NeurASP is to regularise the NN integrated with rich background knowledge encoded as logic programs, such that the NN also learns from logically encoded semantic constraints (Yang et al., 2020). To inform M_{rule} the knowledge of whether the predicted rules contribute to a valid RPM (Wang and Su, 2015), we define the NeurASP program $\Pi_{\text{valid}}^{\text{nn}}$ (Listing 19 - 24), underlining the validity of predictions made by M_{rule}.

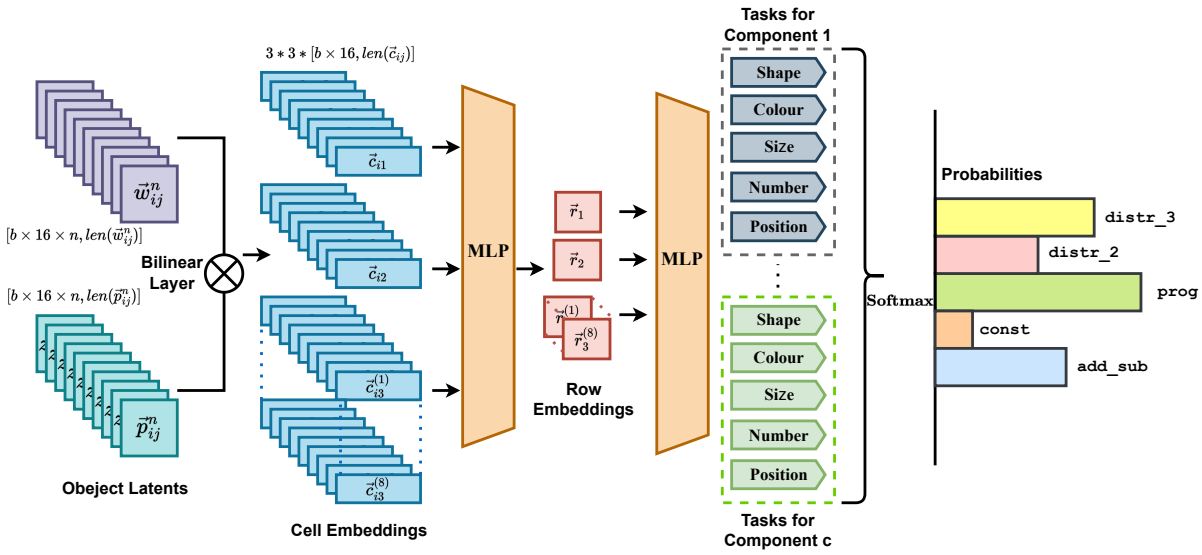


Figure. 8: Multi-task Rule Classification Module

To express M_{rule} and introduce the atomic facts gathered from its predicted outcomes, we first define the neural input Listing 20 and the rule 19 with a neural atom as head and the occurrence of component atoms as body. While the predicate $comp/2$ in the body of rule 19 represents the input of M_{rule} is a component C of the row embedding X, $nn(rule(5, X, C), [\texttt{distr\_3}, \texttt{distr\_2}, \texttt{prog}, \texttt{const}, \texttt{add\_sub}])$ defines the output of M_{rule} for the input component w.r.t. 5 attributes over the rule taxonomy. Note that the each row embedding is identified by a tuple $t(X, I)$, where X is grounded by a row embedding and I is the subscript indicates the index of a candidate in answer sets when $X = \boldsymbol{r}_3$. Altogether, for each ground atom in Listing 20, the rule 19 generate all the possible worlds by expanding the neural atom to the choice rules of form 10. As such, the ASP counterpart of $\Pi^{\text{nn}}_{\text{rule}}$ can be obtained, we denote the ASP program by Π_{rule}.

$$nn(rule(5, X, C), [\texttt{distr\_3}, \texttt{distr\_2}, \texttt{prog}, \texttt{const}, \texttt{add\_sub}]) \leftarrow comp(X, C). \tag{19}$$

$$\begin{aligned} &comp(t(\boldsymbol{r}_1, 0), 1, ..., n). \\ &comp(t(\boldsymbol{r}_2, 0), 1, ..., n). \\ &comp(t(\boldsymbol{r}_3, 1), 1, ..., n). \\ &\vdots \\ &comp(t(\boldsymbol{r}_3, 8), 1, ..., n). \end{aligned} \tag{20}$$

Then in Listing 21, we represent an RPM constraint by $rpm\_cr/3$ as the intersections of all the applicable rules to each of the third rows and the dominant rules $dominant/3$ occurring in the first two rows.

$$\begin{aligned}
&dominant(E, C, R) \leftarrow rule(E, C, t(\boldsymbol{r}_1, 0), R), rule(E, C, t(\boldsymbol{r}_2, 0), R).\\
&intersect(E, t(\boldsymbol{r}_3, I), C, R) \leftarrow rule(E, C, t(\boldsymbol{r}_3, I), R), dominant(E, C, R).\\
&rpm_cr(E, C, R) \leftarrow intersect(E, t(\boldsymbol{r}_3, I), C, R).
\end{aligned} \tag{21}$$

Having derived the set of RPM constraints, we check again how each of the third rows aligns with the RPM constraints. The first rule in Listing 22 derives an atom of $cr\_intersect/4$ for every underlying rule of a third row that coincides with one of the RPM constraints. Then we record how many RPM constraints satisfied by the third row of choosing the I-th candidate from the answer set. Therefore, the second rule count the number of constraints satisfied and derive an atom of $answer/2$, indicating the I-th candidate contribute to a third row with N underlying rules coincide with RPM constraints. The actual answer can thus be derived as an atom of $max\_ans/1$ following the ***Monotonicity of RPM Constraints*** (Wang and Su, 2015).

$$\begin{aligned}
&cr_intesect(E, I, C, R) \leftarrow rule(E, t(\boldsymbol{r}_3, I), C, R), rpm_cr(E, C, R).\\
&answer(I, N) \leftarrow rule(E, t(\boldsymbol{r}_3, I), C, R),\\
&\qquad\qquad N = \#\mathtt{sum}[E, C_1, R : rule(E, t(\boldsymbol{r}_3, I), C_1, R), rpm_cr(E, C_1, R)].\\
&max_ans(I) \leftarrow answer(I, N), N = \#\mathtt{max}[N_1 : answer(I_1, N_1)].
\end{aligned} \tag{22}$$

