Neuro-Symbolic Visual Graph Question Answering with LLMs for Language Parsing

Jakob Johannes Bauer, Thomas Eiter, Nelson Higuera Ruiz, and Johannes Oetsch

Vienna University of Technology (TU Wien),
Favoritenstrasse 9–11, Vienna, 1040, Austria
{jakob.bauer,thomas.eiter,nelson.ruiz,johannes.oetsch}@tuwien.ac.at

Abstract. Images containing graph-based structures are an 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 provide a respective novel dataset and present a modular neuro-symbolic approach as a first baseline. Our dataset extends CLEGR, an existing dataset for question answering on graphs inspired by metro networks. Notably, the graphs there are given in symbolic form, while we consider the more challenging problem of taking images of graphs as input. Our solution combines optical graph recognition for graph parsing, a pre-trained optical character recognition neural network for parsing node labels, and answer-set programming for reasoning. The model achieves an overall average accuracy of 73% on the dataset. While regular expressions are sufficient to parse the natural language questions, we also study various large-language models to obtain a more robust solution that also generalises well to variants of questions that are not part of the dataset. Our evaluation provides further evidence of the potential of modular neuro-symbolic systems, in particular with pre-trained models, to solve complex VQA tasks.[1]

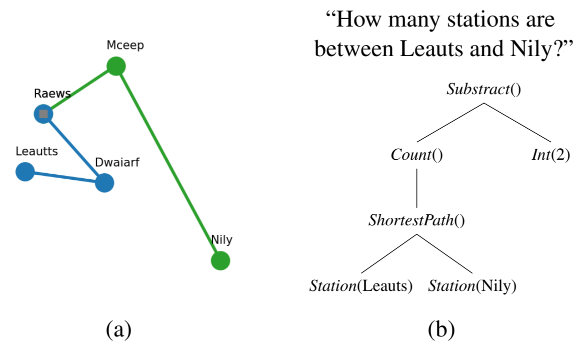
Keywords: neuro-symbolic computation · answer-set programming · visual question answering · large-language models

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 everywhere. Visual Question Answering (VQA) [1] is concerned with inferring the correct answer to a natural language question in the presence of some visual input such as an image or video. VQA enables applications in medicine, assistance for blind people, surveillance, and education [4]. 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.

VQA is driven by suitable datasets, where the images and questions are either synthetically generated or handcrafted. A well-known example is the CLEVR [17] dataset

[1] Code and data can be found at `https://github.com/pudumagico/NSGRAPH`.



- nodes:
  - name: leautts
    size: tiny
    ...
- edges:
  - station1: raows
    station2: dwaiarf
    ...
- lines:
  - built: 90s
    ...

(c)

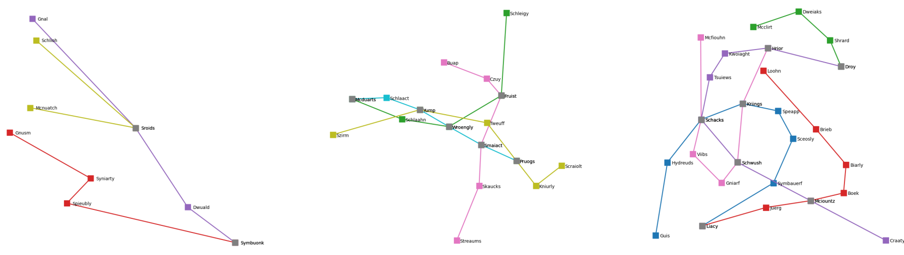
Fig. 1: An instance generated by CLEGR which consists of a colored and vertex-labelled graph, a natural language question, and additional information on the stations and lines. The colors in the graph indicate metro lines and the labels the names of the stations. The station with a gray square symbolises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

involving simple scenes. The dataset we are using for VQA on graphs is based on CLEGR [20], 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 large a station is, whether it is accessible to disabled people, 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.

In particular, instances of the CLEGR dataset are provided in symbolic form. While purely symbolic methods suffice to solve this dataset with ease (we present one in this paper), we consider the more challenging problem of taking images of the graphs instead of their symbolic representations as input. One such image is given in Fig. 1a For the questions, we only consider those that can be answered with information that can be found in the image and 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 with 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 solutions takes the form of a modular neuro-symbolic model [24] that combines the use of *optical graph recognition* (OGR) [2] for graph parsing, a pre-trained *optical character recognition* (OCR) [27] neural network for parsing node labels, and *answer-set programming* (ASP) [5], 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;



(a) (b) (c)

Fig. 2: Examples of graphs of size tiny (2a), small (2b), and medium (2c).

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 CLEGR^V.

