TU WIEN Informatics

GS-VQA

Zero-Shot Neural-Symbolic Visual Question Answering with Vision-Language Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Jan Hadl, BSc (WU)
Matrikelnummer 01609664

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Mitwirkung: Projektass. Dipl.-Ing. Dr.techn. Johannes Oetsch, Bakk.techn.
Projektass. Nelson Nicolas Higuera Ruiz, Magíster en Ciencias

Wien, 20. August 2023

Jan Hadl

Thomas Eiter

TU WIEN Informatics

GS-VQA

Zero-Shot Neural-Symbolic Visual Question Answering with Vision-Language Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Jan Hadl, BSc (WU)
Registration Number 01609664

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Assistance: Projektass. Dipl.-Ing. Dr.techn. Johannes Oetsch, Bakk.techn.
Projektass. Nelson Nicolas Higuera Ruiz, Magíster en Ciencias

Vienna, 20th August, 2023

Jan Hadl | Thomas Eiter

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Jan Hadl, BSc (WU)

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. August 2023

Jan Hadl

Acknowledgements

This was an insane plan and somehow it worked! I'm going to be talking to someone again. I spent three months as the loneliest man in history and it's finally over.

The Martian, Andy Weir

I would like to express my gratitude towards Professor Thomas Eiter, Johannes Oetsch, and Nelson Higuera Ruiz for their continued support throughout the entire process of bringing this thesis to life. Without your ideas, insights, and feedback, this work could not have turned out the way it did.

Also, from the bottom of my heart, I want to thank my parents Erik and Sabine, my siblings Marc and Fiona, and my grandma Rosemarie. For all my life, you have been my emotional safe haven and my biggest supporters. Thank you for believing in me way more than I could ever do myself.

Kurzfassung

Visual Question Answering (VQA) stellt Machine Learning (ML) Systeme vor die Aufgabe, eine über ein Bild gestellte Frage in natürlicher Sprache zu beantworten. Um diese Aufgabe zu erfüllen, benötigen ML Systeme nicht nur ein gemeinsames Verständnis von Bild- und Textdaten, sondern müssen auch in der Lage sein, komplexe Abfolgen von Gedankenschritten durchzuführen. Neural-Symbolic Ansätze für VQA nutzen Deep Learning zur visuellen Wahrnehmung und erstellen eine symbolische Repräsentation der Information, die im Eingabe-Bild und der Eingabe-Frage enthalten ist. Auf Basis dieser Repräsentation wird Reasoning rein symbolisch durchgeführt, um die Antwort auf die Eingabe-Frage herzuleiten. Zu den Vorteilen von Neural-Symbolic Ansätzen gehören ihre Nachvollziehbarkeit, ihre Konsistenz, und ihre Erweiterbarkeit dank ihres modularen Aufbaus. Aktuelle VQA Ansätze, die Reasoning rein symbolisch durchführen, haben jedoch die Limitierung, dass die von ihnen zur visuellen Wahrnehmung verwendeten ML Modelle für den aktuell verwendeten Datensatz trainiert oder fine-tuned werden müssen.

Zur Erforschung eines Ansatzes, mit dem diese Limitierung beseitigt werden kann, designen und implementieren wir die GS-VQA Pipeline für Neural-Symbolic VQA am GQA Datensatz, einem aktuellen und generalistischen Datensatz mit detailreichen Bildern und vielseitigen Fragen mit einer großen Anzahl an möglichen Antworten. Die Pipeline baut auf den jüngsten Erfolgen auf, die Architekturen und Trainings-Strategien von Large Language Models (LLMs) auf multi-modale Vision Language Models (VLMs) anzuwenden. Durch die effiziente und effektive Nutzung dieser VLMs ist GS-VQA in der Lage, VQA zero-shot – also ohne das Training oder Fine-Tuning von Modellen am behandelten Datensatz – durchzuführen. GS-VQA kann 39.50% der Fragen aus GQAs test-dev Set korrekt beantworten. Im Vergleich dazu erreicht das aktuell beste zero-shot Modell für VQA am selben Set eine Accuracy von 49.00%. Wir implementieren und evaluieren zudem mehrere Erweiterungen zur GS-VQA Pipeline, die den Prozentsatz der korrekt beantworteten Fragen auf 40.55% erhöhen.

Abstract

Visual Question Answering (VQA) presents the following task to machine learning (ML) systems: given an image and a natural-language question about the image, provide an accurate natural-language answer. Performing this task requires not just a joint understanding of vision and text, but also the ability to follow complex chains of reasoning operations. Neural-symbolic approaches to VQA use deep learning for perception, producing a symbolic representation of the information contained within the input image and question, and then perform reasoning on this representation purely symbolically. These approaches are able to reason transparently, behave consistently, and be extended easily due to their compositional structure. However, current VQA pipelines that perform reasoning purely symbolically require the training of purpose-built models for visual perception on the dataset at hand.

