Kurzfassung

Diese Thesis untersucht den Einsatz von Large Language Models (LLMs) zur Übersetzung von Fragen aus natürlicher Sprache in logische Programme, mit dem Ziel, diese in modulare Visual Question Answering (VQA)-Pipelines zu integrieren. Sie demonstriert das Potenzial dieses Ansatzes zur symbolischen Fragendarstellung und untersucht Faktoren, die die Qualität der Übersetzung beeinflussen. Darüber hinaus trägt diese Arbeit zur Diskussion über Bewertung von VQA-Systemen bei, indem sie ein Validierungsframework mit Freiheitsgrad der Großzügigkeit vorschlägt und einen GQA datensatzspezifischen Interpreter der Frage präsentiert. Ausgewählte Fehlerfälle sowohl dieses Interpreters, als auch der allgemeinen Pipeline werden ebenfalls untersucht.

Die der Arbeit zugehörigen Experimente zeigen, dass GPT-4 in der Lage ist, selbst in der strengsten Konfiguration des Validierungsframeworks, etwa 82–85% der Fragen korrekt zu übersetzen. Darüber hinaus zeigt sie die Bedeutung von angemessen gewählten Prompts und in-context Beispielen, da ähnlichkeitsbasierte Samplingverfahren zufälliges Ziehen relativ um 150% übertreffen, während das Hinzufügen von vielfältigeren Beispielen bemerkbare, aber abnehmende Leistungssteigerungen zeigt. Um die potenziellen Anwendungen solcher Systeme zu erweitern, zeigt die Arbeit, dass ein Kompromiss hin zu kleineren Modellen in gewissen Fällen angebracht sein kann, da ein 3- und 7-Milliarden-Parameter Modell verglichen mit GPT-4 eine relative Erfolgsquote von 38.4% bzw. 84.7% erreicht.

Abstract

This thesis explores the use of LLMs for translating natural language questions to logical representations suitable for integration with modular VQA pipelines, as a neuro-symbolic question encoding method. It demonstrates feasibility and probes a variety of factors that influence the quality of the translation. In addition, this work contributes to the discussion on fair evaluation of VQA systems, by proposing a variably generous validation framework, alongside a semantic question evaluation engine for the GQA dataset. Select failure cases of both this engine and the general pipeline are also investigated.

The experiments accompanying this work show that GPT-4 is capable of translating around 82–85% of questions correctly with a strict evaluation criterion. Additionally, it demonstrates the importance of well-chosen prompts and in-context examples, as similarity-based sampling outperforms random sampling by a 150% relative margin, while adding diverse examples yields noticeable but diminishing returns. To broaden the potential applications of such systems, the thesis demonstrates that a tradeoff towards smaller models may be made, as a 3B and 7B-parameter model achieve a relative performance of 38.4% and 84.7%, respectively, when compared to GPT-4.

Contents

1 Introduction

This thesis explores the intersection of Large Language Model (LLM) translation, and neuro-symbolic Visual Question Answering (VQA). This section motivates this to be a promising research direction and presents the questions the thesis aims to answer.

1.1 Motivation

Understanding and answering natural language questions has long been a goal of Artificial Intelligence (AI) research. Specifically VQA, which requires the understanding of images and natural language simultaneously, has been a popular research topic in recent years.

Reliable VQA systems would have massive implications for a wide range of applications, such as autonomous driving, robotics and accessibility, yet the current state of the art lags far behind human performance even in toy examples (Hudson and Manning 2019). Several approaches have been proposed, which can roughly be divided into end-to-end and modular approaches.

End-to-end approaches like Vision Language Models (VLMs), which are pre-trained on large knowledge bases and either applied in a zero-shot fashion or finetuned to the specific dataset, have shown promising results. However, they lack explainability and are limited in the complexity of reasoning they can perform.

Modular approaches split the task into several subtasks and solve each of them individually. These modules can be neural networks, but may also take the shape of traditional methods like regular expressions for question parsing and logic programming for reasoning. This paradigm comes with the added benefit of adaptability, since the modules can be exchanged or extended to fit the task at hand. A pipeline split into a visual, language and reasoning module can thus benefit from the advances in each of these fields, without major restructuring or expensive retraining. A potential downside is the need for a predefined interface between the modules. [1]

One common representation for the output of the visual module is to pass detected objects, possibly including relations between them, as a list of bounding boxes with corresponding attributes to the other components. Depending on the reasoning engine, the information then has to be transformed into a logical representation.

This transformation step is the focus of this thesis, which will be explained further in the following section. While the scene information can be encoded algorithmically with relative ease, the question provides a challenge, since it takes the form of natural language and thus may contain linguistic complexities such as ambiguities, descriptions rather than direct references, etc., which may need background knowledge, context, theory of mind or guesswork to resolve. A potential solution is to introduce LLMs as an interpreter to bridge the gap between natural language and logic programming, which this thesis will explore guided by the following research questions.

1.2 Research Questions

Neuro-symbolic approaches to VQA combine the strengths of neural networks and symbolic reasoning. However, they also inherit the need for a structured knowledge representation from the latter, which, in practice, is rarely available. With the advent of

[1]The naïve approach of a shared embedding space is often not compatible with symbolic components and requires retraining each time a component is exchanged, negating a core advantage of modularity.

LLMs, which are capable of generating text, while following instructions, an automatic translation suddenly becomes feasible. The main research question of this thesis is therefore:

(RQ) Are LLMs capable of translating natural language questions into logical representations reliably? Investigating this, further subquestions arise such as:

(ARQ 1) What influence do prompt parameters, such as the number of in-context examples, have on translation quality? The discussed method of generating prompts has several degrees of freedom, which influence translation quality and computational cost. Unlike the parameters that research questions 2 and 3 deal with, the parameters related to the prompt generation heavily interact and thus cannot be studied independently. These parameters are the prompt itself, the number of in-context examples and the sampling strategy thereof. In this thesis, a general strategy for prompt generation is developed and probed to examine cross-parameter interactions and to find an optimal configuration. Details are given in Section 3.1 and the results are discussed in Section 5.1.

(ARQ 2) What representation is best suited for the translation? The exact output format is irrelevant to the rest of the VQA pipeline, as long as it can be unambiguously transformed into the representation consumed by the symbolic solver of choice (e.g., Answer Set Programming (ASP)). The representations compared in this thesis are a linear and nested version of ASP, the GQA native functional representation and a code-like representation. All representations are explained in detail in Section 3.3 and evaluated in Section 5.2.

(ARQ 3) Are grand scale LLMs necessary or can smaller, on-device models be used? Under the assumption that LLMs prove capable of performing the translation, the next question is whether they need to be as large as the current state-of-the-art models. While bigger models are generally better when it comes to output quality, they come with a higher computational cost and possibly dependence on the actors providing the models. This can entail concerns regarding privacy or limits to inference volume, depending on the application. In Section 5.3 a selection of models (listed in Table 1) is evaluated to examine this trade-off.

1.3 Contributions

With the research questions outlined, the contributions of this thesis can be summarized as follows:

Feasibility of LLM translation for Logic Programming A systematic investigation of different factors influencing translation quality, such as models, prompts and representations (Chapter 5).

Evaluation A framework for more accurate evaluation of VQA systems (Section 3.4).

GQA Semantic Interpreter An algorithmic way to evaluate the semantic question representation of the GQA dataset (Section 4.1), along with classification of failure cases for the implementation and the dataset (Section 6.1)

Tooling Utilities for translating between representations (Section 4.3) and a self-contained concept extractor (Section 4.2).

To aid in understanding these contributions as they are presented, the following chapter provides context and links to related work.

2 Background & Related Work

This chapter provides background knowledge necessary for understanding the rest of this thesis. It is not meant to be a comprehensive overview of the topics touched, but selectively covers the aspects directly relevant to the methods of this thesis. The chapter contains a section about VQA, LLMs, ASP and WordNet, as well as an overview of relevant datasets and related work in the field.

2.1 Visual Question Answering

The task of Visual Question Answering (VQA) lies at the intersection of the fields of Computer Vision (CV) and Natural Language Processing (NLP) and presents the challenge of answering natural language questions about an image. It was introduced by Agrawal et al. (2016) with an accompanying dataset, introduced in Section 2.5. VQA builds upon the well-studied problem of text-based question answering and transforms it into a multimodal task. Its applications range from aiding visually impaired people to image retrieval and content moderation. The profoundness of the task led to it even being deemed a holy grail of AI (Agrawal et al. 2016).

Compared to similar tasks, such as image captioning, question answering not only provides a much more challenging environment, but is also easier to evaluate. VQA has been a popular research topic in recent years and many approaches have been proposed, a selection of which is presented in Section 2.6.

Question Answering (QA) provides a good testbed for evaluating the interplay of LLM translation and symbolic reasoning, as it introduces ambiguity through natural language, which thwarts simple pattern matching[2]. Additionally, data availability and active research, which allows for comparison to other approaches, make it a solid choice.

2.2 Large Language Models

In addition to solving concrete tasks, a goal of AI research has been to develop general models, capable of handling problems without requiring task-specific training. These so-called foundation models display promising results in a number of domains, but come at the cost of requiring large amounts of data and compute to tune the possibly billions of parameters. This is particularly true for language processing models, hence the name LLM (Bommasani et al. 2022).

In the context of this thesis, such models offer a way to translate between natural language and logical representations, without relying on human effort or brittle rule-based systems.

Typically, foundation models are pre-trained on a downstream-task agnostic objective, which for NLP is often next-sentence-prediction or masked language modeling, where models are asked to predict a token based either on just the preceding- or surrounding text respectively. This paradigm of learning on one general task and applying the model to another, with or without fine-tuning, is also called transfer learning (Bommasani et al. 2022). A range of pre-training objectives have shown success. It is also possible to combine several such tasks in a pre-training routine.

Tokens can range from a single letter or symbol to short words and constitute the units LLMs process. In practice, models are fed a sequence of tokens and output a probability

[2]though it may be argued that with synthetic question generation, as in GQA, this is nullified

distribution over the token space. A common post-processing step is to then sample from this distribution[3] to obtain a single token. At inference, the model is typically presented an initial piece of text, known as a prompt. It is then auto-recursively applied to its output until an End of Sequence (EoS) token is predicted or the maximum length is exceeded (Y. Liu et al. 2024).

This procedure on its own yields a model that simply aims to continue the prompt in a stylistically identical manner, which is not always the desired pattern. Most training regimes are therefore augmented with additional objectives to guide model behavior; for example, to encourage following instructions or adhering to ethical or legal guardrails (Ouyang et al. 2022).

This thesis examines both instruction-based models and pure completion models, the distinction between which becomes relevant in 3.1.

As opposed to zero-shot application, where foundation models are presented with tasks differing from the pre-training objective without guidance, these models empirically perform better in a few shot[4] setting, where the prompt includes examples of the desired output (Brown et al. 2020).

However, LLMs typically have a limited context window which restricts the number of tokens they can process at a time. This means in-context examples have to be carefully selected to maximize the amount of information conveyed to the model, while minimizing computational cost and staying within the context window.

This limitation stems from the attention mechanism (Vaswani et al. 2023), which scales quadratically in terms of memory and compute. While historically other sequence modeling architectures have seen use, transformers, which are models based on attention, have become the de facto standard in language modeling(Y. Liu et al. 2024).

While countless LLMs have emerged, this thesis focuses on a select few, which are listed in Table 1.

Model	#Params	Contextsize	Origin
Orca3B	3B	1024	Microsoft
Zephyr7B	7B	8192	Hugging Face
GPT-3.5 turbo	20B	4,096	OpenAI
GPT-4 turbo	?	128K	OpenAI

Table 1: Selection of LLMs.

The models are chosen to represent a range of sizes. The Orca3B and Zephyr7B are open-source and locally hosted models, while GPT-3.5 and GPT-4 are proprietary models accessed through an API. Orca3B is a small model trained through knowledge distillation as described by Mukherjee et al. (2023), with GPT-3.5 as a teacher. Zephyr7B is a fine-tuned version of Mistral7B, a model trained on synthetic Datasets by Mistral.AI (Jiang et al. 2023). Zephyr7B is trained with less alignment to be a helpful assistant and performs competitively with larger open-source LLMs on a range of benchmarks (Tunstall et al. 2023).

[3]usually biased depending on a temperature parameter

[4]also known as in-context learning

2.3 Answer Set Programming

The section discusses the reasoning paradigm of choice for this thesis, Answer Set Programming (ASP) (Brewka et al. 2011), which is an approach to declarative problem solving that comprises an expressive modelling language and efficient solvers. It lends itself to this task, as it is expressive enough to model the problem, while staying performant and has already proven to be a good fit for question answering in the literature (Basu et al. 2020; Eiter et al. 2022; Hadl 2023).

ASP is a declarative programming paradigm where programs are specified as an unordered set of rules of the form

$$a \text{ :- } b_1, \dots, b_m, \text{ not } c_1, \dots, \text{ not } c_n \tag{1}$$

where a, b_i and c_i are Boolean elements called atoms. Atoms and their negated counterparts are also called literals. ASP is declarative in the sense that these rules do not specify how to compute a solution, but rather what a solution should look like. Intuitively a rule r states that if its body $body(r)$ is satisfied, that is all b_i's are true and all c_i's are false, then its head $head(r) = [a]$ is implied and thus true as well. The body of a rule may be further divided into a positive part $body^+(r)$ which contains all b_i's and a negative part denoted by $body^-(r)$ which contains all c_i's. If both $body^+(r)$ and $body^-(r)$ are empty, then r is called a fact and a is trivially true. If $head(r)$ is empty, then r is called a constraint and $body(r)$ must not be satisfied for the program to have a solution. A solution, also called answer set, is an interpretation, i.e., a subset of all atoms, which satisfies every rule in the program, when considering the rules under the stable model semantics, as defined by Gelfond and Lifschitz (2000).

It is important to note that ASP distinguishes between weak and strong negation. Weak negation is denoted by the keyword *not* and simply states that there is no evidence for the negated atom. Strong negation is denoted by $\neg$ and states that the negated atom is explicitly false (Gelfond and Lifschitz 1991).

While this basic syntax is powerful enough to express Boolean logic, it becomes even more expressive through the use of predicates and further extensions. Predicates model relations between objects and are defined by a name and an arity, that is, the number of arguments a predicate takes. A rule with predicates has the form

$$A \text{ :- } B_1, \dots, B_m, \text{ not } C_1, \dots, \text{ not } C_n \tag{2}$$

where A, B_i and C_i are atomic formulas and thus take the form $R(s_1, \dots, s_k)$ where R is a predicate of arity k and s_i are either variables or constants, which are denoted by upper- and lowercase letters, respectively (Brewka et al. 2011).

The introduction of variables complicates the solving process, as the search space increases exponentially with the number of variables and their possible instantiations (the Herbrand universe) (Brewka et al. 2011). This space of all possible combinations of variable instantiations is called the grounding of a program. The grounding of a program is a prerequisite for the solving process, as it is not possible to reason about variables. To address this, a variety of optimization techniques have been developed, which attempt to find simpler programs with the same answer sets or constrain the search space in other ways (Brewka et al. 2011).

The expressivity of ASP can be further improved by the use of cardinality atoms. These terms declare that a certain number of atoms in the specified set must be true and take the form

$$l\{a_1, \dots, a_n\}k \tag{3}$$

where l and k are integers that specify the lower and upper bound of the number of atoms which must be true or, if the cardinality expression is the head of a rule, the number of atoms to be included in the answer set. In the special case where the expression is unconstrained, meaning $l = 0$ and $k = n$, the cardinality atom is called a choice atom and l and k are omitted (Calimeri et al. 2020).