While regular expressions are sufficient to parse the natural language questions, we also study various large-language models (LLMs) based on the transformer architecture [29] to obtain a more robust solution that also generalises well to variants of questions that are not part of the dataset. Our approach to using LLMs follows related work [25] and relies on prompting an LLMs to extract relevant ASP predicates from the question. We evaluated this approach on questions based on CLEGR and new questions obtained from a questionnaire.

The contribution of this paper is threefold:

(i) We introduce **VGQA, a novel VQA task that is concerned with images of graphs**, together with **a VGQA dataset**;
(ii) We present **a neuro-symbolic VGQA approach** and thus create a first baseline;
(iii) We evaluate **LLMs for language parsing** for the VGQA task.

This work provides further evidence of the potential of modular neuro-symbolic systems, in particular with pre-trained models, for solving complex VQA tasks. 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 pre-trained 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 may be specified in addition. A specific GQA dataset is CLEGR [20], 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 to 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:

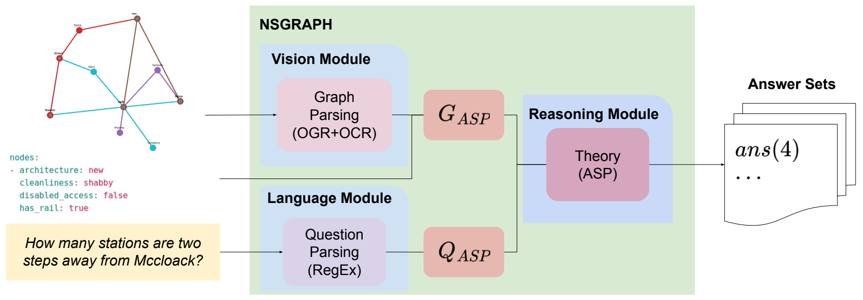
– Describe {Station} station's architectural style.
– How many stations are between {Station} and {Station}?
– Which {Architecture} station is adjacent to {Station}?
– How many stations playing {Music} does {Line} pass through?
– Which line has the most {Architecture} stations?

For a full list of the questions, we refer the reader to the online repository of the dataset [20]. 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 example from the data set CLEGR that includes such a functional program.

Visual Graph Question Answering. 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 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.

The NSGRAPH dataset. We can in fact derive a challenging VGQA dataset from CLEGR by generating images of the transit graphs. To this end, we used the generator of the CLEGR dataset that can also produce images of the symbolic graphs. Each image shows 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. We call the resulting dataset CLEGR^V; it is available online at `https://github.com/pudumagico/NSGRAPH`.

In particular, CLEGR^V consists 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 2 shows three graphs, one of each size.


```
nodes:
- architecture: new
  cleanliness: shabby
  disabled_access: false
  has_rail: true
```

How many stations are two steps away from Mccloack?

Fig. 3: 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.

3 Our Neuro-Symbolic Framework for VQA on Graphs

We present our solution to the VGQA tasks which we call NSGRAPH. It is a modular neuro-symbolic 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 3 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 subtasks: (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 subtask. In particular, we use a publicly available OGR script [8] that implements the approach due to Auer et al. [2]. The script takes an image as input and outputs the pixel coordinates of each detected node plus an adjacency matrix that contains the detected edges.

For the second subtask of detecting labels, we use an optical character recognition (OCR) system, namely, we use a pre-trained neural network called EasyOCR [16] 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 type of question. There are in general 35 different question templates that CLEGR can 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.

Table 1: ASP questions encodings for the twelve types of questions.

ASP Facts	Question
end(3). countNodesBetween(2). shortestPath(1). station(0,{ }). station(0,{ })	How many stations are between ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). withinHops(1, 2). station(0,{ })	How many other stations are two stops or closer to ([a-zA-Z]+)?
end(2). paths(1). station(0,{ }). station(0,{ })	How many distinct routes are there between ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). cycle(1). station(0,{ })	Is ([a-zA-Z]+) part of a cycle?
end(2). adjacent(1). station(0,{ }). station(0,{ })	Are ([a-zA-Z]+) and ([a-zA-Z]+) adjacent?
end(2). adjacentTo(1). station(0,{ }).station(0,{ })	Which station is adjacent to ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). commonStation(1). station(0,{ }). station(0,{ })	Are ([a-zA-Z]+) and ([a-zA-Z]+) connected by the same station?
end(2). exist(1). station(0,{ })	Is there a station called ([a-zA-Z0-9]+)?
end(2). linesOnNames(1). station(0,{ })	Which lines is ([a-zA-Z]+) on?
end(2). linesOnCount(1). station(0,{ })	How many lines is ([a-zA-Z]+) on?
end(2). sameLine(1). station(0,{ }). station(0,{ })	Are ([a-zA-Z]+) and ([a-zA-Z]+) on the same line?
end(2). stations(1). line(0,{ })	Which stations does ([a-zA-Z]+) pass through?