To check the well-formedness of RPMs (Wang and Su, 2015), we specify the notion of equivalent answers in Listing 23, then eliminate the stable models which violate the integrity constraints w.r.t. well-formedness in Listing 24. Specifically, rules in Listing 23 describe that an atom of eqv_ans is derived if two answers satisfy the same amount of RPM constraints, and such relation of equivalence is closed under symmetric, transitive and reflective closures. While the first constraint in Listing 24 corresponds to the prerequisite defined in (Wang and Su, 2015), it ensures that an RPM must have at least one satisfiable RPM constraint. The following constraints reject any stable models derived more than one actual answer (correctness) and refuse any other answers but the actual one have less than one equivalent answer

(necessity).

$$
\begin{aligned}
&eqv\_ans(X,Y) \leftarrow answer(X,N), answer(Y,N),\\
&\qquad\qquad \#\mathtt{sum}[E,C,R : cr\_intersect(E,X,C,R), cr\_intersect(E,Y,C,R)] = N_1, N_1 = N.\\
&eqv\_ans(X,Y) \leftarrow eqv\_ans(Y,X).\\
&eqv\_ans(X,Z) \leftarrow eqv\_ans(X,Y), eqv\_ans(Y,Z).\\
&eqv\_ans(X,X) \leftarrow answer(X,N).
\end{aligned} \tag{23}
$$

$$
\begin{aligned}
&\bot \leftarrow \#\mathtt{sum}\{E,C,R : rpm\_cr(E,C,R)\} \leq 1.\\
&\bot \leftarrow max\_ans(I_1), max\_ans(I_2), I_1 \neq I_2.\\
&\bot \leftarrow \sim max\_ans(I), answer(I,N), \#\mathtt{sum}[Y : eqv\_ans(I,Y), Y \neq I] < 1.
\end{aligned} \tag{24}
$$

In addition, to inform the model about correct predicted rules and the right answers, we append to $\Pi^{\text{nn}}_{\text{rule}}$ the ground truth label of each row and the actual answer as set of observation constraints $\boldsymbol{O}$ for each RPM example during training, that is,

$$
\begin{aligned}
&\bot \leftarrow rule(1, t(\boldsymbol{r}_i, j), 1, r^{(1)}_{ij}).\\
&\vdots\\
&\bot \leftarrow rule(e, t(\boldsymbol{r}_i, j), n, r^{(n\times e)}_{ij}).\\
&\bot \leftarrow max\_ans(a).
\end{aligned} \tag{25}
$$

Here, e is the total number of attributes, and $r^{(n\times e)}_{ij} \in H$ is the rule type of the n-th component on the e-th attribute of row $t(\boldsymbol{r}_i, j)$ ($j \in \{1, ..., 8\}$ if $i = 3$), a is the index of the actual answer of the RPM.

To compute the probabilities of the stable models, we denote $\sigma^{\text{nn}}_{\text{rule}}$ the set of ground atoms of the form $rule(e, t(\boldsymbol{r}_i, j), c) = r^{(k)} \in \sigma^{\text{nn}}_{\text{rule}}$ introduced by Listing 19 and 20 in all the possible worlds, where $i \in \{1, 2, 3\}, e \in \{1, ..., |\mathcal{A}|\}, r \in H, k \in \{1, ..., |H|\}, c \in \{1, ..., n\}$ and $j \in 1, ..., 8$ if $i = 3$ otherwise 0. Recall that the probabilities output by M_{rule} for each input $\boldsymbol{r}_i \in H$ is a matrix $P^{(i)}$ in $\mathbb{R}^{n\times|\mathcal{A}|\times|\mathcal{R}|}$, then the probability of assigned to each ground atom $rule(e, t(\boldsymbol{r}_i, j), c) = r^{(k)}$ is

$$
\mathrm{p}\Big(rule(e, t(\boldsymbol{r}_i, j), c) = r^{(k)}\Big) = M_{\text{rule}}\Big(\boldsymbol{D}(\boldsymbol{r}_{t(i,j)})\Big)[c, e, k] = P^{(t(i,j))}[c, e, k].
$$

In light of this, we compute the probability of a stable model under Π_{rule}. Given an interpretation I, we denote the projection of I onto $\sigma^{nn}_{\text{rule}}$ as $I|_{\sigma^{nn}_{\text{rule}}}$. The number of stable models of Π_{rule} for I is denoted by

Num$(I|_{\mathcal{R}^{nn}}, \Pi_{\text{rule}})$. For simplicity, we refer each $rule(e, t(\boldsymbol{r}_i, j), c) = r^{(k)}$ as $(e = r) \in \sigma^{nn}_{\text{rule}}$, then the probability of a stable model of Π_{rule} can be computed as

$$P_{\Pi_{\text{rule}}}(I) = \begin{cases} \frac{\prod_{e=r \in I|_{\sigma^{nn}_{\text{rule}}}} \text{p}(e=r)}{\textit{Num}(I|_{\sigma^{nn}_{\text{rule}}, \Pi_{\text{rule}}})}, & \text{if } I \text{ is a stable model of } \Pi_{\text{rule}}; \\ 0, & \text{otherwise.} \end{cases}$$

The probability of all satisfiable stable models of all integrity constraints $O \in \boldsymbol{O}$ is thus computed as

$$P_{\Pi_{\text{rule}}}(O) = \sum_{I \vDash O} P_{\Pi_{\text{rule}}}(I),$$
$$P_{\Pi}(\mathbf{O}) = \prod_{O_i \in \mathbf{O}} P_{\Pi}(O_i).$$

Corresponding to Section 4.3, learning objective of our reasoning module is defined as finding the parameter $\hat{\boldsymbol{\theta}}$ of neural network M_{rule}, such that the probability distribution produced by M_{rule} gives the maximal log-likelihood of observation O in Equation 24 under Π_{rule}, i.e.,

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} log P_{\Pi_{\text{rule}}(\boldsymbol{\theta})}(\boldsymbol{O}). \tag{26}$$

To further incorporate M_{rule} and Π_{rule}, we unify the learning objectives (Equations 18 and 26) as one by treating the learning objective of reasoning module as semantic constraint (Xu et al., 2018; Yang et al., 2020) to M_{rule} during training (see Figure 6), formulated as follow

$$\mathcal{L}_{\text{rpm}} = (1 - \alpha)\mathcal{L}_{\text{semantics}} + \alpha \sum^{10 \times N \times n} \mathcal{L}_{\text{MTLNN}},$$

where N now denotes the sample size of dataset. By taking both classification and semantic loss into account M_{rule} makes use of the background knowledge in Π_{rule} as regularisation, such that Π_{rule} can help identify predictions that violate the semantic constraints and therefore make the row embedding classification in M_{rule} more robust.

