A Modular Neurosymbolic Approach for Visual Graph Question Answering

Thomas Eiter, Nelson Higuera Ruiz and Johannes Oetsch

Vienna University of Technology (TU Wien), Favoritenstrasse 9–11, Vienna, 1040, Austria

Abstract

Images containing graph-based structures are a ubiquitous and popular form of data representation that, to the best of our knowledge, have not yet been considered in the domain of Visual Question Answering (VQA). We use CLEGR, a graph question answering dataset with a generator that synthetically produces vertex-labelled graphs that are inspired by metro networks. Structured information about stations and lines is provided, and the task is to answer natural language questions concerning such graphs. While symbolic methods suffice to solve this dataset, we consider the more challenging problem of taking images of the graphs instead of their symbolic representations as input. Our solution takes the form of a modular neurosymbolic model that combines the use of optical graph recognition for graph parsing, a pretrained optical character recognition neural network for parsing node labels, and answer-set programming, a popular logic-based approach to declarative problem solving, for reasoning. The implementation of the model achieves an overall average accuracy of 73% on the dataset, providing further evidence of the potential of modular neurosymbolic systems in solving complex VQA tasks, in particular, the use and control of pretrained models in this architecture.

Keywords

neurosymbolic computation, answer-set programming, visual question answering

1. Introduction

Visual representations of structures that are based on graphs are a popular form of presenting information and ubiquitous in real life and on the internet. Examples include depictions of transit networks such as metro or train networks but graphs are truly everywhere. Visual Question Answering (VQA) [1] is concerned with inferring the correct answer to a natural language question in presence of some visual input such as an image or video. VQA enables applications in medicine, assistance for blind people, surveillance, and education [2]. The questions that arise in presence of graphs have been of interest to computer scientists since early days and are the basis of many complex systems. It is almost surprising that VQA tasks where the visual input contains a graph have, to the best of our knowledge, not been considered so far. The contribution of this paper is threefold: (i) we introduce **a novel VQA task that is concerned with images of graphs** that we call VGQA, (ii) we provide **a VGQA dataset**, and (iii) we present **a neurosymbolic VGQA approach** and thereby create a first baseline.

Implementation and data to reproduce the experiments is available at https://github.com/pudumagico/NSGRAPH.

NeSy'23: 17th International Workshop on Neural-Symbolic Learning and Reasoning, July 03–05, 2023, Siena, Italy

thomas.eiter@tuwien.ac.at (T. Eiter); nelson.ruiz@tuwien.ac.at (N. Higuera Ruiz); johannes.oetsch@tuwien.ac.at (J. Oetsch)

0000-0001-6003-6345 (T. Eiter); 0000-0003-3172-723X (N. Higuera Ruiz); 0000-0002-9902-7662 (J. Oetsch)