Another powerful extension of ASP are optimization formulations. In contrast to regular (also called integrity) constraints, which must be satisfied for a program to have a solution, weak constraints may be violated which incurs a cost. The goal is now to find the optimal answer set, i.e., the one which minimizes the sum of all costs. Weak constraints have the form

$$:\sim\ b_1, \ldots, b_m,\ \text{not}\ c_1, \ldots,\ \text{not}\ c_n.[w, (t_1, \ldots, t_k)] \tag{4}$$

where $(t_1, \ldots, t_k)$ make up the term tuple contributed to the cost function and w is an associated weight (Calimeri et al. 2020).

In this thesis, ASP serves as a reasoning engine, which takes scene information and questions in symbolic form and outputs an answer. This procedure is described in detail in the Implementation chapter of Hadl (2023).

2.4 WordNet

Since treating words as disconnected entities misses out on the semantic relationships often necessary for reasoning, this thesis uses WordNet to enrich the object classes with semantic information.

WordNet (Miller 1995) is a manually curated lexical database of words, their meanings and relations between those meanings. It can be thought of as a dictionary optimized for machine use.

WordNet consists of nouns, verbs, adjectives and adverbs linked to so-called synsets, which are sets of synonyms with an associated definition and other information. A specific meaning in a synset is addressed by $\langle$*lemma*$\rangle$.$\langle$*pos*$\rangle$.$\langle$*number*$\rangle$, where *lemma* refers to the word the synset belongs to, *pos* specifies the part-of-speech tag and *number* roughly represents how common this specific meaning is within the word class. Thus, *fall.n.02* maps to the second most common definition of "fall" when used as a noun: "a sudden drop from an upright position", with the most common (*fall.n.01*) being "the season when the leaves fall from the trees".

Synsets keep pointers for the following semantic relations (Miller 1995):

- **Synonymy** Words that have identical meanings. *happy* $\leftrightarrow$ *glad*
- **Antonymy** Words with opposing meanings. *happy* $\leftrightarrow$ *sad*
- **Hypo- & Hypernymy** Nouns with sub- and superordinate relations. *finger* $\rightarrow$ *body part, object* $\rightarrow$ *body part*
- **Mero- & Holonymy** Nouns being a part of another and the inverse relation. *fingertip* $\rightarrow$ *finger, hand* $\rightarrow$ *finger*
- **Troponymy** Verbs which are more specific versions of another verb, similar to hyponymy in nouns. *talk* $\rightarrow$ *yell*
- **Entailment** Verbs which imply another verb. *eat* $\rightarrow$ *chew*

These relations allow for better comparison between synsets as opposed to simple string matching. The directed relations allow for the construction of a hierarchical tree of meanings, which can be used to process ancestry beyond immediate parents.

In the context of this thesis, WordNet is used to properly compare words and allow valid interpretations for answers, rather than simply string matching. The exact procedure is described in 3.

2.5 Datasets

This thesis deals with the broad task of VQA, which has many facets of difficulty, each of which can be challenging to evaluate in isolation for a given pipeline, let alone in combination. A range of datasets have been proposed to address this, varying in size, difficulty, domain and ideas to probe VQA models.

The following section introduces common datasets and highlights their specific strengths.

VQA The VQA dataset was collected by Agrawal et al. (2016) as an initial benchmark for the novel task of VQA. It combines roughly 250K images from COCO (Lin et al. 2015) with 760K free-form open-ended questions, leading to a total of around 10M accepted answers. The dataset requires some degree of common sense knowledge and importantly targets various image regions, not just the most prominent objects, which sets the task of VQA further apart from the related task of image captioning.

CLEVR **C**ompositional **L**anguage and **E**lementary **V**isual **R**easoning (Johnson et al. 2016) is a dataset consisting of rendered images of synthetic 3D scenes. It is intended as a diagnostic tool for visual reasoning, testing a variety of primitive reasoning skills, such as counting, comparing, spatial- and logical operations in an unambiguous setting. Thus, the visual challenge remains intentionally limited and even a simple model can achieve near-perfect accuracy. The dataset provides 100K images with 10 questions each, sampled to keep a balanced distribution of question types and answers. Figure 1 gives an overview of the content of the dataset in terms of object attributes, relationships and question types, as well as the question elements and structure. The question-generation process partially inspired the GQA dataset.

Visual Genome The Visual Genome dataset (Krishna et al. 2016) was created as a step towards rich image annotation, a necessity for training, and evaluating scene understanding models. The dataset includes 108K images, each linked to a list of around 50 human-generated region descriptions. These region descriptions are natural language representations of the content in the corresponding image region, from which the mentioned objects, attributes and relationships are extracted and assigned a WordNet synset ID for standardization purposes. The resulting region graphs are then joined into a large scene graph. On average, each image contains 35 objects, 26 attributes and 21 relationships classified into one of around 34K, 68K and 42K categories, respectively. Additionally, Visual Genome provides over 1.7 million questions, each linked to the region, where the answer is located.

GQA The GQA dataset (Hudson and Manning 2019) is an adaptation of the Visual Genome dataset, aiming to provide more varied questions with a balanced answer

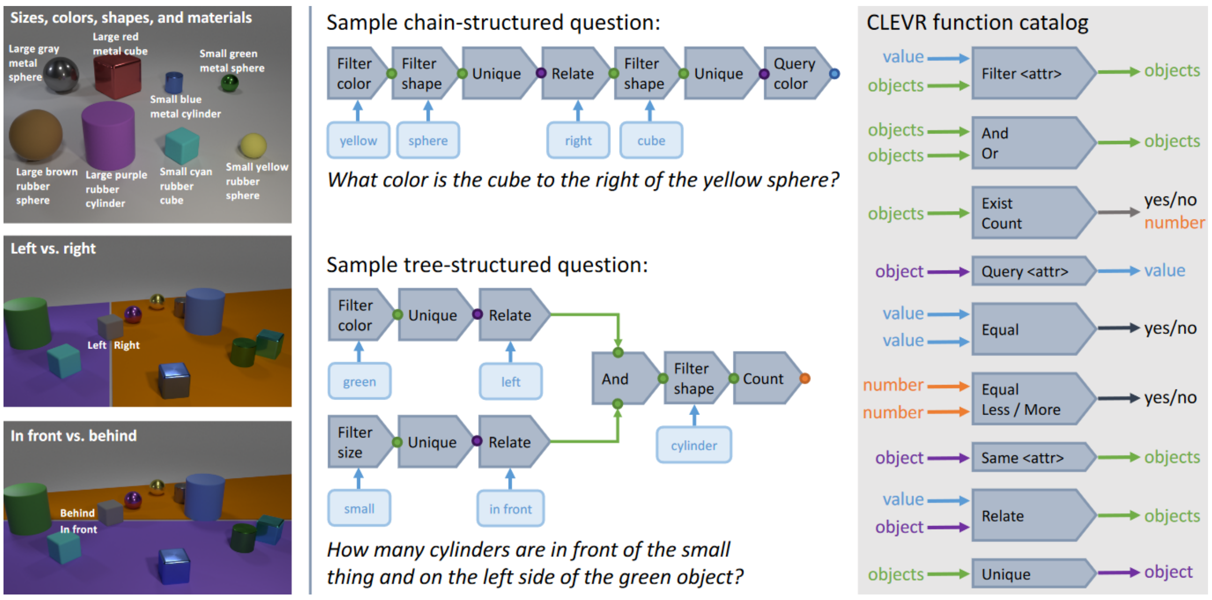

Figure 1: Overview of the CLEVR dataset question structure. (Johnson et al. 2016, Figure 2)

distribution as a benchmark specifically for visual question answering. It includes 22M questions over 113K images. Whereas questions in Visual Genome were human-generated, GQA draws inspiration from CLEVR and proposes a question generation engine that combines question templates with the scene graph to generate question-answer pairs. The question templates are compositional in nature to increase complexity. For example, the template "is there a <object>" could be extended by replacing <object> with <attribute> <object> or <object> <relation> <object>, which would be instantiated to "is there a white chair?" or "is there a chair left of the table?" instead of simply "is there a chair?". Each question originally has the form of a logic program specifying the necessary operations to arrive at the answer, which is translated to natural language and rephrased to increase diversity. Answers are given as a short and full version, where short includes just the answer and full gives a full sentence which can include additional clarification. For the question "is there a frisbee and a fence?" the short answer might be "no", whereas full could look like "No, there is a frisbee but no fences".

Additionally, GQA proposes five evaluation metrics beyond accuracy, which simply measures the fraction of correctly answered questions.

- **Consistency** Questions are linked by entailment, since a model that asserts that a given object has a specific attribute should ideally also agree that, for example, said object is even present in the image. Consistency is computed by measuring the accuracy over all entailed questions for each question and averaging.
- **Validity** This metric measures whether a potentially valid answer was given. Valid is defined as part of the answer scope for a given question, meaning colors for questions like "what color ..." or yes/no for questions like "is there...".
- **Plausibility** Plausibility is only defined for questions about attributes and is measured by checking if the predicted attribute occurs for the object in question at least once in the dataset.

- **Grounding** As each question includes a pointer to an image region, if a given model provides a scoring of regions relevant to its answer (e.g., an attention map in a transformer). these regions can be compared to validate that the model derives its answer by paying attention to the correct location. Concretely, grounding is computed as the fraction of attention inside the bounding box containing the answer.
- **Distribution** This final metric compares the actual versus predicted answer distribution by computing the χ^2 statistic for each question type.

While these metrics are a good starting point, they require dataset statistics to be computed and are thus not directly applicable to other datasets. Additionally metrics like grounding are only applicable to models that provide attention maps or similar. This thesis therefore proposes a more general evaluation framework.

OK-VQA **O**utside **K**nowledge-**V**isual **Q**uestion **A**nswering (Marino et al. 2019) is a dataset that aims to test the ability of VQA models to combine visual clues with external knowledge sources. It provides 14K questions linked to images sampled from COCO (Lin et al. 2015). Each question is assigned one of several categories of knowledge required to solve it, as seen in Figure 2. The knowledge is not provided, but since both questions and answers are human-authored, it can be assumed that it would be common sense to a human.

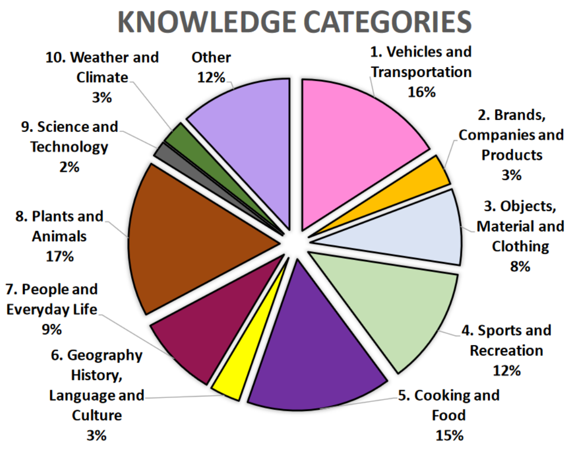

Figure 2: Distribution of knowledge categories. (Marino et al. 2019, Figure 3)

NExT-QA **Ne**xt generation of VQA models to **Ex**plain **T**emporal actions (Xiao et al. 2021) is a benchmark for VQA models, which targets video understanding and temporal reasoning. NExT-QA provides over 5.4K videos and 52K questions covering both multiple choice and open-ended questions, whereof roughly half require causal reasoning, 29% temporal reasoning, while the rest are purely descriptive.

Dataset of Choice While the methods proposed in this thesis are intended to be general, evaluation is performed exclusively on the GQA dataset, for several reasons. GQA is large and diverse, providing a good benchmark for a variety of reasoning types, while being transparent enough to allow for a detailed analysis of model failures with the provided scene graph, metrics and semantic question representation. It is more difficult than CLEVR, which has been investigated by Eiter et al. (2022) and struggles to bring neuro-symbolic methods to their limits. It is self-contained, unlike OK-VQA. While external knowledge retrieval is an interesting direction for future work, it is beyond the scope of this thesis. While video processing is analogous to linked still frame processing in theory, even a partial temporal scene graph would drastically increase program complexity and thus computation time, making NExT-QA unsuitable. Finally, since GS-VQA evaluated their pipeline on GQA, parts of the implementation can conveniently be used as a basis for this work, while providing a baseline to compare against.

2.6 Related Work

The following section aims to give an overview of work relevant to the problem of VQA and automatic translation to ASP. As this field of research is vast and volatile, this section is not intended to be a full survey of the research for the areas touched on in this thesis, but rather focuses on aspects of works relevant to the understanding of the discussed methods. For a more comprehensive overview, the reader is referred to dedicated surveys such as Faria et al. (2023).

2.6.1 Standalone Models

This section introduces a selection of models that are commonly used as components in VQA pipelines. With the exception of BERT, none of these are relevant to the translation itself, but they provide context for related work and the evaluation strategy.

YOLO **Y**ou **O**nly **L**ook **O**nce (Redmon et al. 2016) is an object detection framework that has been incrementally improved since its initial publication and provides a flexible baseline for a variety of object detection tasks. Compared to previous multi-stage object detectors, which separated region proposal and classification, YOLO outputs bounding boxes and class probabilities in a combined prediction step, which saves on computation time without loss of accuracy. YOLO also predicts at different scales to improve performance across varying object sizes.

CLIP **C**ontrastive **L**anguage–**I**mage **P**retraining (Radford et al. 2021) is a multimodal image classifier developed by OpenAI. It is trained on a large corpus of image-text pairs to learn a joint embedding space for images and text, which allows for zero-shot classification. This means that the model can detect previously unseen concepts in images, if they are mentioned in the text input. CLIPs output for a given image is a vector of probabilities along the classes in the text input.

BLIP **B**ootstrapping **L**anguage–**I**mage **P**re-training (J. Li et al. 2022) is a VLM pre-training routine developed by Salesforce. The authors propose an adversarial training scheme with an image captioning and filtering component to improve data quality compared to images from the web paired with their alt-text. The proposed architecture trained on the resulting dataset performs better than comparable models trained on image alt-text pairs at a range of tasks like image captioning, image-text retrieval and VQA.

OWL-ViT **O**pen-**W**orld **L**ocalization **V**ision **T**ransformer (Minderer et al. 2022) is a transformer-based object detection model. It builds on a CLIP backbone and uses slight architecture modifications to output bounding boxes instead of similarity between image and text. This enables open vocabulary object detection, a crucial capability for zero-shot VQA.

BERT **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (Devlin et al. 2019) is an encoder-only language model developed by Google. It is pre-trained on masked language modeling and next-sentence prediction tasks on the Toronto BookCorpus (*bookcorpus · Datasets at Hugging Face* 2023) and Wikipedia (*Wikipedia* 2024). BERT can be fine-tuned for a variety of NLP tasks or its embeddings can be used directly for downstream tasks. These embeddings have a dimensionality of 768, much less than the vocabulary size of 30K, which means that they compress meaning into a dense vector space. In the context of this thesis, the cosine distance between two BERT embeddings is used as a measure of semantic similarity.

Building upon these or similar models, the next sections briefly introduce two paradigms for VQA.

2.6.2 End-to-End VQA

A natural approach to VQA is to tackle the problem as a whole, without enforcing intermediate structure or explainability. This is the most successful approach to date and models like BLIP (J. Li et al. 2022), GPT-4 Vision (OpenAI et al. 2023; Y. Li et al. 2023) and PaLI (Chen et al. 2023) perform well on this task without adaptation. However, they lack interpretability and can behave unpredictably, which is a major drawback for real-world applications. It is therefore not advisable to abandon modular approaches entirely, even if they lag behind in performance as of the time of writing.

2.6.3 Modular VQA Approaches

The following works break the problem down into manageable subproblems, commonly a visual and language component, sometimes with a separate reasoning engine. While not the only possible representation, the prominent choice for the interface between the visual and language component is a scene graph, i.e., a graph whose nodes are objects in the images and whose edges are relations between them. To deal with uncertainty in the visual component, the scene graph may be enriched with confidence values to form a probabilistic graph.