6.6 Neuro-symbolic Inference

At the inference stage, our goal is to incorporate the trained M_{rule} with the NeurASP program and infer the correct answers. To achieve this, we define an inference program Π_{infer} by simply removing all the observation constraints $\boldsymbol{O}$, i.e. $\Pi_{\text{infer}} = \Pi_{\text{validity}} \backslash \boldsymbol{O}$. Similar to Section 6.5, for each input RPM, we use

the trained M_{rule} to obtain the set of probability matrices $\{P^{(1)}, ..., P^{(i)}\}$ to be mapped to Π_{infer} and compute the probabilities of stable models. In this regard, the most probable stable model is derived as the inferring answer. Note that as high-level reasoning is done by answer set solver, there is *no statistical bias involved* for inferring the answer and the *results are explainable* due to the declarative nature of logic.

7 Experiments

With the proposed framework, we are interested in examining following questions:

1. Can the AIR module effectively disentangle figural elements in RPMs to introduce the right object-centric inductive bias as expected?
2. Would the proposed Neuro-symbolic architecture work or not? How effective it is? How does it perform on different settings of RPMs?
3. Can the background knowledge provided facilitate the training of MTLNN? How do the models trained with/without semantic loss perform in inference time?

To seek the answers to questions above, we train and evaluate the AIR module and the proposed architecture on I-RAVEN dataset (Hu et al., 2021). In particular, we first experiment the disentangling capability of the AIR module. Taking the best performed AIR model to preprocess the I-RAVEN dataset, we train the MTLNN together with the validity program and evaluate the MTLNN's performance in rule inferring. Lastly, we evaluate the best performed MTLNN integrated with the inference program on the RPM answer inference task.

7.1 Dataset

While there are two major types of RPM-like datasets available for experiment, we choose RAVEN/I-RAVEN (Zhang et al., 2019a; Hu et al., 2021) dataset over PGM dataset (Santoro et al., 2018) since RAVEN/I-RAVEN datasets are more challenging in reasoning (Zhang et al., 2019a), thus could allow a better evaluation of abstract reasoning ability.

The RAVEN dataset is a set of RPM matrix instances which automatically generated via the A-SIG structure. It consists of $1,120,000$ images and $70,000$ RPM problems. In particular, the dataset provides total 7 types of configurations of RPM problems, including Center-Single (Center), Left-Right (L-R), Up-Down (U-D), Out-InCenter (O-IC), Out-InGrid (O-ID), 2×2Grid, and 3×3Grid. Each of the configurations contains $10,000$ problems. Figural elements in each of the problems have 5 attributes: number

of objects, position, shape, size, and colour. Attributes of the objects could vary according to 4 types of rules: Constant, Progression, Arithmetic and Distribute Three. The implementations of rules include all the 5 relations in Advanced RPM identified by Carpenter et al. (1990). The number of variations in the combinations rule-attribute combinations is 19, excluding an inapplicable variation of Arithmetic rule on Type attribute. A sample taken from RAVEN dataset is shown as Figure 9. In terms of format, raw

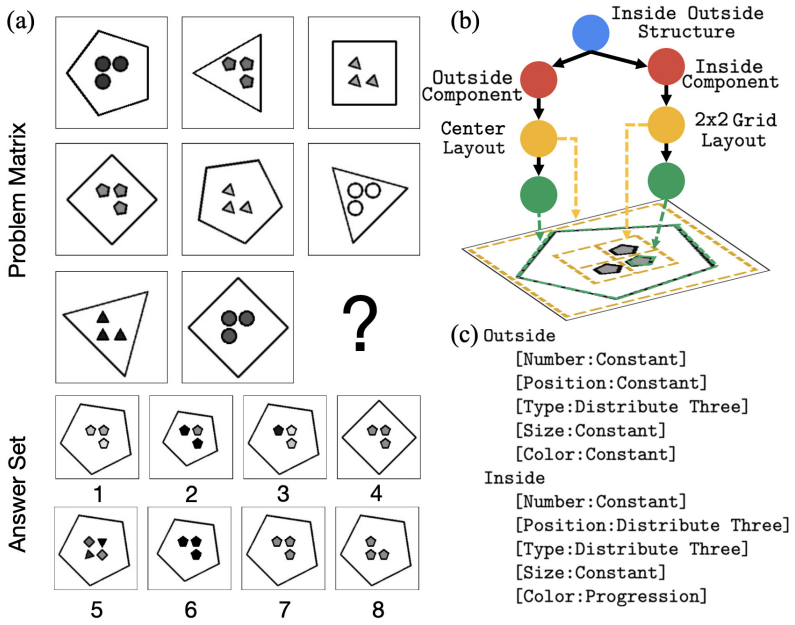


Figure. 9: (a) A sample taken from RAVEN dataset (Zhang et al., 2019a). (b) In this sample problem, the general structure configuration is Out-InGrid (O-ID), the component outside follows the Center-Single (Center) layout, the component inside follows the 2×2Grid layout. (c) The attribute-rule combinations of inside and outside components are listed.

information of each sample is serialised in a `.npz` file, including: (i) 8 cells of a problem matrix and 8 candidates of an answer set, together encapsulated as a 16-channel image tensor in the size of 160×160; (ii) the index of the correct candidate in the answer set; (iii) the structure tree annotation for the sample problem; (iv) meta description of the sample. In addition, detailed A-SIG structure description for each `.npz` file is recorded in an `.xml` file together with the underlying rules to attributes of each component in cells.