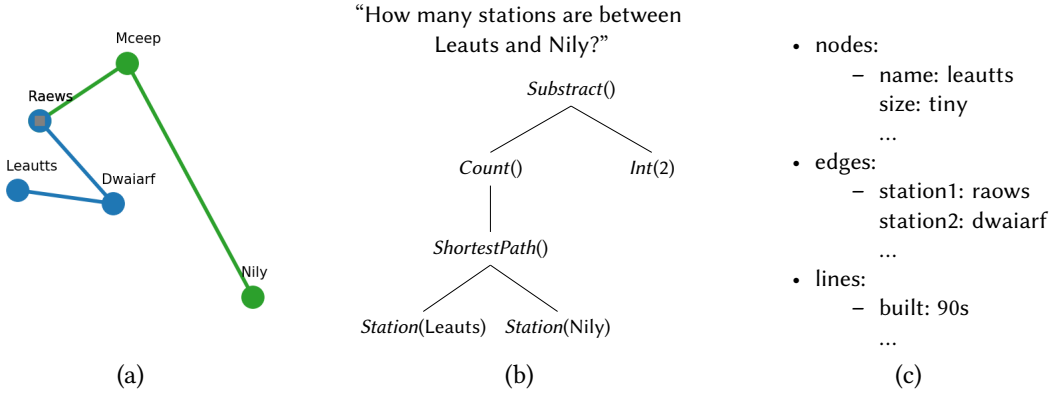


Figure 1: This figure presents an instance generated by CLEGR which consists of a colored and vertex-labelled graph, a natural language question, and additional information on the stations and lines. The colors in the graph indicate metro lines and the labels the names of the stations. The station with a gray square symbolizes a line change station. Below the natural language question is its functional representation. The nodes represent functions that takes as input the output of their leaves. The task is to answer the question (in this case, output"3") using the provided information.

VQA is driven by suitable datasets, where the images and questions are either synthetically generated or handcrafted. A well-known example is the CLEVR [3] dataset involving simple scenes. The dataset we are using for VQA on graphs is based on CLEGR [4], a CLEVR inspired dataset, with a generator that synthetically produces vertex-labelled graphs that are inspired by metro networks. Additional structured information about stations and lines, e.g., how big a station is, whether it is accessible for the disabled, when the line was constructed, etc., is provided. The task is to answer natural language questions concerning such graphs. For example, a question may ask for the shortest path between two stations while avoiding those that have a particular property. An illustration of a graph and a corresponding question is shown in Fig. 1.

Notably, instances of the CLEGR dataset are provided in symbolic form. While purely symbolic methods suffice to solve this dataset with ease (we present a particular approach in this paper), we consider the more challenging problem of taking images of the graphs instead of their symbolic representations as input. Therefore, we develop a solution of this dataset in the context of what we call *visual graph question answering* (VGQA). More specifically, we consider as input only an image of a graph as given in Fig. 1a with coloured nodes for stations and edges for connections between them. The colouration represents the metro line to which stations belong to. Each node in the image appears next to a label that represents the name of the station. For the questions, we only consider those that can be answered with information that can be found in the image, and we disregard all other symbolic information. The challenges to solve this VGQA dataset, we call it CLEGR^V, are threefold: (1) we have to parse the graph to identify nodes and edges, (2) we have to read and understand the labels and associate them which nodes of the graph, and (3) we have to understand the question and reason over the information extracted from the image to answer it accordingly.

Our solution follows a *neurosymbolic* methodology. Neurosymbolic computation [5] seeks to

unify the two main branches in artificial intelligence: statistical machine learning and automated logic-based reasoning. Modular neurosymbolic systems are those where the aforementioned parts are connected through some interface to combine their individual strengths.

In particular, our solutions takes the form of a modular neurosymbolic model that combines the use of *optical graph recognition* (OGR) [6] for graph parsing, a pretrained *optical character recognition* (OCR) [7] neural network for parsing node labels, and *answer-set programming* (ASP) [8], a popular logic-based approach to declarative problem solving, for reasoning. It operates in the following manner:

1. First, we use the OGR tool to parse the graph image into an abstract representation, structuring the information as sets of nodes and edges;
2. We use the OCR algorithm to obtain the text labels and associate them to the closest node;
3. Then, we parse the natural language question using regular expressions which suffices for the considered dataset as questions are structured in a rather simple way;
4. Finally, we use an encoding of the semantics of the question into a logic program which is, combined with the parsed graph and the question in symbolic form, given as input to an ASP solver, and from its output we obtain the answer to the question.

The implementation of the model achieves an overall average accuracy of 73.03% on the dataset, providing further evidence of the potential of modular neurosymbolic systems in solving complex VQA tasks, in particular, the use and control of pretrained models in this architecture. That our system does not require any training related to a particular set of examples—hence solving the dataset in an *zero-shot manner*—is a practical feature that hints to what may become the norm as large pretrained models are more than ever available for public use.