To explore a way to remove this limitation, we design and implement the GS-VQA pipeline for neural-symbolic VQA on GQA, a challenging and generalist dataset with images depicting complex visual scenes, and diverse questions with a large number of possible answers. The pipeline builds on the recent successes in adapting the model architectures and training regimes of large language models (LLMs) to multi-modal vision-language models (VLMs). By using these VLMs efficiently and effectively, GS-VQA is able to perform VQA in a zero-shot manner, that is, without the training or fine-tuning of models to the current dataset. Of the questions in the test-dev set of GQA, GS-VQA is able to answer 39.50% correctly, compared to the 49.00% achieved by the current best zero-shot approach for GQA. We also implement and evaluate multiple extensions to the core pipeline architecture, which improve the answer accuracy further to 40.55%.

Contents

CHAPTER 1

Introduction

Take a look at the image and question in Figure 1.1. To you, it is probably trivial to conclude that the answer to the question is "yellow", but to a machine learning (ML) system, finding the right answer presents a major challenge.

Visual Question Answering (VQA) formalises the task that you just performed: given an image and a natural-language question about the image, provide an accurate natural-language answer [AAL+15]. While the task is easily explained and performed by humans, its implications for ML systems are profound: apart from a joint understanding of the vision and text data presented to it as input, a ML system for VQA must be able to combine multiple reasoning operations to arrive at the correct answer. These reasoning operations are both complex and diverse, and include filtering ("What do the *red* cars have in common?"), spatial reasoning ("What color are the shoes of the man *standing behind* a lamppost?"), comparisons ("Is the picture on the left *taller than* the one on the right?"), and counting ("*How many* motorcycles do you see?").

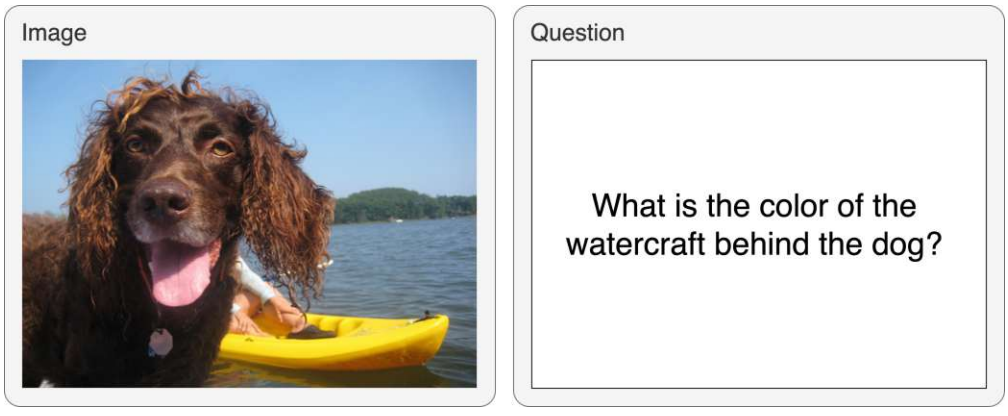


Figure 1.1: An example of a VQA task (Image Source: Visual Genome [KZG+17])

In recent years, a large variety of approaches have been introduced for the VQA task. Neural-symbolic approaches [YWG+18; APP+20; EHO+22] use deep learning for perception, producing a symbolic representation of the information contained within the input image and question, and then perform reasoning on this representation purely symbolically. Since the semantics of the reasoning formalism are known, the way in which an answer is reached is transparent [EHO+22]. Compared to end-to-end models that produce an answer in a single forward pass of some neural network, neural-symbolic approaches also benefit from consistency and compositionality: Since reasoning operations are performed symbolically, they are guaranteed to behave in a consistent manner (e.g., counting always obeys the rules of arithmetic). And since the perceptive component of a neural-symbolic VQA system can be composed of multiple neural networks contributing different parts of the symbolic representation, advancements in specific computer vision (CV) or natural language processing (NLP) disciplines (object detection, classification, translation, etc.) can be directly applied to yield a more accurate symbolic representation of the input text or image.

One such advancement is the adaptation of the Transformer [VSP+17] neural network architecture and training on vast amounts of data with task-agnostic pre-training objectives from large language models (LLMs) to multi-modal vision-language models (VLMs). As with LLMs for language tasks, this has led to models that can be applied to a wide variety of vision and vision-language tasks in a zero-shot manner, i.e., without training or fine-tuning the model for the task or dataset at hand.