We use the fairer version of RAVEN dataset, i.e. I-RAVEN (Hu et al., 2021) in the experiment to mitigate potential statistical backdoors. Following (Zhang et al., 2019a), we divide the dataset into three splits, 6 folds for training, 2 folds for validation, and 2 folds for testing.

7.2 The AIR Module

Setup. We train a self-supervised AIR model with cell images divided from each example of the dataset prior to the rule classification training pipeline, and utilise the trained model to obtain disentangled representations of each RPM instance. We use a batch size of 32 (for cell images) to enable batch parallelisation. To further prevent the model from exploiting any statistical bias and mitigate overfitting, when loading training samples, answer set are shuffled randomly. Following the specifications from (Spratley et al., 2020) and the original AIR paper (Eslami et al., 2016), we implement the AIR module using the Pyro library (Bingham et al., 2019). The maximal amount of figural elements in a cell is set to 9. The Gaussian prior is set to $(\mu = 2.0, \sigma = 0.4)$, while geometric prior is assigned to 0.01 with a linear anneal after $4,000,000$ steps. The module are trained for 60 epochs with early stoppings or maximum number of epochs reached. The ADAM optimiser (Kingma and Ba, 2015) is used for optimisation and with the set of learning rate $\{1e^{-4}, 5e^{-5}, 1e^{-5}\}$.

Results of AIR Module. From Figure 10 we can see that the assignments of learning rates have huge impacts on loss optimisation of the AIR module. While the learning rates of $1e^{-4}$ and $1e^{-5}$ seems either too large or too small, the ELBO loss raises steeply and converge early respectively. The minimal ELBO loss during 60 epochs is achieved by the model with a learning rate of $5e^{-5}$. When observing the qualitative results (in Figure 11) of the model with a learning rate of $5e^{-5}$, we found that the model seem to have the capabilities in distinguishing figural elements from background but having difficulties in inferring the presences of figural elements , and also struggling at singling out figural elements in the 9-object setting.

As claimed in (Spratley et al., 2020), the AIR model does suffer from poor performance in small object settings. Whereas for the first problem, we observed part of the source code that is available, and found that models to process different settings w.r.t. numbers of figural elements were trained separately with different hyperparameter settings. However, such details are not specified in (Spratley et al., 2020). Although we believe that separately training AIR models for different number settings could be inappropriate as this explicitly gives away the number of figural elements in a cell, trained models could be overfitting to the setting thus lost their generality. Nevertheless, we will stick to the plan and use the trained model as a tool for prerprocessing since the model does highlight important elements of a cell thus may provide useful inductive bias to the model.

7.3 The Main Pipeline

Pre-processing via AIR. In this step, a pre-trained AIR module picked from the best performing models from the previous step is used to preprocess each of the data. As specified in subsection 6.3, we use only

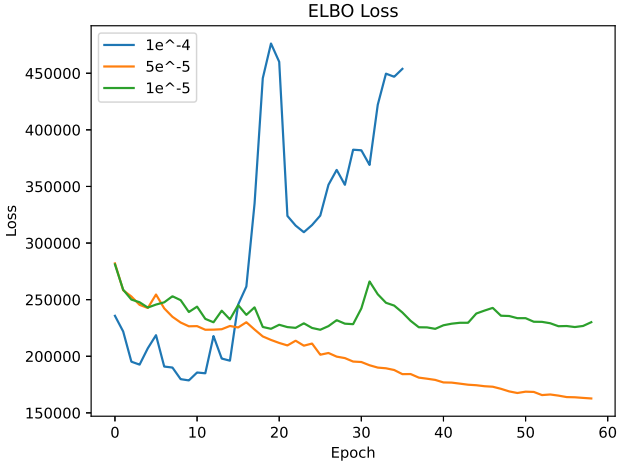
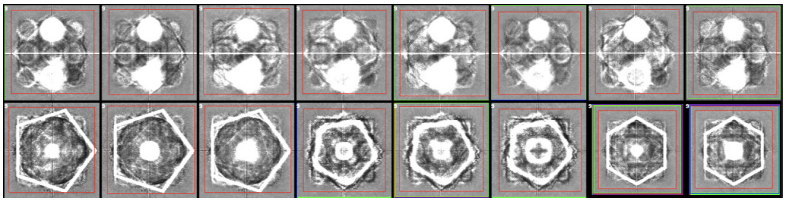
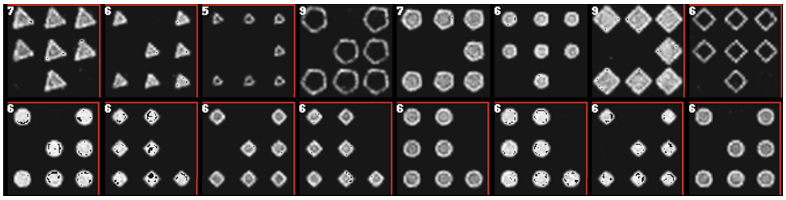


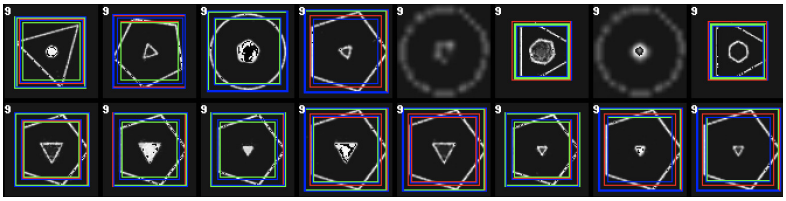
Figure. 10: ELBO loss evaluation of the AIR module on test set



(a) Cells reconstructed on the 10th epoch



(b) Cells reconstructed on the 30th epoch



(c) Cells reconstructed on the 60th epoch

Figure. 11: Qualitative results of AIR module

the encoder to produce pairs of latent vectors w.r.t appearances $\boldsymbol{w}$ and locations $\boldsymbol{p}$ of objects in each example. The model achieved $161,743$ on ELBO loss is used in this case. For each pair of latent vectors, the dimensionalities of $\boldsymbol{w}$ and $\boldsymbol{p}$ are set to 80 and 3 respectively (following (Spratley et al., 2020)). As a result, for each input example, we encode a pair of tensors $\boldsymbol{w} \in \mathbb{R}^{b\times16\times9\times80}$ and $\boldsymbol{p} \in \mathbb{R}^{b\times16\times9\times3}$, where b is the batch size (will be discussed later).