2. Visual Question Answering on Graphs

Graph Question Answering (GQA) is the task of answering a natural language question for a given graph in symbolic form. The graph consists of nodes and edges, but further attributes of nodes and edges may be additionally specified. A specific GQA dataset is CLEGR [4], which is concerned with graph structures that resemble transit networks like metro lines. Hence, the nodes correspond to stations and the edges represent lines going between stations. The questions are ones that are typically asked around mass transit like "How many stops are between **X** and **Y**?". The dataset is synthetic and comes with a generator that can be used produce instances of varying complexity.

Graphs come in the form of a YAML file containing records about attributes of the stations and lines. Each station has a name, a size, a type of architecture, a level of cleanliness, potentially disabled access, potentially rail access, and a type of music played. Stations can be described as relations over the aforementioned attributes. Edges connect stations but additionally have a colour, a line ID, and a line name. For lines, we have, besides name and ID, a construction year, a colour, and optional presence of air conditioning.

Examples of questions from the dataset are:

- Describe {Station} station's architectural style.
- How many stations are between {Station} and {Station}?
- Which {Architecture} station is adjacent to {Station}?

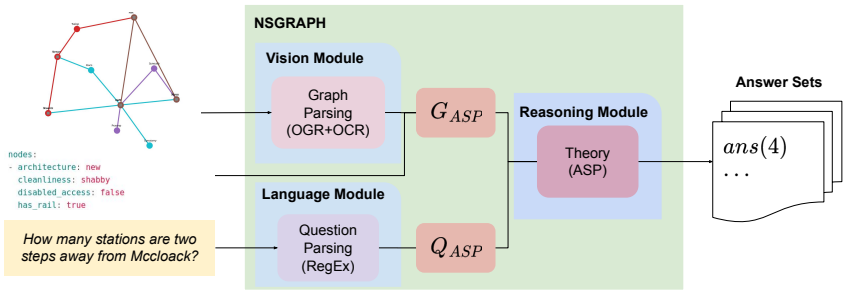


Figure 2: NSGRAPH system overview. The input is either an image of a graph or its symbolic description. The answer is generated by combining neural and symbolic methods.

- How many stations playing {Music} does {Line} pass through?
- Which line has the most {Architecture} stations?

For a full list of the questions, we refer the reader to the online repository of the dataset [4]. The answer to each question is of Boolean type, a number, a list, or a categorical answer. The questions in the dataset can be represented by functional programs, which allows us to decompose them into smaller and semantically less complex components. Figure 1 illustrates an instance of the CLEGR dataset.

Solving instances of the CLEGR dataset is not much of a challenge since all information is given in symbolic form, and we present a respective method later. But what if the graph is not available or given in symbolic form, but just as an image, as it is commonly the case? We define *Visual Graph Question Answering* (VGQA) as a GQA task where the input is a natural language question on a graph depicted in an image. We can in fact derive a challenging VGQA dataset, we call it CLEGR^V, from CLEGR by generating images of the transit graphs. To this end, we used the generator of the CLEGR dataset that has an option to produce images of the symbolic graphs.

Each image depicts stations, their names as labels in their proximity, and lines in different colours that connect them; an example is given in Fig. 1a. For the VGQA task, we drop all further symbolic information and consider only the subset of questions that can be answered with information from the graph image.

3. Our Neurosymbolic Framework for VQA on Graphs

We present our solution to VGQA tasks which we call NSGRAPH. It is a modular neurosymbolic system, where the modules are the typical ones used for a VQA adapted to our VGQA setting: a visual module, a language module, and a reasoning module. Figure 2 shows a summary of the inference process in NSGRAPH.

3.1. Visual Module

The visual model is used for graph parsing which consists of two sub-tasks: (i) detection of nodes and edges, and (ii) detection of labels, i.e., station names.

We employ an optical graph recognition (OGR) system for the first sub-task. In particular, we use a publicly available OGR script [9] that implements the approach due to Auer et al. [6]. The script takes an image as input and outputs the pixel coordinates of each node detected plus an adjacency matrix that contains the detected edges.