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 is two station names `S1` and `S2`. Its outputs 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 the computation to yield this number as the answer to the question. All considered question types and their ASP question encodings are summarised in Table 1.

Although this approach works well for all the questions in CLEGR, its ability to generalise to new types of questions is obviously limited; as a remedy, we discuss LLMs as an alternative to realise the language module in Section 4.

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.

Answer-Set Programming. *Answer-Set Programming* (ASP) [5, 18, 12] is a declarative logic-based approach to combinatorial search and optimisation with roots in knowledge representation and reasoning. It offers a simple modelling language and efficient solvers[2] In 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 solutions.

An ASP program is a set of rules of the form

$$a_1 \mid \cdots \mid a_m \text{ :- } b_1, \ldots, b_n, \; not\; c_1, \ldots, \; not\; c_n$$

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 must 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 [7].

Question Encoding. The symbolic representations obtained from the language and visual modules are first translated into ASP facts; we refer to them as G_{ASP} and Q_{ASP} in Fig.3, respectively. The functional program from a question is already in a fact format. The graph is translated into 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 question templates can 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 facts, together with ones for edges and nodes, serve as input to the ASP encoding for computing the answer as they only appear in rule bodies:

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

Table 2: Accuracy results of NSGRAPH on CLEGR^V 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.

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%

```
sp(T,S1,S2) :- shortestPath(T), station(T-1,S1),
               station(T-1,S2), S1<S2'.
{in_path(T,S1,S2)} :- edge(S1,S2),
                                        shortestPath(T).
reach(T,S1,S2) :- in_path(T,S1,S2).
reach(T,S1,S3) :- reach(T,S1,S2), reach(T,S2,S3).
:- sp(T,S1,S2), not reach(T,S1,S2).

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

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

The first rule expresses that if we see `shortestPath(T)` in the input, then we have to compute the shortest path between station `S1` and `S2`. This path is produced by the next rule which non-deterministically decides for every edge if this edge is part of the path. The following two rules jointly define the transitive closure of this path relation, and the constraint afterwards enforces that station `S1` is reachable from `S2` on that path. We use a weak constraint to minimise the number of edges that are selected and thus enforce that we indeed get a shortest path. The number of edges is calculated using an aggregate expression to count. Finally, the penultimate rule calculates the number of stations on the shortest path, as it takes as input the nodes that came out of the shortest path from the previous step and counts them, and the last rule defines the answer to the question as that number. The complete encoding is part of the online repository of this project (`https://github.com/pudumagico/NSGRAPH`).

3.4 Evaluation of NSGRAPH on CLEGR^V

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 facts without the need to parse an image.

We summarise the results for the more challenging VGQA task on CLEGR^V in Table 2.[3] The task becomes more difficult with increasing the 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 the OGR as the main reason for wrong answers. The average run time to answer a question was $5.86\,s$ for tiny graphs, $14\,s$ for small graphs, and $35.33\,s$ for medium graphs. NSGRAPH is the first baseline for this VGQA dataset and further improvements a certainly possible, e.g., stronger OGR systems could be used.

4 Predicate Extraction with LLMs

LLMs like GPT-4 [22] are deep neural networks based on the transformer architecture [29] with billions of parameters that are trained on a vast amount of data from the Internet to learn to predict the next token for a given text prompt. They have taken the world by storm because of their impressive capabilities for processing natural language. They are typically instructed via text prompts to perform a certain task like, e.g., answering a question or translating a text, but they can also be used for semantic parsing a text into a formal representation suitable for further processing.

In this section, we outline and evaluate an approach to use LLMs to realise the language module of NSGRAPH in a more robust way than by using regular expressions. First, we outline the general method of prompting LLMs to extract ASP predicates from questions. Afterwards, we evaluated this method for different LLMs, including state-of-the-art API-based ones but also open-source models that are free and can be locally installed.

4.1 Prompt Engineering

A particularly useful feature of LLMs is that the user can instruct them for a task by providing a few examples as part of the input prompt without the need to retrain the model on task-specific data; a property of LLMs commonly referred to as *in-context learning*.

Our approach uses in-context learning to instruct the LLM to extract the ASP atoms needed to solve the reasoning task from a question. This idea is inspired by recent work on LLMs for language understanding [25]. To obtain an answer to a question Q, we

(i) create a prompt P(Q) that contains the question Q along with additional instructions and examples for ASP question encodings,
(ii) pass P(Q) as input to an LLM and extract the ASP question encoding from the answer,
(iii) use extracted ASP facts together with the ASP rules described in the previous section to derive the answer.

The prompt P(Q) starts with a general pre-prompt that sets the stage for the task. In our case, this looks as follows:

[3] 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 used clingo (v. 5.6.2) [11] as ASP solver.

Table 3: Comparison of LLMs used in our evaluation.

Model	**Parameters**	**Open Source**	**Price**	**Company**	**Token Limit**
GPT-4	1.5×10^{12}	$\times$	USD 20 p/m	OpenAI	32 768
GPT-3.5	175×10^{9}	$\times$	free	OpenAI	4 096
Bard	1.6×10^{12}	$\times$	free	Google	2 048
GPT4ALL	7×10^{9}	✓	self hosted	Nomic AI	2 048
Vicuna 13b	13×10^{9}	✓	per request	Meta	2 048