Validity Program Guided MTLNN Rule Prediction. For this stage, we are aiming at building the correspondences between rows and rules with the background knowledge provided by the validity program. While sizes of cell embeddings and row embeddings are set to $1,200$ and 400 respectively, we set the maximal amounts of components to 2 as it is in the I-RAVEN dataset. Note that current implementation of NeurASP encapsulates data loading process for the needs of creating instance-specific observations, thus requires to load the whole dataset at once to the memory which could *easily lead to memory overflow* when the dataset is large like the I-RAVEN. We hence modify part of the source code to implement ourselves a `learn_step` function, where data instances and instance-specific observations are processed one by one, avoiding the heavy memory needs in training time. For optimisation, we use a regular ADAM (Kingma and Ba, 2015) setting for classifiers with a learning rate of $1e^{-4}$ and L2 regularisation. Importantly, to examine the effect of background knowledge, we train 3 instances of the MTLNN with different values of α in Equation 6.5. While $\alpha = 0$ and $\alpha = 1$ indicate the models trained with full and none respectively the semantic loss, the model trained under a $\alpha = 0.5$ setup enjoys both regularisation from the semantic loss and classification loss. Furthermore, since ASP solving is required when calculating the semantic loss, NeurASP cannot process examples in batches. For this reason, the batch size is set to 1. Not being able to full leverage parallel optimisation and the needs for solving answer set programs per example lead to a *significant time increase* for training (will be discussed later). Therefore, we train the models under the setups above only on 20% of the training split for 10 epochs.

Setting	Center	L-R	U-D	O-IC	O-ID	2×2Grid	3×3Grid	Row	Avg
$\alpha = 0$	77.05	29.37	29.22	38.00	51.42	**59.68**	**58.92**	49.11	0.19
$\alpha = 0.5$	**87.38**	31.11	36.21	**72.28**	52.49	57.97	50.90	55.49	**3.05**
$\alpha = 1$	86.65	**31.95**	**41.16**	72.10	**52.94**	57.76	48.87	**55.92**	2.68

Table 1: MTLNN classification results on the test split

Inference. For the final stage, we take the best performed MTLNN from the previous step, and perform logical reasoning to infer the correct answers for examples in the test split. Following the commonly adopted evaluation protocol, we evaluate our approach via accuracy on the correctness of inferred answer. Keeping setups of the MTLNN the same, we compare the inference results of our approach with the baseline DRT (Zhang et al., 2019a) and the state-of-the-art model SRAN (Hu et al., 2021) on I-RAVEN dataset.

Implementation. The MTL classification network and reasoning components are developed based on Pytorch library (Paszke et al., 2019) and NeurASP (Yang et al., 2020) respectively, all the logic programs are encoded in the format of ASP (Calimeri et al., 2020). The ASP encoding are grounded and solved using `clingo 5.4` API [3]. All the experiments were run on the GPU resources (Nvidia P100 and V100) provided by the *Advanced Research Computing at Cardiff (ARCCA)* [4] with job scripts available in the Git repository.

7.4 Results of the Main Pipeline

Results on Rule Prediction. Although the MTLNNs are trained with only 20% percent training data, we evaluate their rule prediction performances with *full* test split in the test time to examine the effectiveness of background knowledge and object-centric inductive bias since both factors could promote the data efficiency in training time. The results can be seen in Table 1. Overall, we observed the followings

1. Even though the models trained on less data, results on some of the problem settings still appear optimistic.
2. While the model trained with only semantic loss ($\alpha = 0$) obtained inferior performances in general, performances of the models appear similar when α is set to $\{0.5, 1\}$. Performances of all setups w.r.t. most of problem settings, except for `O-IC`, behaved similarly. Surprisingly, the model trained without semantic loss obtained the best results overall.
3. Models trained with semantic losses tend to predict better on single component problem settings (`Center`, `2×2` and `3×3`).

The first observation could reflect the representative power of disentangled representations produced by the AIR module, as models are able to generalise well on the problem settings (`Center` and `O-IC` particularly) even when less training data is used with and without semantic loss included. The second

[3] `https://potassco.org/clingo/python-api/5.4/`
[4] `https://www.cardiff.ac.uk/advanced-research-computing/about-us`

observation implies that regularisation on classification losses themselves are more crucial than semantic loss for rule prediction in our method. This could due to fact that predicted outcomes of all 10 row embeddings are used to calculate the semantic loss, semantic constraints that are distributed to individual row embeddings could be really sparse. The assumption seems coherent to the third observation, where the semantic loss is computed with a denser probability distributions since in the single-component problem settings since rule labels on the second component of each row are padded by 0 (meaning no rules are applied). While the model could be more certain about the labels for the second components, predictions made for the first components become a stronger indications for RPM validity checking. This can particularly be reflected by the results on the 2×2 and 3×3 settings, where including only semantic loss can lead to better performances.

Results on Inference. As can be seen in Table 2, our approach is clearly not a successful attempt: inference results of the method are still quite distant from the state-of-the-art approaches, and surprisingly, results obtained in the `L-R`, `U-D` and `O-IC` settings are even worse than random guess. While the results appear pessimistic, we attribute the poor performances to two potential reasons.

First, and perhaps the most direct reason is the poor performances of the pretrained MTLNN. Clearly, having correct inference results in NeurASP depends heavily on the output probability distributions of NNs to be input to logic programs (the inference program in our case). When output of the upstream NN is not as accurate, it is not surprising that the downstream logic reasoning will derive incorrect conclusions. More critically, taking output from NNs for multiple times as random event could potentially lead to a *cascading failure*. For example, even though a decent accuracy (87.38% the highest) is achieved by the model in the `Center` setting of RAVEN, making full correct predictions on rules for 10 row embeddings w.r.t. 5 attributes of 2 components results to a vanishingly small probability of success (a joint probability of 0.875^{50}). This could also be the reason why inference accuracies on the high prediction accuracy settings are far away from expectation. Second, the representation of rule prediction at a row level as MTLNN is in fact really restricted. As described in section 5 and section 6, many assumptions about the problem settings are made in order to adapt NeurASP to the RPM problem. It is clear that assuming the type of rules and maximal number of components are know is unrealistic. From the perspective of computation, a 20-head categorical classifier is indeed expensive to train. This is also shown by the fact that even model was trained without semantic loss, the training process still takes a significant long time for each epoch. For this reason, we ended up using only 20% of the training data, which could lead to poor generalisations on several of the problem settings.