AQuA **A**SP-based **Qu**estion **A**nswering (Basu et al. 2020) propose a pipeline that integrates YOLOv3 for object detection with an ASP solver. Translation of question and scene information to a logic program is done via traditional NLP methods, namely semantic relation extraction, and enriched with manually defined commonsense facts and rules. They evaluate on CLEVR and demonstrate better-than-human performance at 93.7% accuracy.

Eiter et al. (2022) tackle the CLEVR dataset with a neuro-symbolic pipeline. They use fine-tuned YOLOv3 for object detection and translate the semantic question representation into ASP rules algorithmically. Unlike similar work (Yang et al. 2023b; Manhaeve et al. 2018), they threshold the confidence values of the object detection based on statistical measures to prevent runtime explosion from too many objects.

GS-VQA Hadl (2023) is of particular importance to this thesis, as part of the evaluation is done building on this work. Thus, the relevant aspects of this work are explained in more detail throughout the thesis, specifically in Sections 3.2 and 4.4.

The author combines the VLMs CLIP and OWL-ViT with an ASP solver to answer questions from the GQA dataset. However, like Eiter et al. (2022) the pipeline inputs are image and GQA semantic question representation, not natural language, which is where the results of this thesis can come in to close the gap.

A core idea of this work is to only partially construct the scene graph to save on program complexity and thus computation time.

In practice, this means processing the question to extract the relevant objects, attributes and relations. The set of object classes is then passed to OWL-ViT for object detection, whose output is enriched by CLIP to include attribute and relation information. This partial scene graph, also called scene encoding, is combined with a predefined set of rules and facts, referred to as the theory, and the question encoding, into a standalone logic program. The answer set of this program corresponds to the answer to the original question. To deal with the inherent uncertainty of the perception component, the confidence values are included in the encoding and multiple predictions for each object are included. This way a lower-confidence but sensible interpretation can be considered, instead of being discarded in favor of a high-confidence one.

ViperGPT & CodeVQA Surís et al. (2023) and Subramanian et al. (2023) both use code generation as a reasoning engine. Their approach presents an LLM with the task of generating a program to solve the specific question using a predefined API. The program is then executed and the output is used as the answer. This approach is flexible, since the API can be extended to include new capabilities, such as depth estimation, color retrieval or segmentation. As a fallback, the VLM ordinarily used to answer simple subquestions is called on the entire image, should the program fail to return an answer or execute at all. CodeVQA improves on the results of ViperGPT by adding a similarity-based in-context example retrieval.

Drawing from the presented works, this thesis builds on the modular pipeline of Hadl (2023) and extends it to handle natural language input. It takes inspiration for LLM-based translation from ViperGPT and CodeVQA, but uses ASP instead of interpreted code as the reasoning engine, like Basu et al. (2020) and Eiter et al. (2022). This intersection of ASP and LLMs translation is not novel and has been explored in other contexts, which is the subject of the next section.

2.6.4 LLM translation to ASP

The following works explore the use of LLMs to translate text into ASP programs for various tasks. Crucially, none of them evaluate on this capability in the context of VQA.

STAR **S**emantic-parsing **T**ransformer and **A**SP **R**easoner (Rajasekharan et al. 2023) propose a reasoning pipeline for generic Natural Language Understanding (NLU) tasks involving LLMs. They show significant performance improvements on qualitative- and mathematical reasoning, which declines with model size. Larger models seem to benefit little or not at all. Their pipeline also includes an external commonsense knowledge base, which potentially limits the applications.

Yang et al. (2023a) & Ishay et al. (2023) use LLMs to translate natural language instructions into ASP programs. They evaluate on several NLP benchmarks, targeting puzzle-solving and planning tasks and demonstrate strong performance[5] on most of them. Their pipeline uses knowledge modules, that can be reused for other tasks.

These works demonstrate the feasibility of using LLMs to translate natural language to ASP programs for a variety of small and well-defined tasks or puzzles. This thesis aims to extend this capability to the more complex task of VQA and evaluates the performance loss from integrating an LLM translation component into an established pipeline.

[5]In fact, on some benchmarks the most frequent error source is faults in the dataset, which leads the authors to propose their method as a data validator.

3 Methodology

This chapter describes the methodology chosen in answering whether LLMs can be used to translate natural language questions into logical representations. The methods discussed are applicable to any domain, as long as a suitable solver and in-context examples are available. As discussed in 2.1, the VQA task provides a good testbed for this evaluation, due to the rich availability of data paired with definitively answerable questions. For the purposes of demonstration and quantitative evaluation, this thesis implements integrating a solver for the GQA dataset and examines how it compares as part of a full VQA pipeline.

Figure 3 provides an overview of the pipeline. The following sections explain the Prompt Creator, Solver and Evaluator in detail. The roles of the dataset and LLM components are apparent and adequately discussed in Sections 2.5 and 2.2 respectively.

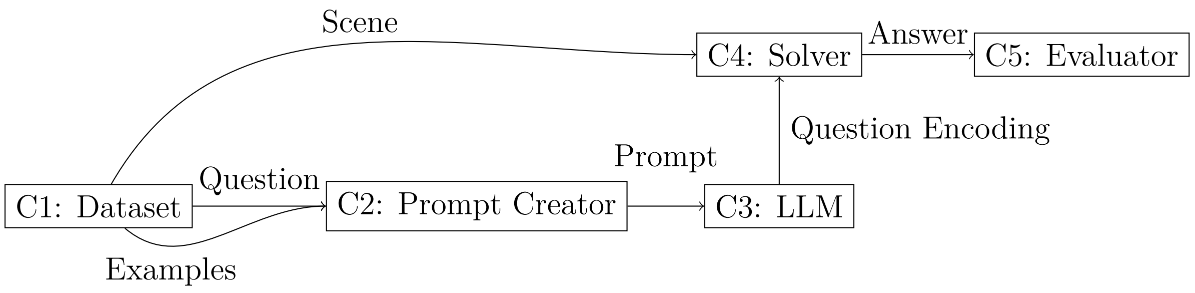

Figure 3: Overview of a question answering pipeline. Components are assigned numbers for reference.

Throughout the pipeline, there are several degrees of freedom, through which the reliability of the translation process can be tweaked. These are the choice of model, prompt, target representation, whether to include examples covering all operations, the number of in-context examples and the sampling strategy thereof. Aside from target representation, which Section 3.3 covers, these are explained in the respective component they appear in.

3.1 Prompt Construction (2C)

Shaping LLM output, a process known as prompt engineering, is a delicate and largely intransparent task (*OpenAI Platform* 2024). Rough guidelines for prompt construction have been established, but the process is still largely trial and error and there are countless, often model-specific, "best practices" available. This thesis does not claim to have found an optimal prompt, since the main focus of this thesis is establishing feasibility. Therefore, mild performance improvements would be expected with more investment in this direction. Additionally, attempts have been made to keep the prompt minimal, to limit API costs and computation time. The full prompt templates can be found in Appendix 7.1.

The prompting strategy used differs between two classes of models: conversation-based and completion models: Conversation-based models like Zephyr (Tunstall et al. 2023) are given a preprompt by a designated *system* role, which is followed by in-context examples and the question to be translated by the *user* role.

3.1.1 System Prompt

The preprompt describes the persona the model should take on (a question translator) and general instructions, optionally including the available operations. Completion-based models are not assigned a role, instead, the prompt is structured as if it were simply a list of examples with the last one missing the translation. This often leads to the model inventing new questions and translations after answering the intended one if the token limit is not reached, however, this is easily filtered with the following post-processing step. For the flat asp representation, the output is cut off after the *end()* predicate, for all other representations the number of open parenthesis is counted until the first one is closed, at which point the output is cut off. Additionally, if the output contains three backticks, the first line is filtered out. "‘‘‘<language>" denotes a code block in markdown, which some models use to mark the section as, for example, ASP. This is correct but undesirable for parsing.

The system prompt is agnostic to the question. An example of a system prompt is given in Listing 5.

You are now a question parser. (5)
Your task is to translate a question into a functional program.
The available operations are:
select, relate, common, verify, choose, filter, query, same, different, and, or, exist.

3.1.2 User Prompt

The prompt itself consists of example questions and their translation, optionally a directive to translate the next question and finally the question to be translated. Unlike the system prompt, the user prompt is a template and filled in with the actual question and examples at runtime. An example of a user prompt template is given in Listing 6.

Here are examples of questions and corresponding programs: (6)
{examples}
Can you translate the following question into an ASP program?
{question}

The filled-in version for the question "Is there a green car?" is given in Listing 7.

Here are examples of questions and corresponding programs: (7)
What is the dish inside of?
query(unique(relate_any(select(scene(), dish), inside, object)), name)
. . .
Can you translate the following question into an ASP program?
Is there a green car?

Given the limited context window, it is important that the provided examples are as representative as possible. Here two partially conflicting goals have to be balanced: On the one hand, examples should be as diverse as possible, to cover as many cases as possible. On the other hand, they should be as similar as possible to the target question, as this knowledge is more likely to be directly transferable. Other works, such as Subramanian et al. (2023), have relied purely on similarity and argue that knowledge about most likely unused operations is not useful enough to adjust the sampling strategy. This thesis takes a more conservative approach and uses a combination of similarity and diversity to sample examples. First, similar examples are gathered, then examples including non-used operations are added until all operations are covered. This is later referred to as the *cover_operators* option and compared to purely similarity based sampling.

Similarity can be measured in a variety of ways. In this thesis, BERT and Jaccard similarity were explored. BERT similarity is defined as the cosine similarity of BERT embeddings averaged over the question Jaccard similarity, on the other, hand measures word overlap between two questions. BERT similarity, while more semantically meaningful, is more expensive to compute and requires a pre-trained model, whereas Jaccard similarity is cheap to compute and can be used without any additional resources. These sampling strategies are compared to a baseline of random sampling.

The set of possible examples is taken from the training set of the GQA dataset, while the questions for the experiments are taken from the validation set. Splitting the data sources alone is not enough to guarantee no overlap, as the same question may be asked about different images, which can make it appear in both sets. This is addressed by filtering the chosen in-context examples for the target question.

Increasing the number of potential in-context examples is a trade-off between potentially more relevant examples and the increased computational cost of computing the similarity measures. For the experiments in this thesis, the first 1000 questions are chosen, as this guarantees a diverse set of examples including every operation, while not noticeably impacting the computation time. The number of examples is a parameter, that could theoretically also be investigated within the scope of ARQ 1, but the influence[6] is expected to be relatively small, so this is left for future work.

3.2 VQA System (C4)

The process of answering a question once the logical representation has been obtained is not the main focus of this thesis. Instead, solving a given scene-question pair for an answer is done purely to evaluate the translation process end-to-end.

A distinction is necessary between a perfect information setting, where the scene graph is given[7], and an imperfect information setting, where the relevant parts of the scene graph have to be derived from an image. In the perfect information setting while translating to the GQA representation, the GQA semantic interpreter explained in Section 4.1 can be invoked. For all other cases, rather than proposing novel ways of tackling the challenge of VQA, the methodology of Hadl (2023) introduced in Section 2.6 is reused in different configurations. GS-VQA supports both the perfect information setting by algorithmically translating the scene graph to an ASP program, and the imperfect information setting by using Machine Learning (ML) models to generate a partial scene graph from the image.

Figure 4 shows the pipeline initially depicted in 3 in more detail, with the solver decomposed into its subcomponents.

[6]at least keeping the parameter within a reasonable range

[7]It turns out this setting is far from perfect in many cases, as is discussed in 6.1.1

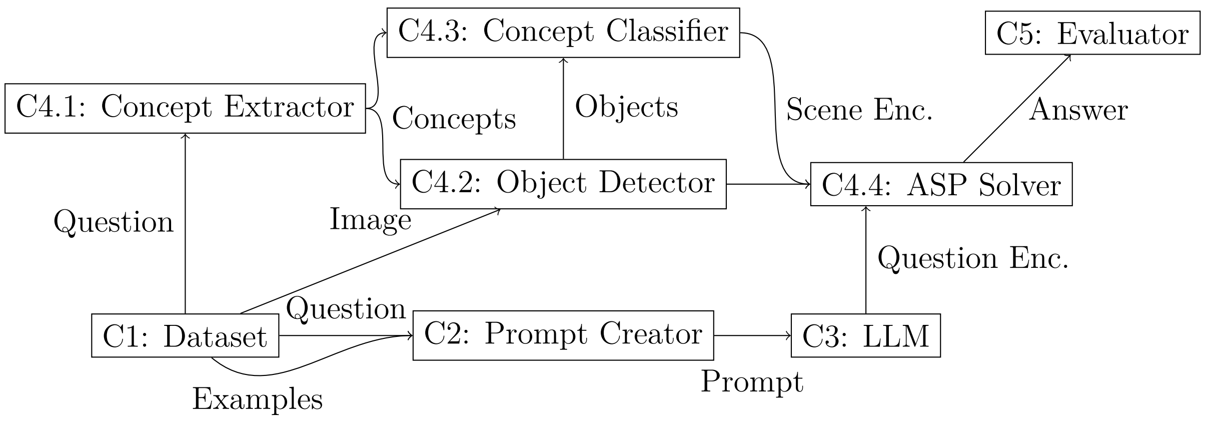

Figure 4: Question translation as part of a VQA pipeline, exemplified with Hadl (2023)'s GS-VQA.

3.2.1 Concept Extraction (C4.1)

This component processes the question to extract all relevant object-, attribute- and relation classes, such that the image can be examined in a targeted way. The concept extractor is a rule-based system, which collects mentions of concepts from the question and cross-references them with a predefined list of classes. If, for example, the concept *device* appears in a question, the concept extractor also marks *laptop*, *camera*, etc. as relevant concepts. The set of relevant concepts is then passed on to the object detector and concept classifier.

3.2.2 Object Detection (C4.2)

The object detector is a VLM, which processes the image to find all relevant objects present in the image. The exact model is a degree of freedom, thanks to the modular design of the pipeline. However, the experiments are performed with OWL-ViT as in Hadl (2023). For each detected object, the maximally specific subclass is determined and passed to the concept classifier and asp solver, alongside the predicted bounding box. The asp solver additionally receives the detection confidence, which is used in the optimization formulation to penalize answers including unlikely interpretations of the image.

3.2.3 Concept Classification (C4.3)

The concept classifier is also a VLM, which queries the image for attributes and relations of the detected objects. Again, the model may vary, but the experiments are performed with CLIP, following Hadl (2023). Since model output is not directly interpretable, a target vs. neutral approach is employed, inspired by Sarri and Rodriguez-Fernandez (2021).

CLIP is prompted not just for the attribute or relation in question, but also for a neutral concept. For attributes, the neutral concept is the object itself, (e.g., *a red ball* vs. *a ball*). The neutral relation is chosen to be *and* (e.g., *a person kicking a ball* vs. *a person and a ball*). The likelihood that the concept applies is then determined by the relative score via a softmax function. Let c_{target} be the target concept, $c_{neutral}$ be the neutral concept and $CLIP(c)$ be the cosine similarity between the concept embedding of

CLIPs text encoder for the concept and the image encoders embedding for the relevant image patch. The likelihood of the target concept is then given by Equation 8.

$$P(c_{target}) = \frac{e^{CLIP(c_{target})}}{e^{CLIP(c_{target})} + e^{CLIP(c_{neutral})}} \tag{8}$$

The resulting attribute and relation likelihoods together with the detected objects constitute the scene encoding, which is passed to the asp solver.

3.2.4 ASP Solving (C4.4)

The ASP solver combines the scene encoding with the question encoding as received by the LLM component to form an ASP program. The program is appended to a base program, known as theory (see Appendix 7.2), which defines the operations to make the program satisfiable. An ASP solver is then invoked to generate answer candidates ranked by a combined score of plausibility and confidence, as discussed in Section 2.3. Finally, the answer candidates are passed to the evaluator to determine, whether the answer can be considered correct.