```
You are now a Question Parser that translates natural language
questions into ASP ground truths about different stations.
Output only the ground truths and nothing else. The stations to
be selected from are arbitrary.
```

Afterwards, we provide a number of examples that illustrate what is expected from the LLM. In particular, we use one two three examples for each type of question in the dataset which amounts to 36 in-context examples in total. We only show an excerpt due to space reasons, the complete prompt is shown in the appendix:

```
I now provide you with some examples on how to parse Questions:

Q: ``How many stations are between Inzersdorf and Mainstation?''
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,``Inzersdorf'').station(0,``Mainstation'').

Q: ``What is the amount of stations between Station A and
Station B?''
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,``Station A'').station(0,``Station B'').
```

Finally, the prompt contains the questions that should be answered:

```
Now provide the output for the following question:
What are the stations that lie on line 7?
```

4.2 Evaluation

We next evaluate the method from the previous section to answer to following questions:

(R1) Is the method suitable for realising the language component of NSGRAPH?

(R2) What is the trade-off between grand scale LLMs and smaller and more cost-efficient alternatives?

(R3) How well does the method generalise to questions formulated in a different way than in CLEGR?

Before we present the results of our evaluation, we first discuss the different LLMs and the datasets that we used.

Overview of used LLMs. We compared five different models (GPT-4, GPT3.5, Bard, GPT4All, and Vicuna 13b) on the task of predicate extraction; information on these models is summarised in Table 3.

GPT-4[4] is the latest model developed by OpenAI with 1.5 trillion parameters and a context limit of 32 768 tokens. For a price of USD 20 per month, the ChatGPT Plus offer can be subscribed, enabling users to send up to 50 requests in a three-hour time frame to a hosted version of GPT-4.

GPT3.5[5] is the predecessor to OpenAIs GPT-4 and is available online for free. It utilizes 175 billion parameters and is capable contexts of 4 096 tokens.

Bard[6] is Google's answer to OpenAIs dominant LLMs, using slightly more parameters than GPT-4 while only having a context window of 2 048 tokens. It is free to its users, however, all EU states are currently excluded from using the service due to copyright concerns by the company.

GPT4All[7] is an open-source model that needs only 7 billion parameters. With a context limit of 2 048 tokens, it competes with Google Bard, however, there is no official hosted service to run this LLM. It was developed using the open source weights of Alpaca, a model developed and released by Meta. GPT-4 served as a training data generator for this model, making it a cheap alternative to expensive large-scale models.

Vicuna 13b[8] was developed and open-sourced by Meta and comes with 13 billion parameters and a context window of 2 018 tokens. It serves as a middle ground between large-scale LLMs and small alternatives like GPT4All. It is not hosted on an official server, but there are external services that host this model and even offer fine-tuning to user specific use cases.

Datasets. We created two datasets for our evaluation: CLEGR$^+$ and CLEGR-Human. The former is a straight-forward hand-crafted extension of the questions from the original CLEGR dataset. Besides original questions that can be parsed with regular expressions, the dataset also contains version where words are swapped with synonyms and the position of words is slightly changed, as well as questions that entirely rephrase the original ones. For example, "Are Station A and B on the same line?" could be rephrased as "Can I reach Station A from Station B without a line change?". The CLEGR$^+$ dataset consists of 74 questions in total.

CLEGR-Human is a dataset that was created using an online survey. The survey takers were presented with a metro map and a couple of example questions. After that, they had the task of formulating further questions such as "Ask about the distance between Station A and Station B" and answering their own questions. This enables cross-peer validation by having other users evaluate the same question and compare their answers. Each surveyor had to answer a total of 12 questions; 27 people from Austria, Switzerland, and Germany between the ages of 18 and 33 completed the survey, 22 of which were students. The dataset thus consists of 324 questions in total.