Method	Center	L-R	U-D	O-IC	O-ID	2×2Grid	3×3Grid	Avg
Random Guess	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
DRT	54.6	15.3	10.8	39.7	51.4	27.6	19.1	22.1
SRAN	65.1	25.7	24.7	55.8	66.5	43.9	28.1	40.1
Ours	13.0	6.5	9.0	11.0	15.5	13.0	13.5	13.0

Table 2: Inference results on the test split

8 Future Work and A Potential Solution

8.1 Future Work

Although our method appear to be an unsuccessful attempt of adapting NeurASP to the RPM problem, there are still unexplored aspects of the approach to be further experimented. Experimentally, provided that training on only 20% percent of the data is one of the reasons leads to inferior performances of the MTLNNs, it would be interesting to see how well the models could generalised the rule prediction task when the whole training set is used. In line with this, different settings of hyperparameters are still worth trying if time permits. For example, experimenting on different setups of latent size, learning rate of the MTLNNs, another proportions of the semantic loss ($\alpha = \{0.2, 0.8\}$) and etc. Obviously, results of the inference stage could be improved if the upstream prediction accuracies raised after the models are experimentally fine-tuned.

From a methodological point of view, although latent representations preprocessed by the AIR module seem to introduce object-centric inductive bias to the models, the training procedure of the AIR module and the main pipeline are however, in isolation. Hence it would also be interesting to see how an end-to-end solution with typical methods of deep computer vision models, e.g. ResNet (He et al., 2016), can actually performance for rule predictions. Additionally, one could consider the end-to-end unsupervised training idea proposed in (Agarwal et al., 2021), where amazingly, training of the DL model can be integrated with the symbolic reasoner without any supervisory signals (labels). In this way, the AIR module could be trained jointly under the regularisation of the validity program and thus logically constrained disentangled representations could be learnt. In fact, recent works on disentangled representation learning seem to provide really neat ways to model attributive variations of objects via distinguishable latent factors (Jiang and Ahn, 2020; Sahu and Pavlovic, 2021). Together with vector discretisation techniques like (van den Oord et al., 2017), one might be able to capture attributive variations at the object-level and ground the logic program with discretised disentangled representations. Moreover, another logic program-based NPLP framework is DeepProblog (Manhaeve et al., 2021a). Interestingly, DeepProblog enables logic program with the ability to operate on neural embeddings directly, offering a more flexible

way to extend the semantics of LP predicates with neural representations. In light of this, how to leverage background knowledge to enhance the performances of embedding-based models like (Hu et al., 2021) would definitely be an interesting research question.

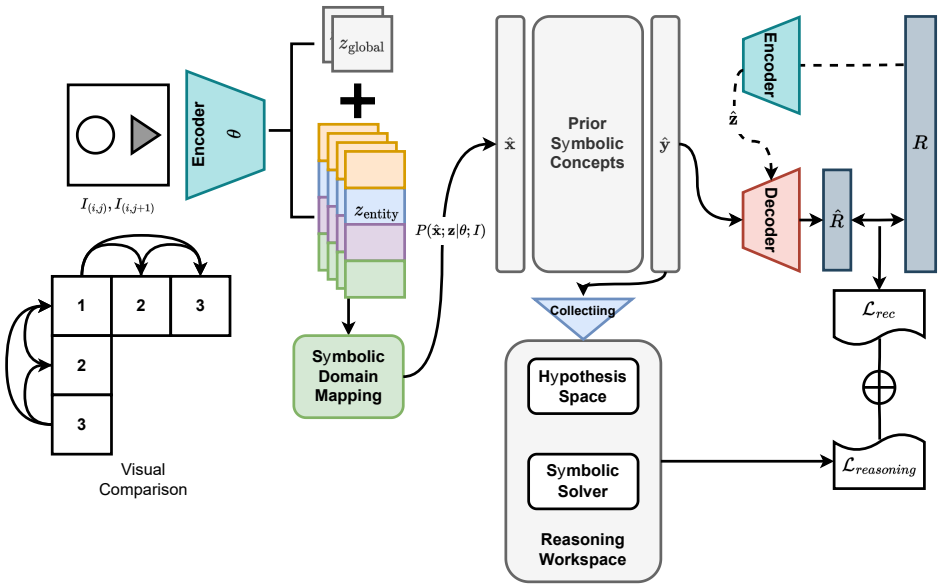


Figure. 12: An envision of future work

8.2 A Potential Solution

With all has been said, we visualise one potential future solutions by Figure 12. Provided the issues of NeurASP have been identified in this report, key aspects one might need to consider in future work using NPLP frameworks could be

1. provided the non-verbal nature of RPMs, how can we integrate DL models with logic programs in an unsupervised manner and get around with the domain enumeration problem?
2. What representation should we adopt in order to incorporate the semantic information available in neural embeddings to symbolic system more smoothly?
3. How can we leverage background knowledge to regularise DL models to learn the neural representations that are transferable (at least to different datasets of RPMs)?

For a potential solutions to the questions above, we consider a object-level representation of cells encoded by a disentangled encoder (Jiang and Ahn, 2020). While having the disentangled encoder and the

Neuro-symbolic framework in (Agarwal et al., 2021) provides us a possible way to integrate DL models with logic programs without supervisory signals, the learnt disentangled representations can capture the attributive values of individual objects in a distinguished manor. Since relations between objects can be visually determined by comparison (Lovett and Forbus, 2017), we could actually build mappings from attributive variations to symbolically encoded commonsense concepts , e.g. bigger than, via NPLP programs to be grounded by a discretised disentangled representations. Being able to imply what commonsense concepts the objects between cells in a row follow, we can further derive the actual underlying rules and consequently entail the right answer by these rules as we have seen in the current approach. In this case, we do not need hard labels for NN supervision. More importantly, jointly training disentangled encoder with commonsense concepts could inform the model with transferable knowledge, thus regularise the model to learn in a transferable way.