3.3 Comparing Representations

The second auxiliary research question of this thesis deals with the exploration of different logical representations. Such a representation needs to capture the semantics of the question and the scene and be compatible with a reasoning engine, while ideally remaining human-readable. Given these requirements, the most important factor is reliability in translation.

Given the intransparency of large language models, it is difficult to predict how well a certain representation lends itself to translation, so empirical evaluation is necessary. This section describes the different representations that were explored, while the final evaluation is presented in Section 5.

All representations share roughly the same set of operations, which is based on the GQA semantic representation:

1. **select**: selects all objects of a given type
2. **relate**: given a node, selects nodes, optionally of a specified type itself, that are adjacent via a given edge type.
3. **common**: given two target nodes, returns the category of their common attribute (color, material, etc.).
4. **verify**: given a target node, returns whether it has a given attribute or relation to another node.
5. **choose**: given two options of either attributes or relations, returns the one that is present for the target node.
6. **filter**: given a list of nodes, returns only those for which a condition holds (e.g., that have a certain attribute)
7. **query**: given a node, return a specific piece of information about it (e.g., its color, class, type of relation to a different node, etc.)

8. **same & different**: given two nodes and a target attribute, returns whether they share or do not share that attribute, respectively.
9. **and & or**: given two Boolean values, returns their conjunction or disjunction respectively.
10. **exist**: given a target node, returns whether it exists.

With the possible operations defined, the advantages and disadvantages of the different ways to communicate them to the language model are discussed in the following sections.

3.3.1 GQA Semantic Representation

Questions in GQA are annotated with a semantic representation that captures the structure of the question. It is therefore a natural choice to use this representation as a starting point. Conceptually, each question is a directed graph, where a node represents an operation and each edge represents a dependency, where one operation depends on the output of another.

Practically, they are provided as a list of operations, or alternatively, a consecutive string containing the operations. The latter is not used in this thesis and is not discussed further. Examples of both for the question "Does the truck tire look round and black?" are presented in Listings 9a and 9b, respectively.

$$\begin{array}{l} [\\ \quad \{\mathit{operation}\colon \mathit{select},\ \mathit{dependencies}\colon [],\ \mathit{argument}\colon \mathit{truck}\}, \\ \quad \{\mathit{operation}\colon \mathit{relate},\ \mathit{dependencies}\colon [0],\ \mathit{argument}\colon \mathit{tire},\ \mathit{of},\ s\}, \\ \quad \{\mathit{operation}\colon \mathit{verify\ color},\ \mathit{dependencies}\colon [1],\ \mathit{argument}\colon \mathit{black}\}, \\ \quad \{\mathit{operation}\colon \mathit{verify\ shape},\ \mathit{dependencies}\colon [1],\ \mathit{argument}\colon \mathit{round}\}, \\ \quad \{\mathit{operation}\colon \mathit{and},\ \mathit{dependencies}\colon [2,3],\ \mathit{argument}\colon -\} \\ ] \end{array} \tag{9a}$$

$$\begin{array}{l} \mathit{select}\colon \mathit{truck} \to \mathit{relate} : \mathit{tire},\ \mathit{of},\ s[0] \to \\ \mathit{verify\ color} : \mathit{black}[1] \to \mathit{verify\ shape} : \mathit{round}[1] \to \mathit{and} : [2,3] \end{array} \tag{9b}$$

Figure 5 further illustrates the dependencies.

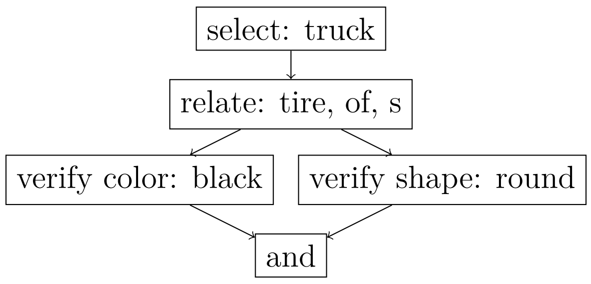

Figure 5: Dependency structure of the semantic representation of the question "Does the truck tire look round and black?".

It is evident that this representation is verbose and very syntax-dependent. This makes it convenient to parse, but complicates the task for a language model, as it adds noise to the input and puts additional constraints on the output format. Not only does less information density increase the token count and therefore computational cost, but it potentially blurs connections between relevant tokens, as they are separated by purely syntactically relevant ones. While the attention mechanism theoretically handles this, it has been shown that position does have a non-negligible impact on information retention (N. F. Liu et al. 2023). Additionally, dependencies are resolved by positional index, which is never explicitly stated for each step and can only be deduced by counting, a weakness of LLMs (Subramanian et al. 2023).

3.3.2 Flat & Nested ASP

While the above representation works reasonably well, requires no processing steps and is readily available, it is specific to the GQA dataset and the evaluation requires a scene graph with perfect information. To address these issues, this thesis takes inspiration from Hadl (2023) and uses ASP directly. The native form here is a list of predicates, similar to the GQA representation, where dependencies are realized through indices. Given the strengths of LLMs which are often trained on large amounts of code, where multi-level dependencies are commonly passed directly as arguments, a nested representation is also evaluated.

Both share the same set of predicates, which is largely similar to the set used by the representation described in Section 3.3.1, but subdivided into base-, intermediate- and terminal operations. The base predicates must be present and encompass the rest of the program.

- **scene**: constitutes a root node and acts as a collection of every object in the scene.
- **end**: marks the end of the program, at which point all remaining variables are considered part of the answer set.

The intermediate predicates are used to construct the program. **select**, **filter**, **relate**, **and** and **or** behave exactly like their GQA counterparts.

- **unique**: collapses the state into a single answer, rather than simply verifying that there is just one. In the case of uncertainty, it chooses the one with the lowest optimization penalty (in the discussed case:inverse detection confidence).
- **compare**: compares two objects with respect to a specific attribute, assuming that either but not both have it. It also consumes a mode, which decides whether the object having or not having the attribute should be returned.
- **negate**: given two states, returns all objects that are present in the first but not the second.

The terminal predicates collapse the state into a single answer, which is then consumed by the end operation. **query**, **verify**, **choose**, **exist** and **common** behave exactly like their GQA counterparts.

- **same & different**: are generalized forms of the GQA operations, which are capable of comparing arbitrarily many nodes.

The earlier example question “Does the truck tire look round and black?” is represented in both flat and nested ASP in Listings 10a and 10b, respectively. Note, that for the nested representation, the end predicate can be omitted.

$$\begin{aligned}
&scene(0).\\
&select(1, 0, truck).\\
&relate(2, 1, tire, of, subject).\\
&unique(3, 2).\\
&verify\_attr(4, 3, color, black).\\
&unique(5, 2).\\
&verify\_attr(6, 5, shape, round).\\
&and(7, 4, 6).\\
&end(7).
\end{aligned} \tag{10a}$$

$$\begin{aligned}
&and(\\
&\quad verify\_attr(unique(relate(select(scene(), truck), tire, of, subject)), color, black),\\
&\quad verify\_attr(unique(relate(select(scene(), truck), tire, of, subject)), shape, round)\\
&\quad)
\end{aligned} \tag{10b}$$

This example demonstrates a weakness of the nested representation. While it is usually equally concise, it only supports direct dependencies and branches in the dependency structure lead to duplicate statements. While this is a negligible difference to the solver, it harms readability and increases the number of tokens necessary to encode the same information. This is addressed by the code-like representation, which is discussed in the next section.

3.3.3 Code-like Representation

Taking the idea of playing to the strengths of language models further, this thesis also explores a code-like representation. It is essentially identical to the nested ASP representation, but allows for the use of variables. This disentangles the potentially convoluted program structure and prevents duplication.

Any given program can be represented with arbitrarily many intermediate variables, however, it makes little sense to use more than one per operation. In fact, the two extreme cases of no variables and one variable per operation are equivalent to the nested and flat ASP representations, respectively. While arguments regarding readability can be made for including more variables, the main concern is the impact on the language model and the resulting token count. Thus, while creating examples for the code-like representation, variables are only introduced on dependency branches, that is, whenever the output of an operation is used by more than one other operation. The resulting representation for the example question is presented in Listing 11.

$$\begin{aligned}
&truck = unique(relate(select(scene(), truck), tire, of, subject))\\
&and(verify\_attr(truck, color, black), verify\_attr(truck, shape, round))
\end{aligned} \tag{11}$$

For readability the variable name in this example has been chosen sensibly, but algorithmically generating names is a non-trivial task and is not addressed in this thesis. Instead, the variables are enumerated as $var1, \ldots, varN$ during preparation of in-context examples. The LLM is free to choose any name for the variables, as long as it is unique and does not conflict with any other variable or operation name and does not contain parentheses.

3.4 Evaluation (C5)

Evaluating the performance of the logical translation systems is a challenging task, one, that in the context of this thesis has merely been approximated. A naïve approach to evaluate the translation alone would be a simple string comparison between a ground truth representation and language model output. This does not yield fair results, as it is trivial to construct logically equivalent statements, which fail this test (e.g., double negation: $\neg(\neg A) \equiv A$). Testing for all instantiations of the variables is not feasible, as even executing the program for a given image or scene graph can be computationally expensive, let alone for the theoretically infinite number of possible inputs. Heuristics, such as comparing the set or frequency of operations used, make some headway, but are still fallible.

The approach chosen for this thesis is to evaluate in an end-to-end fashion, by comparing the solver answer with the expected one. This does not take into account the different ways the system might fail, which have varying degrees of severity and should ideally be treated differently.

Theoretically, this may also artificially boost results, as semantically correct but practically unnecessary predicates may be dropped without penalty. This is illustrated by the image of a single green car. For the question "Is there a green car?" both programs 12a and 12b yield the correct answer "yes", even if the former would fail for other cases, for example, if the car had a different color.

```
scene(0).select(1, car).exists(2,1).end(2)                          (12a)
scene(0).select(1, car).filter(2, 1, green).exists(3,2).end(3)     (12b)
```

Empirically, this has not been observed.

Engineering refined evaluation metrics is mostly left to future work, but some modifications are employed to diminish the warped perception of performance, which are discussed in the following:

Filtering Naturally, a given program can only lead to a correct answer if the context encoding contains the necessary information. As illuminated in Section 6.1.1, this is not always the case for GQA questions. To avoid skewing the results, questions are filtered by whether they can be answered by the given semantic question representation. In some cases, this is immediately obvious, such as with the question "What place is pictured?", as it includes the operation *query: place*[8]. In other cases, it is more subtle, such as when answering the question would traverse an edge not present in the scene graph.

Since the dataset does not provide a way to apply the semantic representation to the scene graph, a secondary contribution of this thesis is a GQA semantic interpreter.

This interpreter either yields an answer or raises one of the exceptions detailed in Section 6.1.1. If the interpreter yields the same answer as the ground truth, the question is

[8]It is not uncommon in the literature to drop these questions outright (Amizadeh et al. 2020)

deemed answerable and included in the evaluation. Usage of the GQA semantic question representation, either through the semantic interpreter or through the perfect information setting of the GS-VQA pipeline, is termed the *Oracle*, since it makes use of information not available to the LLM. Comparing to the Oracle is therefore a measure of how many answerable questions the LLM can answer correctly.

Generous Interpretation While it is not uncommon for datasets in the field of VQA to allow for multiple correct answers (Zou and Xie 2020), GQA only provides a single ground truth answer. This leads to cases, where the model provides an answer, which would be considered correct by a human, but is penalized for not matching the ground truth. The effect is especially large for zero-shot approaches, which are unfamiliar with the dataset vocabulary and thus more likely to produce answers that are semantically correct, but not identical to the ground truth.

To mitigate this, the evaluation makes use of two strategies to accept answers in a more generous fashion. If the answer is not an exact match, WordNet is used to check if the provided answer is either a synonym or hyponym of the ground truth, Allowing hypernymy, i.e., too unspecific answers, indiscriminately is not desirable, as this system could be cheated by always returning WordNets topmost category "entity". Thus, the evaluation engine has a parameter controlling how many levels of hypernymy are allowed. The values used in this thesis are zero, termed the *generous* case, and five, termed the *generous+* case.

In a similar fashion, the evaluation engine also allows the top-k answers to be considered. This impacts comparability, so the results are reported for top-1, top-3 and top-5 answers. Special consideration has to be given to the reliability of the visual component, if more than the top-1 answer is considered for non open-ended questions. For a binary question like "Is there a green car?", a near-perfect visual component might assign a high confidence to the answer "yes", but still assign a non-zero confidence to the answer "no", guaranteeing that the correct answer is among the top-2, invalidating the evaluation. A less reliable VLM might manifest the uncertainty by predicting several possible bounding boxes for the car, thus answering "yes" multiple times, mitigating this effect. The perfect information setting, does not fall victim to this.

Given the general evaluation strategy, experiments can be devised to investigate the individual research questions. The setup chosen for this is detailed in Section 4.5.

Concluding the overview of the pipeline and the evaluation thereof, the following section describes the implementation of the discussed methods and surrounding infrastructure.

4 Implementation

This chapter provides implementation details for the GQA semantic interpreter (4.1), blind concept extractor (4.2), the transformation between representations (4.3), the ASP program (4.4) and the setup for all experiments (4.5).

4.1 GQA Semantic Interpretation

The semantic representation of questions in GQA is a directed acyclic graph composed of a finite set of operations. As discussed in Section 2.5, questions are created by traversing the scene graph, however, a way of reversing this process is not natively part of the dataset. The GQA semantic interpreter tackles this challenge by evaluating each operation successively on the scene graph. Some operations require utility functions, which are explained in Section 4.1.2.

4.1.1 Operations

Each operation takes an argument, dependencies, possibly argument modifiers and a reference to the scene graph. Additionally, some operations have an associated exception, which may be thrown when the operation does not succeed or is called incorrectly. Note that this is not always desirable: *Relate*, for example, should fail when called on its own, but may return nothing when called as part of a *Verify Relation* operation.

A handful of operations require resolving category membership. The interpreter has two modes for this. Either with the WordNet hierarchy or with a curated category mapping compiled by Amizadeh et al. (2020) and expanded by Hadl (2023), specific to the dataset. The former is more general, but error-prone.

The operations are defined as follows:

Select requires an argument specifying the node class to select. It uses neither optional arguments nor dependencies. It simply establishes that the argument is not *scene*, in which case it returns the scene node. Otherwise, it discards the nodeID from the argument to obtain the desired node class and returns a list of all nodes that satisfy *equal_or_hyponym* with said class.

Relate takes a single string specifying target node class, traversed edge class and whether it is an in- or outgoing edge as its argument. It has no optional arguments, but depends on the result of an operation returning a single node. It then loops over all edges adjacent to the dependency node and returns all nodes that satisfy a matching condition. A distinction is made depending on whether the relation is explicit or implicit via a shared attribute, such as *relate: same color*. The first case is resolved with *equal_or_hyponym* between the target node class if the edge class matches the specified edge class and direction, while the second case is resolved by picking the first match of *pick_attr* with the specified category on the target node, filtered by edge direction. Should no nodes satisfy the condition, it raises a *MissingEdgeException* unless suppressed.

Common takes only two dependencies. It finds the set intersection between both lists of attributes and returns the category as determined by *get_category*.

Verify distinguishes between two cases: *Verify Relation* and *Verify Attribute*. If the extra argument is *rel* it calls *Relate* with the same arguments, suppressing the empty exception and returns whether the result is non-empty. Otherwise, it calls *get_attr* on its dependency and returns whether the attribute to verify is present.

Choose takes two options and a category or the keyword "rel" to indicate that the choice is between relations. If "rel" is specified, it calls *Relate* on both options and returns the first non-empty result. Otherwise, it ignores the category and returns the first option that is part of the attributes of the dependency node. This fails for non-trivial cases, such as *choose healthier*, which would require knowledge beyond the scene graph and WordNet.

Filter takes a potentially negated attribute as its argument. It then filters the list of nodes from its dependency by whether they have the specified attribute, or whether they lack it, if the attribute is negated.