[4] `https://openai.com/research/gpt-4`
[5] `https://platform.openai.com/docs/models/gpt-3-5`
[6] `https://bard.google.com/`
[7] `https://gpt4all.io/index.html`
[8] `https://huggingface.co/lmsys/vicuna-13b-v1.3`

Results and Discussion. The results of our evaluation are summarised in Fig. 4 for the CLEGR$^+$ dataset and in Fig. 5 for CLEGR-Human. Data and instructions to reproduce our results are part of the online repository of the project. We classified the answers produced by the LLMs into four categories:

- *full match*: The response matches exactly the set of expected atoms.
- *contains solution*: The response contains the expected atoms.
- *task missed*: The response contains some text but not the expected atoms.
- *no answer*: The response consists of only whitespace characters.

Note that "full match" as well as "contains solution" can be used for the downstream reasoning task, while answers from the other categories cannot be used.

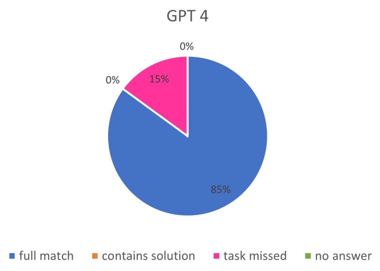
GPT-4 performed best among the considered LLMs as it produced 85% completely correct responses on CLEGR$^+$ and even 94% on CLEGR-Human. It also always provided an answer to the prompt. GPT-3.5 invented new predicates for half of the questions. For the remaining ones, its response matched exactly or contained the solution. Vicuna often did not give a proper response due to context overflow and trails behind GPT-4 and GPT-3.5 also in terms of correct answers. Google Bard never got the exact solution due to extensive additional explanations for all predicates no matter the prompt. However, the responses contained the solution in about three quarters of the cases. In this regard, it is only outmatched by GPT-4. GPT4All gives a response that contains the correct solution for only 23% and 16% of the questions from CLEGR$^+$ and CLEGR-Human, respectively.

We answer our initial research questions therefore as follows: At least GPT-4 is suitable for realising the language component with an acceptable trade-off between accuracy and ability to generalise (R1). Although GPT-4 exhibits the best overall performance, especially the free and much smaller GPT4ALL model shows promising results (R2). Throughout, the LLMs perform similarly on CLEGR$^+$ and CLEGR-Human which showcases the strength of LLMs for language processing without the need for context-specific training (R3).

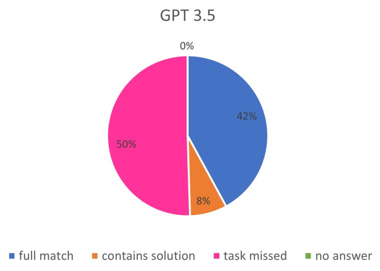
5 Related Work

Our approach follows ideas from previous work [9], where we introduced a similar neuro-symbolic method for VQA in the context of the CLEVR dataset [17]. As here, the reasoning component there is logic-based and implemented in ASP. This work was in turn inspired by NSVQA [32] that uses a combination of RCNN [26] for object detection, an LSTM [13] 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 pre-trained network. In this context, we also mention the neural and end-to-end trainable MAC system [14] that achieves very promising results in 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 [28]; complicated images of graphs would presumably require some fine-tuning for that approach.

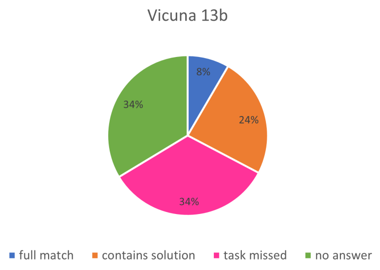
A feature of NSGRAPH is that we use ASP for reasoning. Outside the context of VQA, ASP has been applied for various neuro-symbolic tasks such as validation of electric panels [3], segmentation of laryngeal images [6], and discovery of rules that



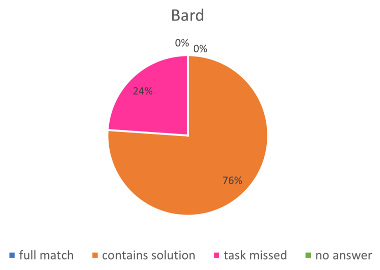
(a) GPT-4



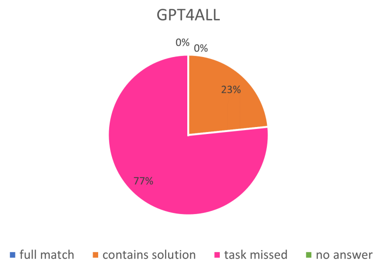
(b) GPT-3.5



(c) Vicuna 13b

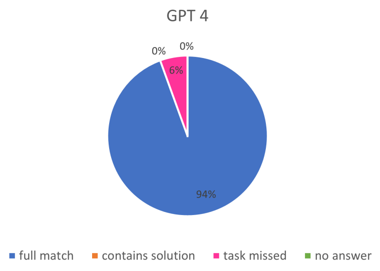


(d) Google Bard

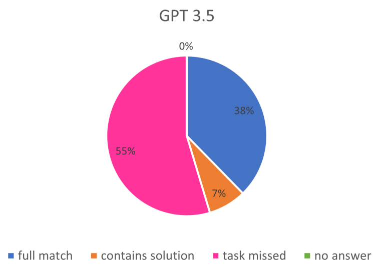


(e) GPT4All

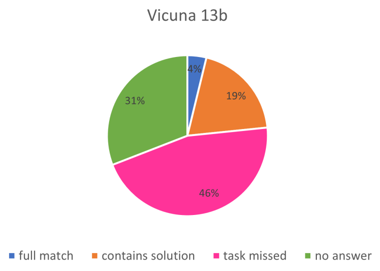
Fig. 4: Results of the evaluation on the CLEGR$^+$ dataset.



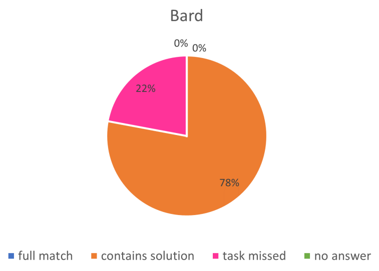
(a) GPT-4



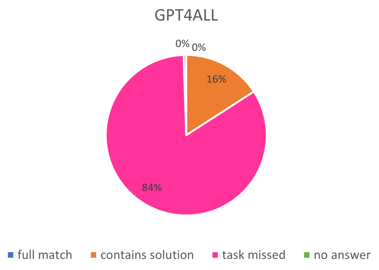
(b) GPT-3.5



(c) Vicuna 13b



(d) Google Bard



(e) GPT4All

Fig. 5: Results of the evaluation on the CLEGR-Human dataset.