For the second sub-task of detecting labels, we use an optical character recognition (OCR) system, in particular, we use a pretrained neural network called EasyOCR [10] to obtain and structure the information contained in the graph image. The algorithm takes an image as input and produces the labels as strings together with their coordinates in pixels. We then connect the detected labels to the closest node found by the OGR system. Thereby, we obtain an abstract representation of the graph image as relations.

3.2. Language Module

The purpose of the language module is to parse the natural language question. It is written in plain Python and uses regular expressions to capture the variables in each question type. There are overall 35 different question templates that CLEGR may use to produce a question instance by replacing variables with names or attributes of stations, lines, or connections. Examples of question templates were already given in the previous section.

For illustration, the question template "How many stations are on the shortest path between S_1 and S_2?" may be instantiated by replacing S_1 and S_2 with station names that appear in the graph. We use regular expressions to capture those variables and furthermore translate the natural language question into a functional program that represents the semantics of the question by a tree of operations to answer the question. Continuing our example, we translate the template described above into the program

```
end(3). countNodesBetween(2). shortestPath(1). station(0,S1). station(0,S2).
```

where the the first numerical argument of each predicate imposes the order of execution of the associated operation and links the input of one operation to the output of the previous one. We can interpret this functional program as follows: the input to the shortest-path operation are two station names `S1` and `S2`. Its output are the stations on the shortest path between `S1` and `S2` which are counted in the next step. The predicate `end` represents the end of computation to yield this number as the answer to the question.

3.3. Reasoning Module

The third module consists of an ASP program that implements the semantics of the operations from the functional program of the question. Before we explain this reasoning component, we briefly review the basics of ASP.

3.3.1. Answer-Set Programming

Answer-Set Programming (ASP) [8, 11, 12] is a declarative logic-based approach for combinatorial search and optimisation problems with roots in knowledge representation and reasoning. It

offers a simple modelling language and efficient solvers[1]. To solve a problem with ASP, the search space and properties of problem solutions are described by means of a logic program such that its models, called *answer sets*, encode the problem's solutions.

An ASP program is a set of rules of the form

$$a_1 \mid \cdots \mid a_m \ :- \ b_1, \ldots, \ b_n, \ not\ c_1, \ldots, \ not\ c_n$$

where all a_i, b_j, c_k are first-order literals and *not* is *default negation*. The set of atoms left of $:-$ is the head of the rule, while the atoms to the right form the body. Intuitively, whenever all b_j are true, and there is no evidence for any c_l, then at least some a_i has to be true.

A rule with an empty body and a single head atom without variables is a *fact* and is always true. A rule with an empty head is a *constraint* and is used to exclude models that would satisfy the body.

ASP provides further language constructs like aggregates and weak (also called soft) constraints, whose violation should only be avoided. For a comprehensive coverage of the ASP language and its semantics, we refer to the language standard [13].

3.3.2. Question Encoding

The symbolic representations obtained from the language and visual modules are first translated into ASP facts. The functional program from a question is already in a fact format. The graph is translated in binary atoms `edge/2` and unary atoms `station/1` as well. These facts combined with an ASP program that encodes the semantics of all CLEGR questions templates can then be used to compute the answer with an ASP solver.

We present an excerpt of the ASP program that implements the semantics of the functional program `end(3), countNodesBetween(2), shortestPath(1), station(0,s), station(0,t)` from above. These atoms, together with facts for edges and nodes, serve as input to the ASP encoding for computing the answer as they only appear in rule bodies:

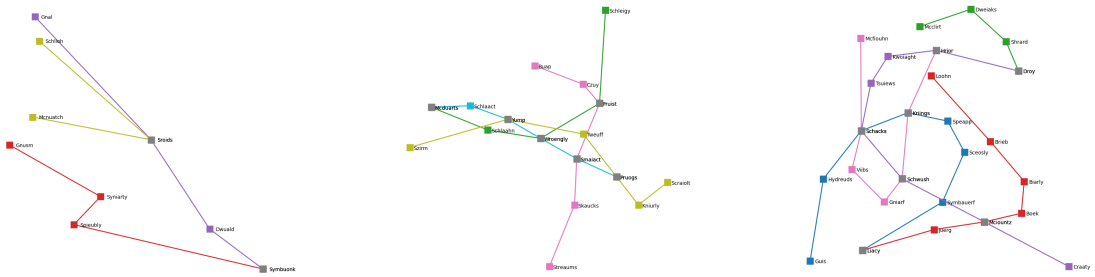
```
1  sp(T,S1,S2) :- shortestPath(T), station(T-1,S1), station(T-1,S2), S1<S2'.
2  in_path(T,S1,S2) | -in_path(T,S1,S2) :- edge(S1,S2), shortestPath(T).
3  reach(T,S1,S2) :- in_path(T,S1,S2).
4  reach(T,S1,S3) :- reach(T,S1,S2), reach(T,S2,S3).
5  :- sp(T,S1,S2), not reach(T,S1,S2).

6  cost(T,C) :- C = #count { S1,S2: in_path(T,S1,S2) }, shortestPath(T).
7  :~ cost(T,C). [C,T]

8  countedNodes(T,C-1) :- countNodesBetween(T), shortestPath(T-1), cost(T-1,C).
9  ans(N) :- end(T), countedNodes(T,N).
```

The first rule expresses that if we see `shortestPath(T)` in the input, then we want to compute the shortest path between station `S1` and `S2`. The actual shortest path is produced by rule 2 which non-deterministically decides for every edge if this edge is part of the shortest path. Rules 3 and 4 jointly define the transitive closure of this path relation and the constraint in

[1]See, for example, www.potassco.org or www.dlvsystem.com.



(a) (b) (c)

Figure 3: Examples of graphs of size tiny (3a), small (3b), and medium (3c).

Graph Size/Setting	NSGRAPH	OCR+GT	OGR+GT	Full GT
Tiny	80.9%	90.2%	83.1%	100%
Small	71.0%	85.2%	72.7%	100%
Medium	67.2%	83.8%	70.5%	100%
Overall	73.0%	86.4%	75.4%	100%

Table 1
Accuracy results of NSGRAPH on our VGQA dataset for tiny, small, and medium sized graphs. For OCR+GT, we replaced the OGR input with its symbolic ground truth. Likewise, we use the ground truth for OCR for OGR+GT, and Full GT stands for ground truth only.

line 5 enforces that station S1 is reachable from S2. Line 7 is a weak constraint that minimises the number of edges that are selected for the shortest path (without, we would not get a shortest path at all). This number of edges is calculated by rule 6 using an aggregate expression for counting. Finally, rule 8 calculates the number stations on the shortest path, and rule 9 defines the answer to the question as that number. We omit the full encoding for space reasons but, it is part of the online repository of this project (https://github.com/pudumagico/NSGRAPH).

4. Experiments

We experimentally evaluated NSGRAPH on GQA and VGQA tasks as described in Section 2. In particular, we generated graphs that fall into three categories: tiny (3 lines and at most 4 stations per line), small (4 and at most 6 stations per line), and medium (5 lines and at most 8 stations per line). We generate 100 graphs of each size accompanied by 10 questions per graph, with a median of 10 nodes and 8 edges for tiny graphs, 15 nodes and 15 edges for small graphs, and 24 nodes with 26 edges for medium ones. Figure 3 shows three graphs, one of each size.

NSGRAPH achieves 100% on the original GQA task, i.e., with graphs in symbolic form as input and with the unrestricted set of questions. Here, the symbolic input is translated directly into ASP fact without the need to parse an image.

For the more challenging VGQA task, we summarise the results in Table 1[2]. The task get more difficult with increasing size of the graphs, but we still achieve an overall accuracy of 73%. As we also consider settings where we replace the OCR, resp., OGR module, with the ground truth as input, we are able to pinpoint to the OGR as the main reason for wrong answers. The average run time to answer a question was 5.86s for tiny graphs, 14s for small graphs, and 35.33s for medium graphs. NSGRAPH is the first baseline for this VGQA dataset and further improvements a certainly possible, e.g., by leveraging the modularity of NSGRAPH, stronger OGR systems could be used.