Query takes an attribute category as its argument. It first establishes that the argument is not *place*, which cannot be answered from the scene graph alone. Then, it loops over the dependencies and returns all attributes that match the category.

Same takes arbitrarily many dependencies and optionally a category. If a category is specified, it gathers the relevant attributes of all dependencies and returns whether they are equal. Otherwise, it compares node classes and returns whether they are equal.

Different is the inverse of *Same*.

And & Or are conjunctions and disjunctions of their dependencies, respectively.

Exist returns whether its dependency is non-empty.

Besides their use in validating answerability, these operations also serve as more explicit counterparts to the predicates used in the downstream ASP program, if an ASP representation is chosen. Whereas each operation in the semantic parser is imperative in nature, the declarative ASP framework defines them purely in terms of their effect on a state, as discussed in Section 4.4. In both cases, additional steps are necessary behind the scenes to resolve a question, which are discussed in the following section.

4.1.2 Utilities

Visual Genome, which GQA is based on, provides a lookup table that maps object names to their WordNet synsets. These are referenced as *obj_synsets* and *attr_synsets* for object and attribute synsets, respectively, from here on. They are supplemented with a few manually added exceptions, such as *grey.n.01* being replaced with *grey.n.05*. The former refers to the writer and is thus not a hyponym of *color.n.01*, which is necessary for category determination.

equal_or_hyponym takes two node classes and returns whether the former is equal to or more specific than the latter. If either class is a single underscore, the GQA placeholder for any, it returns true. Otherwise, it checks whether both classes are in *obj_synsets* and returns whether the second is part of the first's hyponym tree.

pick_attribute attempts to retrieve the first attribute that belongs to a specified category from a list. It first checks whether the category is in *obj_synsets* and otherwise defaults to the most common synset. It then loops over the list of attributes and returns the first hyponym of the category. Adjectives are nounified, as the hypernym tree is not available for adjectives.

get_category keeps a list of potential categories and returns the first one that applies to the specified attribute.

get_attr retrieves a given attribute from the scene graph structure.

sanitize_answer treats several cases that would lead to a correct answer not being accepted. It maps Boolean results to "yes" and "no", lists of length one to their single element and spatial relations such as "to the left" to "left".

Using these operations and utilities, the GQA semantic interpreter can be used to evaluate the semantic representation of a question. As long as the underlying scene graph is correct, this is a mostly reliable way to determine the answer to a question. Returning to the pipeline itself, the next section provides a way to decouple the GQA semantic representation from visual component through a translation-specific concept extraction.

4.2 Blind Concept Extraction

The pipeline by Hadl (2023) requires the semantic question representation for both the question- and scene encoding. Leveraging LLMs to generate a question encoding only partially frees the pipeline from this dependency, as component 4.1 (compare Figure 4) still makes use of the semantic question representation. To this end, an alternative (also termed *blind* in the sense that it is not omniscient) concept extractor was developed, which leverages the generated question encoding instead. This is not infallible, as it depends on LLM output, which may not fully coincide with the GQA internal vocabulary. In practice, this was not observed to be a significant issue, as it turns out, it may actually be beneficial. Consider the following example: The question "What is the brown door made of?" may be mistranslated to 13a instead of 13b.

$$query(select(brown\ door), material) \tag{13a}$$
$$query(filter(select(door), color, brown), material) \tag{13b}$$

Solving for the first query with the blind concept extractor would likely yield the correct answer, since the VLM is capable of implicitly filtering for brown doors. If, however, the *correct* concepts are used, *select*(*brown door*) has no result, as only doors with the attribute *brown* are part of the scene encoding, not objects with the name *brown door*.

The blind concept extractor works similarly to the original concept extractor, by parsing the logical form of the question and processing each operation according to a set of predefined rules, while discarding known generic concepts, such as *any*, *class* or *name*. In the case of *any* specifically, all known classes are added. Otherwise, the relevant classes, relation types, attribute categories or values are compiled and passed on to the object detector and concept classifier, as in the original pipeline. This yields a method that is fully independent of the semantic representation and can be applied zero-shot to any image-question pair.

4.3 Transformation between Representations

A fundamental requirement for representations is availability. The native GQA representation satisfies this, but all other representations need to be synthesized from the given examples. Hadl (2023) already implements a transformation from GQA to flat-asp. Since the flat-asp representation is much closer to the nested-asp and code-like representations, using flat-asp as a starting point is well advised.

Flat to Nested The first step in the transformation is extracting all predicates with their step number and arguments. This is accomplished with the regular expression `(\w+)\((.*?)\)\.`, which matches every predicate in the program. From said match, the step number is determined as the first argument, while the remaining arguments are classified as predicate dependencies if they are numeric, and regular arguments otherwise. Given a list of predicates, the nested representation is constructed by picking the last predicate and recursively replacing the dependencies with their respective predicates. If a dependency is not in the list of predicates, it is replaced by the root operation *scene*. This transformation omits the terminal predicate *end*.

Nested to Flat Reversing this process is not as straightforward, as the program is not guaranteed to be a tree. During the transformation, convergent branches are duplicated, which leads to repetition of predicates when translating back to flat-asp. This is only the case in 76 of the first 1,000 questions, and can safely be ignored, since the resulting program is logically equivalent to the original. The transformation is implemented by iteratively building up the flat program, while keeping track of the step numbers. Each step begins with finding the first leaf predicate, that is to say, whose dependencies are already resolved to constants or references to other predicates. This is easily determined as the predicate belonging to the first closing parenthesis. The predicate is then replaced by its step number in the nested program, and added to the flat program, together with its arguments. Finally, the terminal predicate *end* is added back into the flat program.

Flat to Code-like As discussed in Section 3.3, there are infinitely many code-like representations for a given nested program. However, clearly the most advantageous place for the introduction of a variable is at a point of convergence, to avoid the mentioned repetition. The implemented transformation thus only makes use of variables in this case, but other strategies are conceivable. This conservative approach only differs from the nested representation in 51 of the first 1,000 questions, so it is not surprising that the involved transformation is largely similar to the one from nested to flat. In fact, the only difference is that instead of transforming the entire program at once, it is divided into blocks. Wherever the output of one operation is used more than once, a variable is introduced. The flat program is transformed up to the point of the first convergence, the output of which is assigned to the variable and the block is replaced by the variable name for the rest of the transformation. Once no more variables are necessary, the transformation continues as in the nested to flat case.

Code-like to Flat The transformation from code-like to flat is performed through the intermediate of the nested representation. All occurrences of a variable are replaced by the value of said variable, and the resulting nested program is then transformed to flat as described above.

4.4 ASP Program

The ASP program is composed of three parts, the theory, the scene encoding and the question encoding, following the methodology of Hadl (2023). This section outlines the program composition. For further details, refer to the original paper.

Theory The theory contains the definitions for the operators and intermediate predicates and can be found in full in Appendix 7.2. The order of operations, i.e., the flow of the program, is controlled by a state predicate along with input and output timesteps, referenced as T_i and T_o, respectively. Each operation takes place at T_i and modifies the state at T_o. For brevity and consistency with the literature, let I be an object identifier, C a class, A and V an attribute category and value, respectively, and R a relation. The select predicate, for example, is then defined as:

$$\begin{aligned}&\#\text{defined select/3.}\\&\text{state}(T_o, I) \text{ :- } \text{select}(T_o, T_i, C), \text{state}(T_i, I), \text{has\_attr}(I, \text{class}, C).\end{aligned} \tag{14}$$

In other words, select is a predicate of arity 3, taking the output timestep, input timestep and class as arguments. The state at T_o is derived from the state at T_i if the object I contained in the state at T_i has the class C. The other operations are defined similarly.

The final end predicate is defined as:

$$\begin{aligned}&\#\text{defined end/1.}\\&\text{ans}(V) \text{ :- } \text{end}(T_o), \text{attr\_value}(T_o, V).\\&\text{ans}(A) \text{ :- } \text{end}(T_o), \text{attr}(T_o, A).\\&\text{ans}(R) \text{ :- } \text{end}(T_o), \text{rel}(T_o, R).\\&\text{ans}(B) \text{ :- } \text{end}(T_o), \text{bool}(T_o, B).\\&\text{ :- not ans(\_).}\\&\#\text{show ans/1.}\end{aligned} \tag{15}$$

End has the role of converting the state at the last step into a unified answer predicate, whether that be an attribute type or value, a relation or a Boolean. The constraint :- not ans(_) ensures that an answer is produced.

Scene Encoding The scene encoding is a question-specific partial scene graph with node and edge weights, derived from the combined output of the object detector and concept classifier. The contribution of the object detector might look like Listing 16.

$$\begin{aligned}
&object(o0).\\
&has\_obj\_weight(o0, 1392).\\
&has\_attr(o0, class, table).\\
&has\_attr(o0, class, furniture).\\
&has\_attr(o0, name, table).\\
&has\_attr(o0, hposition, middle).\\
&has\_attr(o0, vposition, middle).
\end{aligned} \tag{16}$$

Each object is assigned a unique identifier, a weight (i.e., inverse confidence) and a basic set of attributes, namely class membership (of which the most specific one doubles as name) and position. This is supplemented by the concept classifier, which provides likelihoods for additional attributes and relations between objects. A partial example of this concept encoding is given in Listing 17.

$$\begin{aligned}
&has\_attr(o0, color, white).\\
&:\sim\ has\_attr(o0, color, white).[156, (o0, color, white)]\\
&:\sim\ not\ has\_attr(o0, color, white).[1932, (o0, color, white)]\\
&has\_attr(o0, color, yellow).\\
&:\sim\ has\_attr(o0, color, yellow).[722, (o0, color, yellow)]\\
&:\sim\ not\ has\_attr(o0, color, yellow).[664, (o0, color, yellow)]\\
&\\
&has\_rel(o0, on\_top\_of, o1).\\
&:\sim\ has\_rel(o0, on\_top\_of, o1).[692, (o0, on\_top\_of, o1)]\\
&:\sim\ not\ has\_rel(o0, on\_top\_of, o1).[693, (o0, on\_top\_of, o1)]
\end{aligned} \tag{17}$$

Recall $:\sim$ denoting a weak constraint, thus, this section of the encoding can be interpreted as assuming that no object has any attribute or relation. If, however, no solution can be reached, an interpretation assigning given attributes or relations is considered at the cost of the corresponding weight. In the case of relations like *relate_any* or *common*, which do not constrain the type of object or relation, the scene encoding can become very large, leading to a combinatorial explosion in the search space.

Question Encoding The question encoding is a symbolic representation of the question, which guides the flow of the program. In the original pipeline this is derived from the GQA semantic representation, the idea of this thesis is to break this dependency and use a language model to generate the question encoding.

4.5 Experiment Setup

To answer the main research question, it is first necessary to find a set of parameters, that maximizes translation quality. In theory, this could be done by performing a grid search over the parameter space, but due to the computational cost of LLM inference, this is not feasible. Even a single pass over the validation set is prohibitively expensive.

On the utilized hardware (see Table 2), answering a question with a 7 billion parameter model takes 20 to 40 seconds, with 132,062 questions in the validation set, this would take one to two months. To mitigate this, all runs are restricted to a small set of questions, depending on the research question.

Component	**Specification**
CPU	Intel Core i7-13700K
RAM	32GB DDR5
GPU	NVIDIA RTX 4080
VRAM	16GB GDDR6X

Table 2: Specifications of the machine used for local experiments.

Instead of performing said grid search for all experiments, a base setup was heuristically chosen, and adjustments were made by probing the parameter space. Finding the optimal values coincides with answering the auxiliary research questions, so it is natural to structure the discussion around these questions. This section outlines the experiments, while the results are presented in Chapter 5. While minor differences in sensitivity to the parameters are expected when using different LLMs, for the sake of simplicity, it is assumed that the results are generalizable. Thus, the parameter search in research questions 1 and 2 is performed exclusively using *Zephyr7B*.

All LLM inference was performed with the temperature parameter set to 0, as creativity is not desired in this context. Besides limiting the number of output tokens, no further parameters were adjusted from their default values.

(ARQ 1) Influence of prompt parameters on translation quality Since there is considerable interaction between the parameters, they cannot be studied independently. The effect of adding examples until every operation is covered, for example, has a much greater effect if otherwise only few examples would have been given. Estimating the degree of this interaction necessitates a grid search, though it can be relatively coarse. The dimensions of the grid search are shown in Table 3.

Parameter	**Values**
In-context Examples	`[1, 3, 5, 10]`
Sampling Strategy	`[Random, Jaccard, BERT]`
Cover all Operations	`[True, False]`

Table 3: Grid search setup for ARQ 1.

While the final performance between representations differs, it is assumed that the optimal prompt parameters are similar for all. Thus, the grid search is performed using the nested ASP representation only. Each run is restricted to the first 25 questions from the validation set. Results are reported both in terms of raw accuracy and accuracy compared to the baseline Oracle.

Additionally, 10 different prompt templates are compared, as even rephrasing the task can have a significant impact on performance. The prompts can be found in Appendix 7.1. Separating the prompt search from the parameter search is necessary, since it can be assumed that it is largely independent of the other parameters and model-specific.

As opposed to the parameter search, which is performed with Zephyr7B with an empirically optimized prompt for the model, the goal of the prompt search is finding a prompt which works well for the GPT model family, thus, GPT-3.5 is invoked. The sample size is increased to 40 questions, but restricted to answerable questions, so the results are only comparable to the baseline Oracle.

(ARQ 2) Comparison of Representations Building upon the results of the first research question, the best prompt parameters are used to compare the representations. Comparing the flat-asp, nested-asp and GQA representations can be accomplished by answering the same set of questions with each and comparing the resulting accuracy. However, the code-like representation in the chosen format of introducing variables only when intermediate results are reused, rarely differs from the nested representation. In fact, in the scope of the experiment, the code-like representation is equivalent to the nested representation in all cases. To highlight the differences, a selection of questions with this property is chosen and the code-like representation is compared to the nested representation. Again, results are reported both raw and compared to the baseline Oracle.

(ARQ 3) Influence of Model Size The initial question of whether a locally hosted model can be used to achieve acceptable translation quality is technically already answered by the previous experiments, since they are performed with Zephyr7B. Nonetheless, further investigation into the tradeoff is warranted, thus, a larger set of 150 questions is answered by all models. To save on API- and computational costs, questions are prefiltered based on answerability, as described in Section 3.4, so the resulting score is only comparable to the baseline Oracle.

(RQ) Quality of Translation The main research question is answered by picking the best configuration from the previous experiments and answering 250 questions from the validation set with the LLM as part of the full VQA system described in Section 3.2. As in auxiliary research question 3, the questions are prefiltered based on answerability. The results are reported as isolated accuracy in the perfect information case, full system accuracy, full system accuracy with blind concept extraction, as described in Section 4.2, all scored by varying levels of generosity and top-k values, as described in Section 3.4.

Recall specifically, that top-1, top-3 and top-5 are chosen, as no improvement was observed when using larger top-k values. *Generous* denotes that synonyms or more specific concepts are accepted as correct, while *generous+* denotes that up to 5 WordNet levels of hypernym relations are accepted as correct.

5 Results

This section presents the results of the experiments in the spirit of answering the (auxiliary) research questions posed in Section 1.2. Recall the Oracle to be a question encoding derived from a semantic question representation, unavailable in practice. While the discussion revolves around total accuracy, comparison to the Oracle helps give an idea which questions were answerable given the scene encoding produced by the visual component.

5.1 (ARQ 1) Influence of prompt parameters on translation quality

Examining the influence of the number of in-context examples and the *cover_operators* option, Table 4 shows that the number of examples has a significant impact on translation quality. With only one example, the model answers less than 15% of the questions correctly. Adding four more increases this to 38.67%, until performance reaches around 53% with fifteen examples. This is unsurprising, as more examples help the LLM imitate the structure of the target language. The parameter *cover_operators*, which controls whether examples should include all operators, has a drastic impact, if the number of examples is low, more than doubling performance if just one similar example is included.