9 Conclusion

In this project, we first observed and argued that assumptions on perceptual output of the NeurASP framework do not align with the non-verbal nature of visual abstract reasoning problem. To lift the observed restrictions, we proposed an row-component-attribute wise representation with object-centric inductive bias and validity/inference ASP encodings as the first attempt of NeurASP to the RPM problem. We experimented our method on the I-RAVEN dataset and showed inferior results compared to state-of-the-art models. Apart from the strong assumptions made in our approaches, we analysed the potential reasons resulted to poor performances could mainly be the inadequate performance in perception and the cascading failure of joint probability. Finally, we envisioned experimentally and methodologically the future work could be done and visualised a potential solution to surging research questions of the NPLP formalism to solve the RPM problem.

10 Reflection

Overview. Having strong interests about neuro-symbolic AI led to the proposal of this project. However, the project was rather challenging for me from the following aspects:

– **First, so many things that I had to learn from scratch.** At a conceptual level, the project covers a wide range of advanced topics in AI that I had never touched with, for instances, representation learning, deep generative models and probabilistic logic programming etc. In order to understand the literature, it's obvious I have to learn from scratch by either lectures recordings online and extensively

reading relevant background literature in deepth (most of which published in top tier conferences in AI). Likewise, for the implementation, I learnt all by myself from scratch how to use advanced DL library such as pytorch and pyro to build DL models. Also, since training and running the system is computationally heavy, I needed to also learn how to use the high performance computing facility (Hawk) in order to deploy and experiment the system.

– **Second, things ran unexpectedly in most of the cases.** Before started, I did not expect the project would be as challenging given the intuition of using DL models as perceptual agents while ASP as the agents of logical reasoning is simple. Unexpectedly, many of the research challenges raised after background investigation about the RPM problem and NeurASP (as one can see in Section 5). Significant time spent before actually figuring out solutions to overcome the challenges and starting implementing as new problems keep raising one by one.

  Even potential solutions are found, implementation and experiments did not go smoothly as expected as well. For example, to reproduce the preprocessing methods proposed in (Spratley et al., 2020), I contacted the author for the missing yet critical part of the source code (the AIR module) but got no response. I ended up implementing the missing components myself with the descriptions and math formulae presented in (Spratley et al., 2020) and (Eslami et al., 2016). Similar situations happened as well when trying to adapt NeurASP, as mentioned in Section 7. Moreover, to run implementations and experiments deployed on Hawk, often times I have to join in a long queue for GPU nodes. Debugging and experiments easily take weeks as the resources got busier.

– **Third, time management and research organisation of such difficult project in such limited time are tough.** Given the amount of concepts, techniques and tools I had to investigate and the obstacles in implementation, how to decompose these packages into small chunks of tasks and arrange time doing them is another challenge. Even though I had started working on the project earlier, I found myself really short in time to finish it. Also, when experimenting the system,it was difficult to organise and keep experiments on track since multiple components with different setups of hyperparameters needed to be monitoring and recording. For instance, to get the best performed AIR model, different distribution parameters, latent size, learning rate were tried at first, results of different setups were messed because I did not keep the experiment running scripts in separation. In fact, I think I did not do well in both time management and experiment organisations.

Lesson Learnt.

– **Critically assessing and relating literature are important.** There are always knowledge one do not know about. When encountering plenty of new concepts, it is important to assess them critically and

relate them to each other. For example, although various different approaches to the RPM problem are proposed in deep learning research, they can be boiled down to the problem of learning good representations to capture the underlying relations. Knowing what is essential helps us to have a better understanding when confronting new concepts.

- **No one knows whether things would work or not before actually examining and trying the problem, so we should plan for the worst.** Given the intuition of adopting NeurASP to solve the RPM problem is simple, I was expecting things would be as simple. The initial plan was made naively, ended up nothing actually followed the plan. We shall therefore, be less optimistic and plan for the worst things could happen.
- **We shall not be too ambitious, focus on one thing at a time, and achieve the bigger goal step by step.** At first, I was too ambitious to include those fancy techniques that I had seen in literature, which actually complicated the problem (for example, the AIR module). I shall, however, trying first simple techniques like basic convolutional NNs to have a base solution, then using more advanced techniques to gradually improve on the base.
- **Failure and exceptions are normal.** Although it was frustrated to that the system achieved incompetent results, I believe through further empirical studies we can understand better the inferior performances. For example, conducting ablation studies on perceptual NNs to understand the effectiveness disentangled representations in our framework. However, due to the time constraint, I did not have enough time to conduct more experiments.
- **Every detail of the experiments is important, keep things in records even though that look trivial at the first glance.** When starting to experiment, models trained with different setups were messed with each other due to the lack of job script separation. Also, training checkpoints were saved for every 5 epochs in the AIR module. This led to the best performed model was omitted. In both of the scenarios, significant time was spent on re-running the experiments. Time could have been saved for taking careful logs on experiments.

Bibliography

Agarwal, A., Shenoy, P., and Mausam (2021). End-to-end neuro-symbolic architecture for image-to-image reasoning tasks. *CoRR*, abs/2106.03121.

Bengio, Y., Courville, A. C., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

Besold, T. R., d'Avila Garcez, A., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kühnberger, K., Lamb, L. C., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. (2021). Neural-symbolic learning and reasoning: A survey and interpretation. In Hitzler, P. and Sarker, M. K., editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 1–51. IOS Press.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P. A., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6.

Booch, G., Fabiano, F., Horesh, L., Kate, K., Lenchner, J., Linck, N., Loreggia, A., Murugesan, K., Mattei, N., Rossi, F., and Srivastava, B. (2021). Thinking fast and slow in AI. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 15042–15046. AAAI Press.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., and Schaub, T. (2020). Asp-core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309.

Carpenter, P. A., Just, M. A., and Shell, P. (1990). What one intelligence test measures: A theoretical account of the processing in the raven progressive matrices test. page 28.

d'Avila Garcez, A. S., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *FLAP*, 6(4):611–632.