5. Related Work

Our approach follows ideas from previous work [15], where we introduced a similar neurosymbolic method for VQA in the context of the CLEVR dataset [3]. As here, the reasoning component there is logic-based and implemented in ASP. This work was in turn inspired by NSVQA [16] that uses a combination of RCNN [17] for object detection, an LSTM [18] for natural language parsing, and Python as a symbolic executor to infer the answer. In this related work, the visual modules are trained for the dataset, while here we use merely a pretrained network. We also mention in this context the neural and end-to-end trainable system MAC [19] that achieves very promising results on VQA datasets. A very recent approach that combines large pre-trained models for images and text in combination with symbolic execution in Python is ViperGPT [20]; complicated images of graphs would presumably require some fine-tuning for that approach.

A characteristic feature of NSGRAPH is that we use ASP for reasoning. Outside the context of VQA, ASP has been applied for various neurosymbolic tasks like validation of electric panels [21], segmentation of laryngeal images [22], discovering rules that explain sequences of sensory input [23], or query answering with large language models [24]. There are also systems that can be used for neurosymbolic learning, e.g., by employing the semantic loss [25], which means they use the information produced by the reasoning module to enhance the learning tasks of the neural networks involved [26, 27].

6. Discussion and Future Work

We introduced a novel VQA problem that we call visual graph question answering (VGQA), where the input is an image of a graph along with a natural language question. Also, we introduced a respective dataset for this task that is based on an existing one for graph question answering on transit networks, and we presented NSGRAPH, a modular neurosymbolic model for VGQA that combines neural components for graph parsing and symbolic reasoning with ASP for question answering. We evaluate NSGRAPH on the VQGA dataset and thus create a first baseline. The advantages of a modular architecture are that the solution is transparent, interpretable, easier to debug, and components can be replaced with better ones over time in contrast to more monolithic end-to-end trained models. Our system notably relies on pretrained components and thus requires no additional training for the VGQA task. With the

[2]We ran the experiments on a computer with 32GB RAM, 12th Gen Intel Core i7-12700K, and a NVIDIA GeForce RTX 3080 Ti, and we are using clingo (v. 5.6.2) [14] as ASP solver.

advent of large pretrained models for language and images like GPT-4 [28] or CLIP [29], such architectures, where symbolic systems are used to control and connect neural ones, may be seen more frequently.

As this is work in progress, quite a number of topics remain for future work. While we advocate neurosymbolic methods, we would also like to see a comparison with purely neural end-to-end trained methods for VGQA. The performance of NSGRAPH is promising but also leaves room for improvement: We want to look into better alternatives for the visual module which is currently the limiting factor. Also, our approach to parse questions with regular expressions will not generalise well, and a large language model could be adopted for this purpose instead. The VGQA dataset is based on random graphs which do not always resemble real transit networks. We plan to improve on this, e.g., by using images of real metro maps.

References

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, D. Parikh, VQA: visual question answering, in: 2015 IEEE International Conference on Computer Vision, ICCV 2015, IEEE Computer Society, 2015, pp. 2425–2433. doi:10.1109/ICCV.2015.279.

[2] S. Barra, C. Bisogni, M. D. Marsico, S. Ricciardi, Visual question answering: Which investigated applications?, Pattern Recognit. Lett. 151 (2021) 325–331. doi:10.1016/j.patrec.2021.09.008.

[3] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, R. B. Girshick, CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, IEEE Computer Society, 2017, pp. 1988–1997. doi:10.1109/CVPR.2017.215.

[4] D. Mack, A. Jefferson, CLEVR graph: A dataset for graph question answering, 2018. URL: https://github.com/Octavian-ai/clevr-graph.

[5] L. D. Raedt, S. Dumancic, R. Manhaeve, G. Marra, From statistical relational to neuro-symbolic artificial intelligence, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 4943–4950.