The effect is less pronounced with more examples and can be unpredictable, as new samples are essentially chosen at random. This leads to the counterintuitive result that performance seems to decline between five and ten similar examples if the *cover_operators* parameter is set to *True*; an artifact of this randomness. The overall performance improvement still observable, as the correct answer rate increases by around 1% per example if 15 similar examples are provided. This strongly suggests that the benefit mostly comes from including more examples, rather than more diverse ones.

In fact, more similar examples seem disproportionately valuable, as comparing setups with a given number of similar examples to others with on average the same number of total examples, the former consistently outperforms the latter.

Cover Operators	#Similar Examples	#Avg Examples	Accuracy	Accuracy vs. Oracle
False	1	1.0	14.67%	17.46%
	3	3.0	28.00%	33.33%
	5	5.0	38.67%	46.03%
	10	10.0	42.67%	50.79%
	15	15.0	53.33%	61.90%
True	1	8.69	33.33%	39.68%
	3	9.83	37.33%	44.44%
	5	11.27	46.67%	55.56%
	10	15.63	41.33%	49.21%
	15	20.16	58.67%	69.84%

Table 4: Translation performance with varying number and type of in-context examples.

Directly comparing similarity sampling schemes yields further evidence for this, as using a similarity measure more than doubles the accuracy, as shown in Table 5. BERT similarity is the most effective at 51.6% relative correct answers, but Jaccard similarity also performs much better at 46% than random sampling with 20.8%. The results are consistent across the other prompt parameters.

Sampling Strategy	Accuracy	Accuracy vs. Oracle
BERT	51.60%	60.95%
Jaccard	46.00%	54.76%
Random	20.80%	24.76%

Table 5: Translation performance with varying sampling strategy.

Finally, the prompt comparison yields the results in Table 6. Most prompts perform similarly at around 70-72.5%, but the prompt labeled *v6* stands out with 77.5% and is thus used for the experiments involving the GPT model family. Notably, the *v1* prompt, which is optimized for completion models and lacks an imperative directive to translate the final question performs poorly.

In smaller models, separating the target question was observed to result in continuation of the task description, rather than compliance. The model may add further instructions or hallucinate additional examples. If instead, the prompt contains the implicit directive by listing the examples with their translation, followed by the target question, the logical continuation is the translation of the target question, which is the desired behavior.

For the GPT family of models the output was mostly of the correct format and syntax[9], it was poor in quality compared to much smaller models when using this strategy. This provides further evidence for the importance of model-specific prompts.

Prompt Version	Accuracy
v6	77.50%
v2	72.50%
v4	72.50%
v5	72.50%
v8	72.50%
v9	72.50%
v10	70.00%
v7	70.00%
v3	65.00%
v1	55.00%

Table 6: Translation performance with varying prompts.

In summary, it is highly advisable to choose examples broadly based on similarity and as plenty as possible, while staying aware of the tradeoff between translation quality and computational cost and the diminishing return of each individual example. The choice of prompt is also crucial and should be tailored to the model used.

[9]Refusal to translate with output similar to “the question is not clear” was observed as a rare occurrence

5.2 (ARQ 2) Comparison of Representations

Table 7 shows a comparison between the discussed representations. Definitively answering the question of which representation is best suited definitively is not trivial, as the results depend on the model choice, prompt format and even the specific question. As hypothesized, the native GQA performs poorly, arguably due to heavy syntax dependence and redundant information. While the LLM is seemingly able to handle the flat representation well, the nested representation, due to the more explicit references by position rather than index, lends itself even better to the task. As the code-like representation can be seen as a special case of the nested representation handling convergent dependencies, it is unsurprising that it performs better still.

Target Representation	Accuracy	Accuracy vs. Oracle
Nested ASP	58.00%	76.32%
Flat ASP	50.00%	65.79%
GQA	42.00%	47.37%

Table 7: Translation performance with varying representation.

The results of the experiment comparing the code-like representation to the nested one can be seen in Table 8. The difference in performance is relatively minor. Out of 50 questions, the code-like representation answers 3 more correctly, which is already an exaggerated difference, as the experiment was conducted exclusively with the type of question that the code-like representation is designed for. Note, that performance is worse on this subset, even though Oracle performance is steady at 90%; a result of the fact that more complex questions are more likely to include convergent dependencies. This complexity is not a problem for the solver, but can push the LLM to the limits of its reasoning capabilities, regardless of the dependency structure.

Target Representation	Accuracy	Accuracy vs. Oracle
Code-like ASP	44.00%	46.67%
Nested ASP	38.00%	40.00%

Table 8: Examination of code-like Representation.

In summary the code-like representation seems to be the most suitable representation for this task, but it is not clear how this generalizes to other logical languages. It can be assumed that performance mostly depends on similarity between training data and the target language.

5.3 (ARQ 3) Influence of Model Size

Table 9 shows the results of the experiments comparing different models. It is evident that the largest model outperforms the smaller ones by a wide margin. However, the smaller models answer some of the questions correctly, proving that they are capable of producing syntactically correct translations. This is a promising result, as it indicates that even smaller models can be used, even if they might require more guardrails or post-processing. More promising still and, is the fact that the locally hosted *zephyr7B* outperforms the larger, proprietary *GPT-3.5 turbo*. This surprising result proves, that the model size is not the only factor determining performance, and that the choice of model can have a significant impact on the results.

Model	Accuracy
GPT-4 turbo	85.00%
zephyr7B	72.00%
GPT-3.5 turbo	67.33%
orca3B	32.67%

Table 9: Comparison of large language models.

5.4 (RQ) Quality of Translation

The main research question of this thesis is whether LLMs are capable of translating natural language questions into logical programs. Table 10 shows that this is the case, and pipeline performance does not disproportionately degrade when the LLM translation component is included.

Model	Generosity	Top 1	Top 3	Top 5	Top 1 vs. Base
GPT-4 Isolated	Strict	84.40%	84.40%	84.40%	-
Base Pipeline	Strict	38.80%	47.20%	50.80%	100.00%
	Generous	40.00%	48.80%	52.40%	100.00%
	Generous+	42.00%	51.20%	54.40%	100.00%
GPT-4 Pipeline	Strict	32.40%	40.40%	42.40%	78.35%
	Generous	33.60%	42.00%	44.00%	79.00%
	Generous+	35.20%	44.00%	45.60%	79.05%
GPT-4 Pipeline Blind	Strict	34.00%	42.00%	44.00%	81.44%
	Generous	35.20%	43.60%	45.60%	82.00%
	Generous+	36.80%	45.60%	47.20%	81.90%

Table 10: End-to-end pipeline performance with LLM translation component for varying degrees of answer acceptance generosity.

GPT-4 performs reasonably well in both the perfect- and imperfect information setting, answering 32.4% to 45.6% of questions correctly, depending on the strictness of the evaluation (recall Section 3.4 for details). Given that the base pipeline (including the Oracle question encoding) answers 38.8% to 54.4% correctly, this means around 20% of the error can be attributed to the translation component. As hypothesized, the blind concept extraction, while technically less accurate, improves the pipeline performance by around 2%, as it is tailored to the vocabulary of the question encoding, rather than that of the dataset. Compared to the Oracle question encoding this difference approaches 3%. In theory, it should therefore benefit more from the generous interpretation, but the experiments do not confirm this, possibly due to the limited scope of just 250 questions. The effect of the generous interpretation is minor overall, due to the fact, that the questions already mostly match the dataset vocabulary. In a realistic setting, this would not be the case.

In summary the results show that neuro-symbolic VQA utilizing LLMs for question translation is feasible. While it introduces a significant error source, extending the pipeline to handle natural language with an open vocabulary is well worth it. The lack of alternatives beyond brittle rule-based systems makes this a promising approach.

The final chapter will highlight the success and shortcomings of this approach and suggest directions for future work.

6 Discussion

This chapter categorizes error types, summarizes the findings of the thesis and discusses limitations as well as directions for future work.

6.1 Analysis of Failure Cases

In aiming to improve the system, it has merit to consider the failure cases of the pipeline, as they may point towards error sources. The following sections discuss these failures, first as they appear in the GQA semantic interpreter, mostly pointing to data quality issues, then as they appear in the translation component.

6.1.1 Semantic Interpreter Failures

This section discusses imperfections in the dataset, which are relevant for the evaluation of the translation system. First, some more subtle inconsistencies are discussed, then the reasons for failure of the semantic interpreter are investigated.

Notably, a given semantic question representation does not always fully match the question. One example of this leads to a failure case, discussed in Section 6.1.2. Here, the question asks about a *white object*, while the semantic representation constrains the object to be a *plate*. While the object in question is indeed a plate, the semantic representation should not contain this type of information, as it is not directly inferable from the question. This has the possibility to further widen the gap between methods that rely on the semantic representation and those that do not.

Error Categorization The dataset contains imperfections which lead to questions being unanswerable with the provided scene graph. For most approaches, this is not an issue, as the scene graph is most commonly used purely for debugging reasons. However, for the evaluation of a question translation component in isolation, it is important to be aware of this shortcoming.

Using the semantic interpreter introduced in 4.1, it is generally possible to narrow down the reasons for the unanswerability to a single problematic operation. From there, it is possible to categorize the errors into a few distinct categories. Some of the categories contain errors due to limitations of the semantic interpreter itself, depending on the configuration. An example would be the *IncompleteMetadataException*, which only appears if the less general GQA-specific category mapping as compiled by Amizadeh et al. (2020) and Hadl (2023) is used. Exact frequencies of the error categories presented are reported in Table 11 for the interpreter with and without metadata.

The following errors were identified:

EmptyQueryException is thrown if the semantic interpreter is unable to find the attribute queried for the given node. This is usually a data issue, but can also be a late sign of failure. If an earlier operation returns no result, logically, no attribute can be derived from it.

AmbiguousAnswerException is thrown if multiple answer candidates remain at the end of program execution.

Error Category	# Occurrences *pure*	# Occurrences *w/ metadata*	Relative % *pure*	Relative % *w/ metadata*
EmptyQueryException	18106	8918	13.71%	6.75%
AmbiguousAnswerException	3801	6067	2.88%	4.59%
MissingEdgeException	548	373	0.42%	0.28%
EmptyChoiceException	226	226	0.17%	0.17%
NonTrivialCategoryException	116	116	0.09%	0.09%
NotInWordnetException	98	84	0.07%	0.06%
IncompleteMetadataException	-	2122	-	1.61%
Uncategorized	89	94	0.07%	0.07%

Table 11: Error categories and their frequencies in the validation set of GQA, with and without metadata.

MissingEdgeException indicates that the functional program includes traversal of an edge, which is not present in the scene graph. It can occur either on a *Relate* operation or if neither option of a *Choose Relation* operation return a result. Inspection of problematic questions strongly suggests this to be purely a data issue.

EmptyChoiceException is thrown in the case that a more-or-less type question is asked, such as "Which is healthier?". This is partially a data issue, as these comparisons are not directly annotated most of the time. If they are, it is in the form of an attribute being present for one of the options, such as *Apple* having the attribute *healthy*. In other cases, it requires background knowledge such as *Man* being older than *Boy*.

NonTrivialCategoryException is thrown if the semantic interpreter attempts a comparison of two nodes along a category that requires additional knowledge. An example would be *Do the woman and the person on the right have the same gender?*, as this is not directly inferable from the scene graph.

NotInWordnetException indicates that a category membership test was performed with a category that is not part of WordNet. This is not a data issue, as the attribute in question either does or does not fall under the category, but it cannot be determined via the WordNet resolver. If the curated category mapping compiled by Amizadeh et al. (2020) and Hadl (2023) it occurs less often, as the WordNet resolver is the fallback option in this case.

IncompleteMetadataException is thrown if the category mapping is missing entries. This is not a dataset issue, as the mapping is not natively part of the dataset.

As established in Section 3.4, ignoring these potentially corrupted questions is necessary to avoid skewing the results. Unfortunately, evaluating within the full pipeline adds another class of errors, which are discussed in the following section. Questions which should not be answerable, but are.

6.1.2 Pipeline Failures

Curiously, there are questions that the pipeline with an LLM component answers correctly, while the pipeline with an Oracle does not. In some cases, this is due to the LLM producing a more general answer, while the Oracle produces a more specific one, which may push the limits of the object detector. In most cases, however, it is due to a subtle question bias.

When a question deals with the existence of a missing object, various failures in the pipeline result in a correct answer, since the final *exist* predicate returns false if the state contains no objects. This can be due to a false negative in the object detection, but in the scenario of *correct answer despite wrong baseline* traces back to a faulty logic program. This section gives examples of such unexpectedly right answers. Note, that these result from questions, which were answerable in the perfect information setting, so the Oracle translation is guaranteed to be correct.

False Positive Detections Given the image in Figure 6 paired with the question "Are there any forks on the white object on top of the table?"[10].

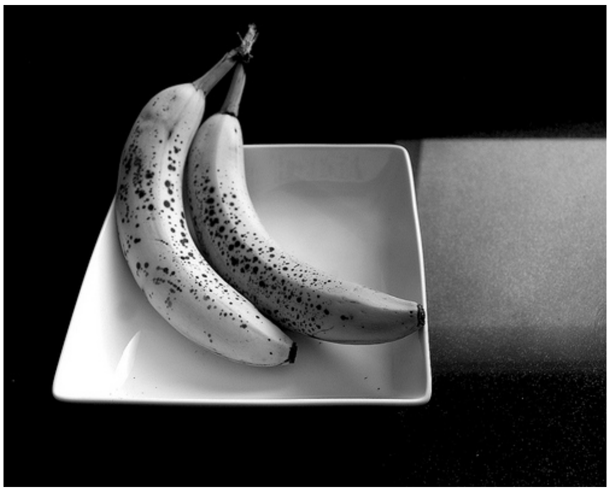
Figure 6: Image with ImageID 2375361 from Krishna et al. (2016).

The attached logic program in GQA native representation already constrains the *white object* to *plate*, which produces the Oracle translation in Listing 18. This inconsistency is an issue in itself and discussed in Section 6.1.1.

$$\begin{aligned}
&scene(0).\\
&select(1, 0, table).\\
&relate(2, 1, plate, on\_top\_of, subject).\\
&filter(3, 2, color, white).\\
&relate(4, 3, fork, on, subject).\\
&exist(5, 4).\\
&end(5).
\end{aligned} \tag{18}$$

It fails due to the object detector not only detecting multiple plates, but also several non-existent forks, resolving to an affirmative answer. The LLM on the other hand produces the translation in Listing 19.

[10]QuestionID: *0416216*

$$
\begin{aligned}
&scene(0).\\
&select(1, 0, table).\\
&relate(2, 1, object, on\_top\_of, subject).\\
&filter(3, 2, color, white).\\
&relate(4, 3, fork, on, subject).\\
&exist(5, 4).\\
&end(5).
\end{aligned}
\tag{19}
$$

The program looks sensible, perhaps even more accurate to the question, since it does not magically constrain white objects to be plates. However, the scene encoding does not include the generic class *object* for every detection, leading to an empty state after predicate 2 and avoiding the pitfall of phantom forks.

Nondeterministic Solving Much more difficult to debug are cases such as question *0833129* over the image shown in Figure 7. The question asks *What vehicle is the horse in front of?* and both the LLM and Oracle correctly deduce the logic program in Listing 20.