explain sequences of sensory input [10]. There are also systems that can be used for neuro-symbolic learning, e.g., by employing semantic loss [30], which means that they use the information produced by the reasoning module to enhance the learning tasks of the neural networks involved [31, 21].

Our approach to using LLMs to extract predicates for the downstream reasoning task is inspired by recent work by Rajasekharan et al. [25]. They proposed the STAR framework which consists of LLMs and prompts for extracting logical predicates in combination with an ASP knowledge base. The authors applied STAR to different problems requiring qualitative reasoning, mathematical reasoning, as well as goal-directed conversation. Going one step further, Ishay et al. [15] introduced a method to translate problems formulated in natural language into complete ASP programs. This method requires multiple prompts with each responsible for a subtask such as identifying constant symbols, forming predicates, and transforming the specification into rules using these predicates. The idea to apply LLMs to parse natural language into a formal language suitable for automated reasoning is also found outside the context of ASP, we mention work by Liu et al. [19] who use prompting techniques to translate text into the Planning Domain Definition Language (PDDL) as an example.

6 Conclusion and Future Work

We introduced a novel VQA task that we call visual graph question answering (VGQA), where the input is an image of a graph along with a natural language question. For this task, we presented a respective dataset that is based on an existing one for graph question answering on transit networks, and we introduced NSGRAPH, a modular neuro-symbolic model for VGQA that combines neural components for graph and question parsing and symbolic reasoning with ASP for question answering. We studied LLMs for the question parsing component to improve how well our model generalises to unseen questions. NSGRAPH has been evaluated on the VGQA dataset and thus constitutes a first baseline for the novel dataset.

The advantages of a modular architecture are that the solution is transparent, interpretable, and 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 pre-trained components and thus requires no additional training. With the advent of large pre-trained models for language and images like GPT -4 [22] or CLIP [23], such architectures, where symbolic systems are used to control and connect neural ones, may be seen more frequently.

While we advocate neuro-symbolic methods, we also plan a comparison with purely neural end-to-end trained methods for VGQA for future work. The performance of NSGRAPH is promising but also leaves room for improvement. In particular, we plan to look into better alternatives for the visual module which is currently the limiting factor. The VGQA dataset is based on random graphs which do not always resemble real transit networks. We intend to also use images of real metro maps in the future.

Acknowledgments

This work was supported by funding from the Bosch Center for Artificial Intelligence.

References

1. Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C.L., Parikh, D.: VQA: visual question answering. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015. pp. 2425–2433. IEEE Computer Society (2015). https://doi.org/10.1109/ICCV.2015.279
2. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A., Reislhuber, J.: Optical graph recognition. In: Graph Drawing. pp. 529–540. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
3. Barbara, V., Buelli, D., Guarascio, M., Ierace, S., Iiritano, S., Laboccetta, G., Leone, N., Manco, G., Pesenti, V., Quarta, A., Ricca, F., Ritacco, E.: A loosely-coupled neural-symbolic approach to compliance of electric panels. In: Proceedings of the 37th Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 3204, pp. 247–253. CEUR-WS.org (2022)
4. Barra, S., Bisogni, C., Marsico, M.D., Ricciardi, S.: Visual question answering: Which investigated applications? Pattern Recognit. Lett. **151**, 325–331 (2021). https://doi.org/10.1016/j.patrec.2021.09.008
5. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195
6. Bruno, P., Calimeri, F., Marte, C., Manna, M.: 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. Lecture Notes in Computer Science, vol. 12851, pp. 95–110. Springer (2021). https://doi.org/10.1007/978-3-030-91167-6_7
7. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: Asp-core-2 input language format. Theory Pract. Log. Program. **20**(2), 294–309 (2020). https://doi.org/10.1017/S1471068419000450
8. Chodziutko, F., Nowakowski, K.: Optical Graph Recognition (OGR) - script (2020), `https://github.com/praktyka-zawodowa-2020/optical_graph_recognition`
9. Eiter, T., Higuera, N., Oetsch, J., Pritz, M.: A neuro-symbolic ASP pipeline for visual question answering. Theory Pract. Log. Program. **22**(5), 739–754 (2022). https://doi.org/10.1017/S1471068422000229
10. Evans, R., Hernández-Orallo, J., Welbl, J., Kohli, P., Sergot, M.J.: Making sense of sensory input. Artif. Intell. **293**, 103438 (2021). https://doi.org/10.1016/j.artint.2020.103438
11. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory Solving Made Easy with Clingo 5. In: Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016). OASIcs, vol. 52, pp. 2:1–2:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/OASIcs.ICLP.2016.2
12. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012). https://doi.org/10.2200/S00457ED1V01Y201211AIM019
13. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
14. Hudson, D.A., Manning, C.D.: Compositional attention networks for machine reasoning. In: 6th International Conference on Learning Representations, (ICLR 2018). OpenReview.net (2018)

15. Ishay, A., Yang, Z., Lee, J.: Leveraging large language models to generate answer set programs. In: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023). pp. 374–383 (2023). https://doi.org/10.24963/kr.2023/37
16. Jaided AI: EasyOCR (2022), `https://https://github.com/JaidedAI/EasyOCR`
17. Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C.L., Girshick, R.B.: CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. pp. 1988–1997. IEEE Computer Society (2017). https://doi.org/10.1109/CVPR.2017.215
18. Lifschitz, V.: Answer Set Programming. Springer (2019)
19. Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., Stone, P.: LLM+P: empowering large language models with optimal planning proficiency. CoRR **abs/2304.11477** (2023). https://doi.org/10.48550/arXiv.2304.11477
20. Mack, D., Jefferson, A.: CLEVR graph: A dataset for graph question answering (2018), `https://github.com/Octavian-ai/clevr-graph`
21. Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., Raedt, L.D.: Deepproblog: Neural probabilistic logic programming. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018. pp. 3753–3763 (2018)
22. OpenAI: Gpt-4 technical report (2023)
23. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021. Proceedings of Machine Learning Research, vol. 139, pp. 8748–8763. PMLR (2021)
24. Raedt, L.D., Dumancic, S., Manhaeve, R., Marra, G.: From statistical relational to neuro-symbolic artificial intelligence. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 4943–4950. ijcai.org (2020)
25. Rajasekharan, A., Zeng, Y., Padalkar, P., Gupta, G.: Reliable natural language understanding with large language models and answer set programming. CoRR **abs/2302.03780** (2023). https://doi.org/10.48550/arXiv.2302.03780
26. Ren, S., He, K., Girshick, R.B., Sun, J.: 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. pp. 91–99 (2015)
27. Sabu, A.M., Das, A.S.: A survey on various optical character recognition techniques. In: 2018 Conference on Emerging Devices and Smart Systems (ICEDSS). pp. 152–155 (2018). https://doi.org/10.1109/ICEDSS.2018.8544323
28. Surís, D., Menon, S., Vondrick, C.: ViperGPT: Visual Inference via Python Execution for Reasoning. CoRR **abs/2303.08128** (2023). https://doi.org/10.48550/arXiv.2303.08128
29. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017. pp. 5998–6008 (2017)
30. Xu, J., Zhang, Z., Friedman, T., Liang, Y., den Broeck, G.V.: A semantic loss function for deep learning with symbolic knowledge. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5498–5507. PMLR (2018)
31. Yang, Z., Ishay, A., Lee, J.: Neurasp: Embracing neural networks into answer set programming. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020. pp. 1755–1762. ijcai.org (2020). https://doi.org/10.24963/ijcai.2020/243

32. Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., Tenenbaum, J.: 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. pp. 1039–1050 (2018)

Appendix

A Example of a Complete Prompt

Question: “How many stations lie on line 7?”
Prompt:

```
You are now a Question Parser that translates natural language
questions into ASP ground truths about different stations.
Output only the ground truths and nothing else.
The stations to be selected from are arbitrary.
I now provide you with some examples on how to parse Questions:

Q: ``How many stations are between Inzersdorf and Mainstation?''
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,``Inzersdorf'').station(0,``Mainstation'').

Q: ``What is the amount of stations between Station A and
Station B?''
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,``Station A'').station(0,``Station B'').

Q: ``How many stations does it take to go from Johnsen
to St. Galeen?''
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,``Johnsen'').station(0,``St. Galeen'').

Q: ``How many other stations are two stops or closer to Kriau?''
A: end(2).withinHops(1, 2).station(0,``Kriau'').

Q: ``How many stations are within at most three stops to
Währingen?''
A: end(2).withinHops(1, 2).station(0,``Währingen'').

Q: ``How many stations can I reach within four stops from
Johannesburg?''
A: end(2).withinHops(1, 2).station(0,``Johannesburg'').

Q: ``How many distinct routes are there between Wien and
Linz?''
A: end(2).paths(1).station(0,``Wien'').station(0,``Linz'').

Q: ``How many different routes can I take from Wurzbach to
Saarlein?''
```

```
A: end(2).paths(1).station(0,``Wurzbach'').
station(0,``Saarlein'').

Q: ``What is the number of distinct ways I can reach
Fortright from Sortfeit?''
A: end(2).paths(1).station(0,``Fortright'').
station(0,``Sortfeit'').

Q: ``Is Klingon part of a cycle?''
A: end(2).cycle(1).station(0,``Klingon'').

Q: ``Is Sortfeit part of a loop?''
A: end(2).cycle(1).station(0,``Sortfeit'').

Q: ``Can I reach Genf from itself?''
A: end(2).cycle(1).station(0,``Genf'').

Q: ``Are Solomon and Kyrgistan adjacent?''
A: end(2).adjacent(1).station(0,``Solomon'').
station(0,``Kyrgistan'').

Q: ``Are Bahrain and India next to each other?''
A: end(2).adjacent(1).station(0,``Bahrein'').
station(0,``India'').

Q: ``Is Belemeth next to Sigualen?''
A: end(2).adjacent(1).station(0,``Belemeth'').
station(0,``Sigualen'').

Q: ``Which station is adjacent to Westbahnhof and
Station T''
A: end(2).adjacentTo(1).station(0,``Westbahnhof'').
station(0,``Station T'').

Q: ``Which station is directly next to Hausberg and
Fortright''
A: end(2).adjacentTo(1).station(0,``Hausberg'').
station(0,``Fortright'').

Q: ``Which station is adjacent to Greenville and
St. Georgen''
A: end(2).adjacentTo(1).station(0,``Greenville'').
station(0,``St. Georgen'').

Q: ``Are the green line and line 5 connected by the
same station?''
A: end(2).commonStation(1).station(0,``line green'').
station(0,``line 5'').

Q: ``Is there a common station between line 8 and
```

```
the yellow line?''
A: end(2).commonStation(1).station(0,``line 8'').
station(0,``line yellow'').

Q: ``Are the following lines connected: line 8,
line 2?''
A: end(2).commonStation(1).station(0,``line 8'').
station(0,``line 2'').

Q: ``Is there a station called Wurzbach?''
A: end(2).exist(1).station(0,``Wurzbach'').

Q: ``Does the station named Gerasdorf exist?''
A: end(2).exist(1).station(0,``Gerasdorf'').

Q: ``Is the station St. Gallen real?''
A: end(2).exist(1).station(0,``St. Gallen'').

Q: ``Which lines is Vorgartenstraße on?''
A: end(2).linesOnNames(1).
station(0,``Vorgartenstra\ss{}e'').

Q: ``Which lines are connected to Gro. Lorelai?''
A: end(2).linesOnNames(1).station(0,``Gro. Lorelai'').

Q: ``Tell me the lines on which Bern is on?''
A: end(2).linesOnNames(1).station(0,``Bern'').

Q: ``How many lines is Station Z on?''
A: end(2).linesOnCount(1).station(0,``Station Z'').

Q: ``How many lines is Hauptbahnhof connected to?''
A: end(2).linesOnCount(1).station(0,``Hauptbahnhof'').

Q: ``What is the number of lines that are connected
to Kleinbühl?''
A: end(2).linesOnCount(1).station(0,``Kleinbühl'').

Q: ``Are Station A and Station B on the same line?''
A: end(2).sameLine(1).station(0,``Station A'').
station(0,``Station B'').

Q: ``Do Hauptbahnhof and St. Margarethen share
the same line?''
A: end(2).sameLine(1).station(0,``Hauptbahnhof'').
station(0,``St. Margarethen'').

Q: ``Can I reach Basel from Zürich without
changing the line?''
A: end(2).sameLine(1).station(0,``Basel'').
```

```
station(0,``Zürich'').

Q: ``Which stations does Line 5 pass through?''
A: end(2).stations(1).line(0,``line 5'').

Q: ``Which stations does the green line pass through?''
A: end(2).stations(1).line(0,``line green'').

Q: ``Which stations are connected to the red line?''
A: end(2).stations(1).line(0,``line red'').

Now provide the output for the following question:
What are the stations that lie on line 7?
```