[6] C. Auer, C. Bachmaier, F. J. Brandenburg, A. Gleißner, J. Reislhuber, Optical graph recognition, in: Graph Drawing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 529–540.

[7] A. M. Sabu, A. S. Das, A survey on various optical character recognition techniques, in: 2018 Conference on Emerging Devices and Smart Systems (ICEDSS), 2018, pp. 152–155. doi:10.1109/ICEDSS.2018.8544323.

[8] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[9] F. Chodziutko, K. Nowakowski, Optical Graph Recognition (OGR) - script, 2020. URL: https://github.com/praktyka-zawodowa-2020/optical_graph_recognition.

[10] Jaided AI, EasyOCR, 2022. URL: https://https://github.com/JaidedAI/EasyOCR.

[11] V. Lifschitz, Answer Set Programming, Springer, 2019.

[12] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis

Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012. doi:10.2200/S00457ED1V01Y201211AIM019.

[13] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. doi:10.1017/S1471068419000450.

[14] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory Solving Made Easy with Clingo 5, in: Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016), volume 52 of *OASIcs*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:15. doi:10.4230/OASIcs.ICLP.2016.2.

[15] T. Eiter, N. Higuera, J. Oetsch, M. Pritz, A neuro-symbolic ASP pipeline for visual question answering, Theory Pract. Log. Program. 22 (2022) 739–754. doi:10.1017/S1471068422000229.

[16] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, J. Tenenbaum, Neural-symbolic VQA: disentangling reasoning from vision and language understanding, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 2018, pp. 1039–1050.

[17] S. Ren, K. He, R. B. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, 2015, pp. 91–99.

[18] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[19] D. A. Hudson, C. D. Manning, Compositional attention networks for machine reasoning, in: 6th International Conference on Learning Representations, (ICLR 2018), OpenReview.net, 2018.

[20] D. Surís, S. Menon, C. Vondrick, ViperGPT: Visual Inference via Python Execution for Reasoning, CoRR abs/2303.08128 (2023). doi:10.48550/arXiv.2303.08128. arXiv:2303.08128.

[21] V. Barbara, D. Buelli, M. Guarascio, S. Ierace, S. Iiritano, G. Laboccetta, N. Leone, G. Manco, V. Pesenti, A. Quarta, F. Ricca, E. Ritacco, A loosely-coupled neural-symbolic approach to compliance of electric panels, in: Proceedings of the 37th Italian Conference on Computational Logic, volume 3204 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 247–253.

[22] P. Bruno, F. Calimeri, C. Marte, M. Manna, Combining deep learning and asp-based models for the semantic segmentation of medical images, in: Proceedings of the 5th International Joint Conference on Rules and Reasoning, RuleML+RR 2021, volume 12851 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 95–110. doi:10.1007/978-3-030-91167-6_7.

[23] R. Evans, J. Hernández-Orallo, J. Welbl, P. Kohli, M. J. Sergot, Making sense of sensory input, Artif. Intell. 293 (2021) 103438. doi:10.1016/j.artint.2020.103438.

[24] A. Rajasekharan, Y. Zeng, P. Padalkar, G. Gupta, Reliable natural language understanding with large language models and answer set programming, CoRR abs/2302.03780 (2023). doi:10.48550/arXiv.2302.03780. arXiv:2302.03780.

[25] J. Xu, Z. Zhang, T. Friedman, Y. Liang, G. V. den Broeck, A semantic loss function for deep learning with symbolic knowledge, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 5498–5507.

[26] Z. Yang, A. Ishay, J. Lee, Neurasp: Embracing neural networks into answer set program-

ming, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 1755–1762. doi:10.24963/ijcai.2020/243.

[27] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. D. Raedt, Deepproblog: Neural probabilistic logic programming, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 2018, pp. 3753–3763.

[28] OpenAI, Gpt-4 technical report, 2023. arXiv:2303.08774.

[29] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, I. Sutskever, Learning transferable visual models from natural language supervision, in: Proceedings of the 38th International Conference on Machine Learning, ICML 2021, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 8748–8763.