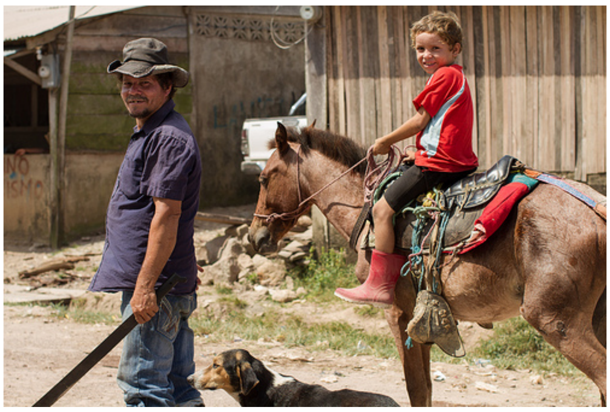
Figure 7: Image with ImageID 2414605 from Krishna et al. (2016).

$$
\begin{aligned}
&scene(0).\\
&select(1, 0, horse).\\
&relate(2, 1, vehicle, in\_front\_of, object).\\
&unique(3, 2).\\
&query(4, 3, name).\\
&end(4).
\end{aligned}
\tag{20}
$$

However, due to the nondeterministic nature of the solver in an optimization setting, the answer is sometimes truck and sometimes suv. Ideally, the generous evaluation would catch this, but "suv" only maps to the Synset *sport_utility.n.01* in WordNet, which is in close proximity to, but not hyper- or hyponym of *truck*.

Vocabulary mismatch Another failure case is question *0832650*: *Does the grass look healthy or unhealthy?* for the same image (see Figure 7). Here, the GPT-4 outputs a program containing the operation $choose\_attr(3, 2, health, healthy, unhealthy)$, which is logically correct, but fails since there exists no attribute category *health* in the GQA ontology, where *healthy* and *unhealthy* are standalone attributes. A method to address this is presented in Section 6.3. The fact that the Oracle translation is incorrect is an artifact of the visual component, which ascribes the attribute *healthy* to the grass.

During preliminary experiments with zephyr7B and GPT-3.5, this class of errors was relatively common, occurring in about 6% of questions. In the final experiments with GPT-4, this number was reduced to 5 occurrences in 250 questions, or 2%.

6.2 Conclusion

This thesis investigated the use of Large Language Model (LLM) for the translation of natural language into logic programs, specifically for the example of Visual Question Answering (VQA) systems using Answer Set Programming (ASP) as a reasoning backend. It set out to answer whether the approach is feasible (RQ), what can be learned about the optimal prompt structure (ARQ 1) and logical representation (ARQ 2) and what scale of LLM is necessary (ARQ 3).

Alongside these questions, contributions to the intuitive evaluation of VQA systems were made. The experiments devised around these questions yield the following key findings:

(ARQ 1) Prompt Parameters The prompt needs to be tailored to the model. The majority of in-context examples should be similar to the question, but examples purely for conveying the structure[11] of the target language can be beneficial. The more in-context examples, the better, but there are diminishing returns and a tradeoff with computational cost to consider.

(ARQ 2) Comparison of Representations The optimal choice of representation is task- and model-dependent, but in general the closer the target language is to the data used in training the model, the better.

(ARQ 3) Influence of Model Size Larger models tend to perform better, but smaller models can be used with acceptable performance. Other aspects, such as well chosen prompts, are more important than sheer parameter count.

(RQ) Quality of Translation LLMs are capable of translating into logical programs and lend themselves to integration as the language component of VQA systems in absence of structured knowledge.

It can be summarized LLMs are a viable choice for the task at hand and complement symbolic reasoning systems well by providing a natural language interface.

6.3 Limitations & Future Work

To conclude the discussion, some limitations and possible directions of future work are mentioned.

[11] in the discussed case this means the syntax of each operation

Computational Considerations To prove useful in interactive applications, a VQA pipeline should ideally resolve an image-question pair in at most a few seconds. For the base pipeline used in the experiments, this was the case with a reported average of 718ms (Hadl 2023). The translation component presented in this thesis has the potential to add significant overhead to this process. For the case of locally hosted models, the inference time was between 20 and 40 seconds depending on model and prompt parameters, with hardware being a significant factor. In the case of cloud-hosted models, the additional latency is likely less significant, but entirely dependent on the time to receive a response for the API call. It can be argued, that the full pipeline no longer meets the real-time requirements. While it stays out of reach for consumer grade single-gpu setups and edge devices for the time being, advances in hardware and inference optimization may make it feasible in the future.

Constrained Vocabularies As hinted towards, the wording of questions may not match the underlying vocabulary of a given dataset, even if many similar examples are given as context. Practical applications, for example, visual aides would likely not suffer from this issue. While it may be argued that building systems for a specific dataset is beside the point of building a general VQA system, it may be harmful to assume a system to be incapable if, in reality, it is merely hindered by such a mismatch.

Feeding an LLM the entire vocabulary as context is not feasible and likely not desirable either, since it would be a significant source of noise. Fine-tuning the model on a constrained vocabulary is a possible solution, but it provides no guarantee that the model will not produce out-of-vocabulary words. It is also expensive and strays further from the goal of a general VQA system.

One option may be adjusting the generation process, without altering the weights of the model. This form of post-processing could take the form of Retrieval-Augmented Generation (Lewis et al. 2021), where a knowledge store is integrated so that each query is supplemented with relevant information. In the case of VQA, this could be the vocabulary of the dataset, such that for each concept mentioned in a question the closest match in the ontology would be added as context, or replacing the word entirely.

Another option would be mathematically guardrailing the output of the model, by enforcing syntax adherence and limiting the vocabulary. If the previous output sequence is "$scene(0).filter(1, 0,$" one may choose to sample only from the tokens which are the beginning of a valid category, since the syntax of the predicate is known to be "$filter(T_o, T_i, ATTR, VALUE).$". This is a form of constrained decoding and quite intrusive to the base generation process. For a commercially hosted model, where only access to the sampled tokens, rather than the output distribution is available, it may not even be an option. In addition, the number of API calls would scale linearly with the length of the output, as each token would have to be requested individually.

Addressing these concerns would be a step towards fast, reliable and interpretable VQA systems, which would enable a wide range of applications in the real world.

References

Agrawal, Aishwarya et al. (Oct. 2016). *VQA: Visual Question Answering.* arXiv:1505.00468 [cs]. DOI: 10 . 48550 / arXiv . 1505 . 00468. URL: http://arxiv.org/abs/1505.00468 (visited on 11/16/2023).

Amizadeh, Saeed et al. (Aug. 2020). *Neuro-Symbolic Visual Reasoning: Disentangling "Visual" from "Reasoning".* arXiv:2006.11524 [cs, stat]. URL: http://arxiv.org/abs/2006.11524 (visited on 02/03/2024).

Basu, Kinjal, Farhad Shakerin, and Gopal Gupta (2020). "AQuA: ASP-Based Visual Question Answering". en. In: ed. by Ekaterina Komendantskaya and Yanhong Annie Liu. Vol. 12007. Book Title: Practical Aspects of Declarative Languages Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 57–72. ISBN: 978-3-030-39196-6 978-3-030-39197-3. DOI: 10 . 1007 / 978 - 3 - 030 - 39197 - 3 _ 4. URL: http://link.springer.com/10.1007/978-3-030-39197-3_4 (visited on 11/16/2023).

Bommasani, Rishi et al. (July 2022). *On the Opportunities and Risks of Foundation Models.* arXiv:2108.07258 [cs]. DOI: 10.48550/arXiv.2108.07258. URL: http://arxiv.org/abs/2108.07258 (visited on 01/07/2024).

bookcorpus · Datasets at Hugging Face (June 2023). URL: https://huggingface.co/datasets/bookcorpus (visited on 01/25/2024).

Brewka, Gerhard, Thomas Eiter, and Mirosław Truszczyński (2011). "Answer set programming at a glance". In: *Communications of the ACM* 54.12, pp. 92–103. ISSN: 0001-0782. DOI: 10 . 1145 / 2043174 . 2043195. URL: https://doi.org/10.1145/2043174.2043195 (visited on 01/06/2024).

Brown, Tom B. et al. (July 2020). *Language Models are Few-Shot Learners.* en. arXiv:2005.14165 [cs]. URL: http://arxiv.org/abs/2005.14165 (visited on 01/07/2024).

Calimeri, Francesco et al. (Mar. 2020). "ASP-Core-2 Input Language Format". In: *Theory and Practice of Logic Programming* 20.2. arXiv:1911.04326 [cs], pp. 294–309. ISSN: 1471-0684, 1475-3081. DOI: 10.1017/S1471068419000450. URL: http://arxiv.org/abs/1911.04326 (visited on 02/09/2024).

Chen, Xi et al. (June 2023). *PaLI: A Jointly-Scaled Multilingual Language-Image Model.* arXiv:2209.06794 [cs]. DOI: 10.48550/arXiv.2209.06794. URL: http://arxiv.org/abs/2209.06794 (visited on 01/20/2024).

Devlin, Jacob et al. (May 2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv:1810.04805 [cs]. DOI: 10.48550/arXiv.1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 01/25/2024).

Eiter, Thomas et al. (Sept. 2022). "A Neuro-Symbolic ASP Pipeline for Visual Question Answering". en. In: *Theory and Practice of Logic Programming* 22.5. Publisher: Cambridge University Press, pp. 739–754. ISSN: 1471-0684, 1475-3081. DOI: 10 . 1017 / S1471068422000229. URL: https://www.cambridge.org/core/journals/theory-and-practice-of-logic-programming/article/abs/neurosymbolic-asp-pipeline-for-visual-question-answering/48A54F7B23E9FAC3AD6AB495CA901701 (visited on 11/16/2023).

Faria, Ana Cláudia Akemi Matsuki de et al. (June 2023). *Visual Question Answering: A Survey on Techniques and Common Trends in Recent Literature.* arXiv:2305.11033

[cs]. DOI: 10.48550/arXiv.2305.11033. URL: http://arxiv.org/abs/2305.11033 (visited on 01/15/2024).

Gelfond, Michael and Vladimir Lifschitz (Aug. 1991). "Classical negation in logic programs and disjunctive databases". en. In: *New Generation Computing* 9.3, pp. 365–385. ISSN: 1882-7055. DOI: 10.1007/BF03037169. URL: https://doi.org/10.1007/BF03037169 (visited on 02/09/2024).

— (Dec. 2000). "The Stable Model Semantics For Logic Programming". In: *Logic Programming* 2.

Hadl, Jan (2023). "GS-VQA: Zero-Shot Neural-Symbolic Visual Question Answering with Vision-Language Models". en. Accepted: 2023-08-31T07:55:06Z. Thesis. Technische Universität Wien. DOI: 10.34726/hss.2023.109680. URL: https://repositum.tuwien.at/handle/20.500.12708/187993 (visited on 11/17/2023).

Hudson, Drew A. and Christopher D. Manning (May 2019). *GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering.* arXiv:1902.09506 [cs]. URL: http://arxiv.org/abs/1902.09506 (visited on 11/17/2023).

Ishay, Adam, Zhun Yang, and Joohyung Lee (July 2023). *Leveraging Large Language Models to Generate Answer Set Programs.* arXiv:2307.07699 [cs]. DOI: 10.48550/arXiv.2307.07699. URL: http://arxiv.org/abs/2307.07699 (visited on 01/12/2024).

Jiang, Albert Q. et al. (Oct. 2023). *Mistral 7B.* arXiv:2310.06825 [cs]. DOI: 10.48550/arXiv.2310.06825. URL: http://arxiv.org/abs/2310.06825 (visited on 03/02/2024).

Johnson, Justin et al. (Dec. 2016). *CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning.* arXiv:1612.06890 [cs]. URL: http://arxiv.org/abs/1612.06890 (visited on 11/17/2023).

Krishna, Ranjay et al. (Feb. 2016). *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations.* arXiv:1602.07332 [cs]. DOI: 10.48550/arXiv.1602.07332. URL: http://arxiv.org/abs/1602.07332 (visited on 11/17/2023).

Lewis, Patrick et al. (Apr. 2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* arXiv:2005.11401 [cs]. DOI: 10.48550/arXiv.2005.11401. URL: http://arxiv.org/abs/2005.11401 (visited on 02/27/2024).

Li, Junnan et al. (Feb. 2022). *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation.* arXiv:2201.12086 [cs]. DOI: 10.48550/arXiv.2201.12086. URL: http://arxiv.org/abs/2201.12086 (visited on 11/17/2023).

Li, Yunxin et al. (Nov. 2023). *A Comprehensive Evaluation of GPT-4V on Knowledge-Intensive Visual Question Answering.* arXiv:2311.07536 [cs]. URL: http://arxiv.org/abs/2311.07536 (visited on 01/20/2024).

Lin, Tsung-Yi et al. (Feb. 2015). *Microsoft COCO: Common Objects in Context.* arXiv:1405.0312 [cs]. DOI: 10.48550/arXiv.1405.0312. URL: http://arxiv.org/abs/1405.0312 (visited on 12/16/2023).

Liu, Nelson F. et al. (Nov. 2023). *Lost in the Middle: How Language Models Use Long Contexts.* arXiv:2307.03172 [cs]. DOI: 10.48550/arXiv.2307.03172. URL: http://arxiv.org/abs/2307.03172 (visited on 01/31/2024).

Liu, Yiheng et al. (Jan. 2024). *Understanding LLMs: A Comprehensive Overview from Training to Inference.* arXiv:2401.02038 [cs]. URL: `http://arxiv.org/abs/2401.02038` (visited on 01/07/2024).

Manhaeve, Robin et al. (Dec. 2018). *DeepProbLog: Neural Probabilistic Logic Programming.* arXiv:1805.10872 [cs]. DOI: `10.48550/arXiv.1805.10872`. URL: `http://arxiv.org/abs/1805.10872` (visited on 01/19/2024).

Marino, Kenneth et al. (Sept. 2019). *OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge.* arXiv:1906.00067 [cs]. DOI: `10.48550/arXiv.1906.00067`. URL: `http://arxiv.org/abs/1906.00067` (visited on 11/17/2023).

Miller, George A. (Nov. 1995). "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11, pp. 39–41. ISSN: 0001-0782. DOI: `10.1145/219717.219748`. URL: `https://dl.acm.org/doi/10.1145/219717.219748` (visited on 01/06/2024).

Minderer, Matthias et al. (July 2022). *Simple Open-Vocabulary Object Detection with Vision Transformers.* arXiv:2205.06230 [cs]. URL: `http://arxiv.org/abs/2205.06230` (visited on 01/20/2024).

Mukherjee, Subhabrata et al. (June 2023). *Orca: Progressive Learning from Complex Explanation Traces of GPT-4.* arXiv:2306.02707 [cs]. DOI: `10.48550/arXiv.2306.02707`. URL: `http://arxiv.org/abs/2306.02707` (visited on 02/15/2024).

OpenAI et al. (Dec. 2023). *GPT-4 Technical Report.* arXiv:2303.08774 [cs]. DOI: `10.48550/arXiv.2303.08774`. URL: `http://arxiv.org/abs/2303.08774` (visited on 01/20/2024).

OpenAI Platform (2024). en. URL: `https://platform.openai.com` (visited on 01/25/2024).

Ouyang, Long et al. (Mar. 2022). *Training language models to follow instructions with human feedback.* arXiv:2203.02155 [cs]. URL: `http://arxiv.org/abs/2203.02155` (visited on 01/07/2024).

Radford, Alec et al. (Feb. 2021). *Learning Transferable Visual Models From Natural Language Supervision.* arXiv:2103.00020 [cs]. DOI: `10.48550/arXiv.2103.00020`. URL: `http://arxiv.org/abs/2103.00020` (visited on 11/16/2023).

Rajasekharan, Abhiramon et al. (Sept. 2023). "Reliable Natural Language Understanding with Large Language Models and Answer Set Programming". In: *Electronic Proceedings in Theoretical Computer Science* 385. arXiv:2302.03780 [cs], pp. 274–287. ISSN: 2075-2180. DOI: `10.4204/EPTCS.385.27`. URL: `http://arxiv.org/abs/2302.03780` (visited on 11/16/2023).

Redmon, Joseph et al. (May 2016). *You Only Look Once: Unified, Real-Time Object Detection.* arXiv:1506.02640 [cs]. DOI: `10.48550/arXiv.1506.02640`. URL: `http://arxiv.org/abs/1506.02640` (visited on 11/17/2023).