Dries, A., Kimmig, A., Meert, W., Renkens, J., den Broeck, G. V., Vlasselaer, J., and Raedt, L. D. (2015). Problog2: Probabilistic logic programming. In Bifet, A., May, M., Zadrozny, B., Gavaldà, R., Pedreschi, D., Bonchi, F., Cardoso, J. S., and Spiliopoulou, M., editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September*

7-11, 2015, Proceedings, Part III, volume 9286 of *Lecture Notes in Computer Science*, pages 312–315. Springer.

Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. (2016). Attend, infer, repeat: Fast scene understanding with generative models. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3225–3233.

Falkenhainer, B., Forbus, K. D., and Gentner, D. (1986). The structure-mapping engine. In Kehler, T., editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 272–277. Morgan Kaufmann.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

Hofstadter, D. R. and Mitchell, M. (1994). The copycat project: A model of mental fluidity and analogy-making.

Hu, S., Ma, Y., Liu, X., Wei, Y., and Bai, S. (2021). Stratified rule-aware network for abstract visual reasoning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 1567–1574. AAAI Press.

Jiang, J. and Ahn, S. (2020). Generative neurosymbolic machines. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

John and Raven, J. (2003). *Raven Progressive Matrices*, pages 223–237.

Kahneman, D. (2011). *Thinking, fast and slow.* Thinking, fast and slow. Farrar, Straus and Giroux.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada,*

April 14-16, 2014, Conference Track Proceedings.

Lee, J., Talsania, S., and Wang, Y. (2017). Computing LPMLN using ASP and MLN solvers. *Theory Pract. Log. Program.*, 17(5-6):942–960.

Lifschitz, V. (2008). What is answer set programming? *In Proceedings of the 23rd National Conference on Artificial Intelligence*, Volume 3:4.

Lovett, A. and Forbus, K. (2017). Modeling visual problem solving as analogical reasoning. *Psychological Review*, 124(1):60–90.

Lovett, A., Forbus, K., and Usher, J. (2007). Analogy with qualitative spatial representations can simulate solving raven's progressive matrices. *Proceedings of the Annual Meeting of the Cognitive Science Society*, 29(29).

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and Raedt, L. D. (2021a). Neural probabilistic logic programming in deepproblog. *Artif. Intell.*, 298:103504.

Manhaeve, R., Marra, G., Demeester, T., Dumancic, S., Kimmig, A., and Raedt, L. D. (2021b). Neuro-symbolic AI = neural + logical + probabilistic AI. In Hitzler, P. and Sarker, M. K., editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 173–191. IOS Press.

Matzen, L. E., Benz, Z. O., Dixon, K. R., Posey, J., Kroger, J. K., and Speed, A. E. (2010). Recreating raven's: Software for systematically generating large numbers of raven-like matrix problems with normed properties. *Behavior Research Methods*, 42(2):525–541.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Magazine*, 27(4):12.

Meo, M., Roberts, M. J., and Marucci, F. S. (2007). Element salience as a predictor of item difficulty for raven's progressive matrices. *Intelligence*, 35(4):359–368.

Mitchell, M. (2021). Abstraction and analogy-making in artificial intelligence. *CoRR*, abs/2102.10717.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Raedt, L. D. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Mach. Learn.*, 100(1):5–47.

Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3788–3800.

Sahu, P. and Pavlovic, V. (2021). Daren: A collaborative approach towards reasoning and disentangling. *CoRR*, abs/2109.13156.

Santoro, A., Hill, F., Barrett, D. G. T., Morcos, A. S., and Lillicrap, T. P. (2018). Measuring abstract reasoning in neural networks. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4477–4486. PMLR.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P. W., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4967–4976.

Sim, K., Gopalkrishnan, V., Chua, H. N., and Ng, S. (2009). Macs: Multi-attribute co-clusters with high correlation information. In Buntine, W. L., Grobelnik, M., Mladenic, D., and Shawe-Taylor, J., editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part II*, volume 5782 of *Lecture Notes in Computer Science*, pages 398–413. Springer.

Spratley, S., Ehinger, K., and Miller, T. (2020). *A Closer Look at Generalisation in RAVEN*, volume 12372 of *Lecture Notes in Computer Science*, page 601–616. Springer International Publishing.

Steenbrugge, X., Leroux, S., Verbelen, T., and Dhoedt, B. (2018). Improving generalization for abstract reasoning tasks using disentangled feature representations. *CoRR*, abs/1811.04784.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural discrete representation learning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6306–6315.

van Steenkiste, S., Locatello, F., Schmidhuber, J., and Bachem, O. (2019). Are disentangled representations helpful for abstract visual reasoning? In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14222–14235.

Wang, D., Jamnik, M., and Liò, P. (2020). Abstract diagrammatic reasoning with multiplex graph networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wang, K. and Su, Z. (2015). Automatic generation of raven's progressive matrices. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 903–909. AAAI Press.

Wang, P., Donti, P. L., Wilder, B., and Kolter, J. Z. (2019). Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and den Broeck, G. V. (2018). A semantic loss function for deep learning with symbolic knowledge. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5498–5507. PMLR.

Yang, Z., Ishay, A., and Lee, J. (2020). Neurasp: Embracing neural networks into answer set programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, page 1755–1762. International Joint Conferences on Artificial Intelligence Organization.

Yu, S., Mo, S., Ahn, S., and Shin, J. (2021). Abstract reasoning via logic-guided generation. *CoRR*, abs/2107.10493.

Zhang, C., Gao, F., Jia, B., Zhu, Y., and Zhu, S.-C. (2019a). Raven: A dataset for relational and analogical visual reasoning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5312–5322. IEEE.

Zhang, C., Jia, B., Gao, F., Zhu, Y., Lu, H., and Zhu, S. (2019b). Learning perceptual inference by contrasting. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1073–1085.

Zhang, C., Jia, B., Zhu, S., and Zhu, Y. (2021). Abstract spatial-temporal reasoning via probabilistic abduction and execution. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 9736–9746. Computer Vision Foundation / IEEE.

Zhang, Z., Luo, P., Loy, C. C., and Tang, X. (2014). Facial landmark detection by deep multi-task learning. In Fleet, D. J., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, volume 8694 of *Lecture Notes in Computer Science*, pages 94–108. Springer.

Zheng, K., Zha, Z., and Wei, W. (2019). Abstract reasoning with distracting features. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5834–5845.