Sarri, Arnau Martí and Victor Rodriguez-Fernandez (2021). "An implementation of the "Guess who?" game using CLIP". In: vol. 13113. arXiv:2112.00599 [cs], pp. 415–425. DOI: `10.1007/978-3-030-91608-4_41`. URL: `http://arxiv.org/abs/2112.00599` (visited on 02/09/2024).

Subramanian, Sanjay et al. (June 2023). *Modular Visual Question Answering via Code Generation.* arXiv:2306.05392 [cs]. DOI: `10.48550/arXiv.2306.05392`. URL: `http://arxiv.org/abs/2306.05392` (visited on 11/16/2023).

Surís, Dídac, Sachit Menon, and Carl Vondrick (Mar. 2023). *ViperGPT: Visual Inference via Python Execution for Reasoning.* arXiv:2303.08128 [cs]. URL: `http://arxiv.org/abs/2303.08128` (visited on 11/16/2023).

Tunstall, Lewis et al. (Oct. 2023). *Zephyr: Direct Distillation of LM Alignment.* arXiv:2310.16944 [cs]. DOI: 10 . 48550 / arXiv . 2310 . 16944. URL: http://arxiv.org/abs/2310.16944 (visited on 01/25/2024).

Vaswani, Ashish et al. (Aug. 2023). *Attention Is All You Need.* arXiv:1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 01/07/2024).

Wikipedia (Jan. 2024). de. Page Version ID: 241360521. URL: https://de.wikipedia.org/w/index.php?title=Wikipedia&oldid=241360521 (visited on 01/25/2024).

Xiao, Junbin et al. (May 2021). *NExT-QA:Next Phase of Question-Answering to Explaining Temporal Actions.* arXiv:2105.08276 [cs]. DOI: 10.48550/arXiv.2105.08276. URL: http://arxiv.org/abs/2105.08276 (visited on 11/17/2023).

Yang, Zhun, Adam Ishay, and Joohyung Lee (July 2023a). *Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text.* arXiv:2307.07696 [cs]. DOI: 10 . 48550 / arXiv . 2307 . 07696. URL: http://arxiv.org/abs/2307.07696 (visited on 01/12/2024).

— (July 2023b). *NeurASP: Embracing Neural Networks into Answer Set Programming.* arXiv:2307.07700 [cs]. DOI: 10.48550/arXiv.2307.07700. URL: http://arxiv.org/abs/2307.07700 (visited on 01/19/2024).

Zou, Yeyun and Qiyu Xie (Dec. 2020). “A survey on VQA_Datasets and Approaches”. In: *2020 2nd International Conference on Information Technology and Computer Application (ITCA).* arXiv:2105.00421 [cs], pp. 289–297. DOI: 10 . 1109 / ITCA52113 . 2020 . 00069. URL: http : / / arxiv . org / abs / 2105 . 00421 (visited on 01/24/2024).

7 Appendix

7.1 Prompt Templates

```
GQA:
preprompt:
You are now a question parser. Your task is to translate a question into a functional program.
The available operations are: select, relate, common, verify, choose, filter, query, same, different, and, or, exist.

template:
Here are examples of question and corresponding programs: {examples} {question}

ASP v1:
preprompt:
You are now a question parser. Your task is to translate a question into an Answer Set Program.
The available operations are: scene, end, select, relate, query, verify_rel, choose_attr, choose_rel, exist,
all_same, all_different, two_same, two_same, and, or, unique, negate.

template:
Here are examples of question and corresponding programs: {examples} {question}

ASP v2:
preprompt:
Imagine you are a translator between human language and a programming language called Answer Set Programming (ASP).
Your task is to convert questions into ASP code.

template:
To help you understand the task, here are some examples of questions and their corresponding ASP programs: {examples}
Now, try to translate the following question into an ASP program: {question}

ASP v3:
preprompt:
You are now a question parser. Your task is to translate a question into an Answer Set Program.
```

```
template:
Consider the following examples of questions and their corresponding ASP programs: {examples}
Based on these examples, translate the following question into an ASP program: {question}

ASP v4:
preprompt:
As an AI, you have the ability to understand human language and translate it into Answer Set Programming (ASP).
Your task is to convert the given question into an ASP program.

template:
Here are some examples of questions and their corresponding ASP programs: {examples}
Can you translate the following question into an ASP program? {question}

ASP v5:
preprompt:
You are an AI language translator. Your job is to take a question and translate it into an Answer Set Program.

template:
Look at these examples of questions and their corresponding ASP programs: {examples}
Translate the following question into an ASP program: {question}

ASP v6:
preprompt:
You are an AI that can understand human language and convert it into Answer Set Programming (ASP).
Your task is to translate the given question into an ASP program.

template:
Here are some examples of questions and their corresponding ASP programs to help you understand the task:
{examples} Now, translate the following question into an ASP program: {question}

ASP v7:
preprompt:
Imagine you are an AI that can understand human language and translate it into a programming language called
Answer Set Programming (ASP).
Your task is to convert the given question into an ASP program.

template:
Consider these examples of questions and their corresponding ASP programs: {examples}
Translate the following question into an ASP program: {question}

ASP v8:
preprompt:
You are an AI that can translate human language into Answer Set Programming (ASP).
Your task is to convert the given question into an ASP program.

template:
Here are some examples of questions and their corresponding ASP programs: {examples}
Can you translate the following question into an ASP program? {question}

ASP v9:
preprompt:
As an AI, you have the ability to understand human language and translate it into a programming language
called Answer Set Programming (ASP).
Your task is to convert the given question into an ASP program.

template:
Look at these examples of questions and their corresponding ASP programs: {examples}
Translate the following question into an ASP program: {question}

ASP v10:
preprompt:
You are an AI that can understand human language and convert it into a programming language
called Answer Set Programming (ASP).
Your task is to translate the given question into an ASP program.

template:
Here are some examples of questions and their corresponding ASP programs to help you understand the task: {examples}
Now, translate the following question into an ASP program: {question}
```

7.2 ASP Theory

```
% ========== Scene Graph Definitions ==========
#defined has_attr/3.
#defined has_rel/3.
#defined is_attr/1.
#defined is_attr_value/2.
#defined object/1.
#defined has_obj_weight/2.

% ========== Base Operations ==========
% ---------- scene ----------
#defined scene/1.

state(TO,ID) :- scene(TO), object(ID).

% ---------- end ----------
#defined end/1.

ans(V) :- end(TO), attr_value(TO,V).
ans(V) :- end(TO), attr(TO,V).
ans(V) :- end(TO), rel(TO,V).
ans(V) :- end(TO), bool(TO,V).

% ---------- ans ----------
% At least one answer must be derivable
:- not ans(_).
#show ans/1.


% ========== Intermediary Operations ==========
% ---------- select ----------
#defined select/3.

state(TO,ID) :- select(TO, TI, CLASS), state(TI, ID), has_attr(ID, class, CLASS).

% ---------- filter ----------
#defined filter/4.

state(TO,ID) :- filter(TO, TI, ATTR, VALUE), state(TI, ID), has_attr(ID, ATTR, VALUE).

#defined filter_any/3.

state(TO,ID) :- filter_any(TO, TI, VALUE), state(TI, ID), has_attr(ID, ATTR, VALUE).

% ---------- relate ----------
#defined relate/5.

state(TO, ID') :- relate(TO, TI, CLASS, REL, subject), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
state(TO, ID') :- relate(TO, TI, CLASS, REL, object), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID, REL, ID').

% relate_any
#defined relate_any/4.

state(TO, ID') :- relate_any(TO, TI, REL, subject), state(TI, ID), has_rel(ID', REL, ID).
state(TO, ID') :- relate_any(TO, TI, REL, object), state(TI, ID), has_rel(ID, REL, ID').

% relate_attr
#defined relate_attr/4.

state(TO, ID') :- relate_attr(TO, TI, CLASS, ATTR), state(TI, ID), has_attr(ID, ATTR, VALUE),
has_attr(ID', class, CLASS), has_attr(ID', ATTR, VALUE'), VALUE==VALUE', ID!=ID'.


% ========== Terminal Operations ==========
% ---------- query ----------
#defined query/3.

{ has_attr(ID, ATTR, VALUE) : is_attr_value(ATTR, VALUE)} = 1 :- query(TO, TI, ATTR), state(TI, ID),
ATTR != name, ATTR != class, ATTR != hposition, ATTR != vposition.
attr_value(TO,VALUE) :- query(TO, TI, ATTR), state(TI, ID), has_attr(ID, ATTR, VALUE).
```

```
% ---------- verify ----------
% verify_attr
#defined verify_attr/4.

bool(T0, yes) :- verify_attr(T0, TI, ATTR, VALUE), state(TI, ID), has_attr(ID, ATTR, VALUE).
bool(T0,no) :- verify_attr(T0, TI, ATTR, VALUE), not bool(T0,yes).

% verify_rel
#defined verify_rel/5.

bool(T0, yes) :- verify_rel(T0, TI, CLASS, REL, subject), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
bool(T0,no) :- verify_rel(T0, TI, CLASS, REL, subject), not bool(T0,yes).

bool(T0, yes) :- verify_rel(T0, TI, CLASS, REL, object), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID, REL, ID').
bool(T0,no) :- verify_rel(T0, TI, CLASS, REL, object), not bool(T0,yes).

% ---------- choose ----------
% choose_attr
#defined choose_attr/5.
{has_attr(ID, ATTR, VALUE); has_attr(ID, ATTR, VALUE')} = 1 :- choose_attr(T0, TI, ATTR, VALUE, VALUE'), state(TI, ID).
attr_value(T0, VALUE) :- choose_attr(T0, TI, ATTR, VALUE, VALUE'), state(TI, ID), has_attr(ID, ATTR, VALUE).
attr_value(T0, VALUE') :- choose_attr(T0, TI, ATTR, VALUE, VALUE'), state(TI, ID), has_attr(ID, ATTR, VALUE').

% choose_rel
#defined choose_rel/6.
{has_rel(ID', REL, ID): has_attr(ID', class, CLASS); has_rel(ID', REL', ID):
has_attr(ID', class, CLASS)} = 1 :- choose_rel(T0, TI, CLASS, REL, REL', subject), state(TI, ID).
rel(T0, REL) :- choose_rel(T0, TI, CLASS, REL, REL', subject), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
rel(T0, REL') :- choose_rel(T0, TI, CLASS, REL, REL', subject), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID', REL', ID).

{has_rel(ID, REL, ID'): has_attr(ID', class, CLASS); has_rel(ID, REL', ID'):
has_attr(ID', class, CLASS)} = 1 :- choose_rel(T0, TI, CLASS, REL, REL', object), state(TI, ID).
rel(T0, REL) :- choose_rel(T0, TI, CLASS, REL, REL', object), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID, REL, ID').
rel(T0, REL') :- choose_rel(T0, TI, CLASS, REL, REL', object), state(TI, ID),
has_attr(ID', class, CLASS), has_rel(ID, REL', ID').

% ---------- exist ----------
#defined exist/2.

bool(T0,yes) :- exist(T0, TI), state(TI,ID).
bool(T0,no) :- exist(T0, TI), not bool(T0,yes).

% ---------- different/same ----------
% all_different
#defined all_different/3.

bool(T0,no) :- all_different(T0, TI, ATTR), state(TI, ID), state(TI, ID'),
has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE).
bool(T0,yes) :- all_different(T0, TI, ATTR), not bool(T0,no).

% all_same
#defined all_same/3.

bool(T0,no) :- all_same(T0, TI, ATTR), state(TI, ID), state(TI, ID'),
has_attr(ID, ATTR, VALUE), not has_attr(ID', ATTR, VALUE).
bool(T0,yes) :- all_same(T0, TI, ATTR), not bool(T0,no).

% two_different
#defined two_different/4.

bool(T0, yes) :- two_different(T0, TI0, TI1, ATTR), state(TI0, ID),
state(TI1, ID'), has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE'), VALUE != VALUE'.
bool(T0, yes) :- two_different(T0, TI0, TI1, ATTR), state(TI0, ID),
state(TI1, ID'), has_attr(ID, ATTR, _), not has_attr(ID', ATTR, _).
bool(T0, yes) :- two_different(T0, TI0, TI1, ATTR), state(TI0, ID),
state(TI1, ID'), not has_attr(ID, ATTR, _), has_attr(ID', ATTR, _).
bool(T0,no) :- two_different(T0, TI0, TI1, ATTR), not bool(T0,yes).
```

```
% two_same
#defined two_same/4.

bool(T0, yes) :- two_same(T0, TI0, TI1, ATTR), state(TI0, ID), state(TI1, ID'),
has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE'), VALUE == VALUE'.
bool(T0,no) :- two_same(T0, TI0, TI1, ATTR), not bool(T0,yes).

% ---------- common ----------
#defined common/3.

attr(T0, ATTR) :- common(T0, TI0, TI1), state(TI0, ID), state(TI1, ID'),
has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE), ATTR != name,
ATTR != class, ATTR != hposition, ATTR != vposition.
{attr(T0, ATTR): is_attr(ATTR)} = 1 :- common(T0, TI0, TI1).

% ---------- compare ----------
#defined compare/5.

state(T0,ID) :- compare(T0, TI0, TI1, VALUE, true), state(TI0, ID), state(TI1, ID'),
has_attr(ID, _, VALUE), not has_attr(ID', _, VALUE).
state(T0,ID') :- compare(T0, TI0, TI1, VALUE, true), state(TI0, ID), state(TI1, ID'),
not has_attr(ID, _, VALUE), has_attr(ID', _, VALUE).


state(T0,ID') :- compare(T0, TI0, TI1, VALUE, false), state(TI0, ID), state(TI1, ID'),
has_attr(ID, _, VALUE), not has_attr(ID', _, VALUE).
state(T0,ID) :- compare(T0, TI0, TI1, VALUE, false), state(TI0, ID), state(TI1, ID'),
not has_attr(ID, _, VALUE), has_attr(ID', _, VALUE).


% ========== Utility Operations ==========
% ---------- boolean ----------
% and
#defined and/3.

bool(T0,yes) :- and(T0, TI0, TI1), bool(TI0,yes), bool(TI1,yes).
bool(T0,no) :- and(T0, TI0, TI1), not bool(T0,yes).

% or
#defined or/3.

bool(T0,yes) :- or(T0, TI0, TI1), bool(TI0,yes).
bool(T0,yes) :- or(T0, TI0, TI1), bool(TI1,yes).
bool(T0,no) :- or(T0, TI0, TI1), not bool(T0,yes).

% ---------- unique ----------
#defined unique/2.

{state(T0,ID): state(TI,ID)} = 1 :- unique(T0, TI).
:~ unique(T0, TI), state(T0,ID), has_obj_weight(ID, P). [P, (T0, ID)]

% ---------- negate ----------
#defined negate/3.
state(T0, ID) :- negate(T0, TI0, TI1), state(TI1, ID), not state(TI0, ID).
```

Acronyms

AI Artificial Intelligence. 1, 4

ASP Answer Set Programming. iv, 2, 4, 6, 7, 11, 12, 13, 14, 17, 19, 21, 22, 25, 29, 31, 42

CV Computer Vision. 4

EoS End of Sequence. 5

LLM Large Language Model. ii, iii, iv, 1, 2, 4, 5, 13, 14, 15, 19, 21, 23, 24, 27, 30, 31, 32, 33, 35, 36, 37, 39, 40, 41, 42, 43

ML Machine Learning. 17

NLP Natural Language Processing. 4, 12, 14

NLU Natural Language Understanding. 14

QA Question Answering. 4

VLM Vision Language Model. 1, 11, 13, 18, 24, 27

VQA Visual Question Answering. ii, iii, iv, 1, 2, 4, 8, 10, 11, 12, 13, 14, 15, 17, 18, 24, 32, 37, 42, 43