1.1 Motivation & Problem Statement

Since the tasks supported by the strong multi-modal understanding of state-of-the-art VLMs include those required by the visual perception component of a neural-symbolic VQA pipeline, it is natural to wonder if the two and their benefits can be combined to yield a system that can perform zero-shot visual question answering with transparent, symbolic reasoning. More concretely, we pose the following main research question (RQ):

(RQ) *What accuracy can be achieved on a current VQA benchmark dataset with a zero-shot neural-symbolic VQA pipeline that uses VLMs for visual perception?*

The construction and evaluation of such a neural-symbolic VQA pipeline is thus the main goal of this thesis. Furthermore, we seek to answer multiple auxiliary research questions (ARQs) to gain deeper insight into the capabilities and limitations of the pipeline:

(ARQ1) *Is the runtime performance of such a pipeline on consumer hardware suitable for human interactive use?*

(ARQ2) *How much would the answer accuracy of the pipeline improve if the used VLMs were more thoroughly pre-trained for understanding attributes (color, shape, etc.) and relations (holding, to the left of, etc.)?*

(ARQ3) *Making use of the compositionality of neural-symbolic VQA approaches, how can the visual perception component be modified/extended to improve the pipeline? In particular, how is the accuracy of the pipeline affected by (1) the explicit computation of spatial relations between objects, and (2) the integration of LLMs to judge the plausibility of object relations?*

1.1.1 Scope

With a sprawling number of datasets presenting many variations of the core VQA task, we concentrate our efforts on one recent and generalist VQA challenge, the GQA [HM19] dataset. Also, since the focus of this thesis lies on the visual perception and reasoning aspects of the VQA task, we omit the translation of the natural language question into a symbolic representation, directly using the so-called semantic question representation of GQA as a starting point (see Section 2.2.1 for an explanation of this representation).

1.2 Approach

To answer the research questions presented in the previous section, we design the GS-VQA neural-symbolic pipeline for zero-shot VQA with four components:

- Concept Extraction: Determines the information—object classes (car, person, etc.), attributes (color, shape, etc.) and relations (holding, to the left of, etc.)—required to answer the input question
- Scene Processing: Performs the visual perception, i.e., extracts the required information from the input image using VLMs
- ASP Encoding: Produces a symbolic representation of the input question and the extracted image information in the answer set programming (ASP) formalism
- ASP Solving: Uses rules defining the semantics of the available reasoning operations to solve the encoded ASP program and determine the answer to the input question

We implement this pipeline using current VLMs for object detection (OWL-ViT [MGS+22]) and general vision-language understanding (CLIP [RKH+21]), and the Potsdam Answer Set Solving Collection (Potassco) [GKK+19]. To permit the analysis of research questions ARQ2 and ARQ3, extensions of the base implementation of the pipeline are built. Finally, we thoroughly evaluate the accuracy and runtime performance of the pipeline and its extensions and put them into context with the results of state-of-the-art VQA approaches of various types (neural-symbolic, end-to-end, trained, zero-shot, etc.).

We find our approach suitable for representing the benefits and drawbacks of combining neural-symbolic VQA with VLMs by using the employed VLMs efficiently and effectively. Through the selective extraction of only the image information relevant to the input question, we are able to use current resource-intensive VLMs while keeping the pipeline

runtime in check. And with a non-deterministic ASP encoding of the extracted information, we are able to capture the uncertainty of the VLMs' predictions in the reasoning process. Together with the experiments and analysis for the auxiliary research questions 2 and 3, we can give an accurate picture of the feasibility of zero-shot neural-symbolic VQA with VLMs today, and the expected development in the near future.

1.3 Outline

In Chapter 2, we introduce the background knowledge required to follow the remainder of the thesis, which includes VQA, the GQA dataset, ASP, VLMs, and scene graph generation. For each of these topics, we also discuss the works from the literature most related to this thesis. Afterwards, in Chapter 3 we give a birds-eye view on the design of the GS-VQA pipeline and present its components, their responsibilities, and the flow of information between them with one continuous example. Chapter 4 then provides the full implementation details of the pipeline components introduced in the previous chapter, followed by the implementation details on the proposed pipeline extensions in Chapter 5. In Chapter 6 we discuss the details of our evaluation methodology, present the obtained results, and discuss their implications. Finally, in Chapter 7, we summarize our results, discuss the limitations of the current implementation of the GS-VQA pipeline, and provide an outlook into future enhancements.

CHAPTER 2

Background & Related Work

In this chapter, we give an introduction into several topics that are needed to understand the remainder of the thesis: visual question answering (VQA), the GQA [HM19] dataset, answer set programming (ASP), vision-language models (VLMs), and scene graph generation. For each topic, we also review the literature that relates most closely to the objectives of this work.

2.1 Visual Question Answering

VQA is a task combining computer vision (CV) and natural language processing (NLP) in which a model is provided with a natural-language question relating to an image, and must answer the question with the image as context (an example image and question pair is depicted in Figure 2.1). The task was proposed by Antol et al. in their equally named paper from 2015, in which they provide a benchmark dataset with roughly 250k images and 760k questions [AAL+15]. Due to the fact that VQA is a challenging multi-modal task that additionally requires a model to be able to perform (often multi-step) reasoning, since its introduction a large number of vastly different solution approaches have been explored, and various benchmark datasets with enhancements or alternations of the original task have been introduced. Apart from being a challenging study in reasoning and multi-modal machine learning, VQA has use-cases in the medical field, assistance systems for the visually impaired, video surveillance, education, and advertising [BBM+21].

2.1.1 Neural-Symbolic VQA

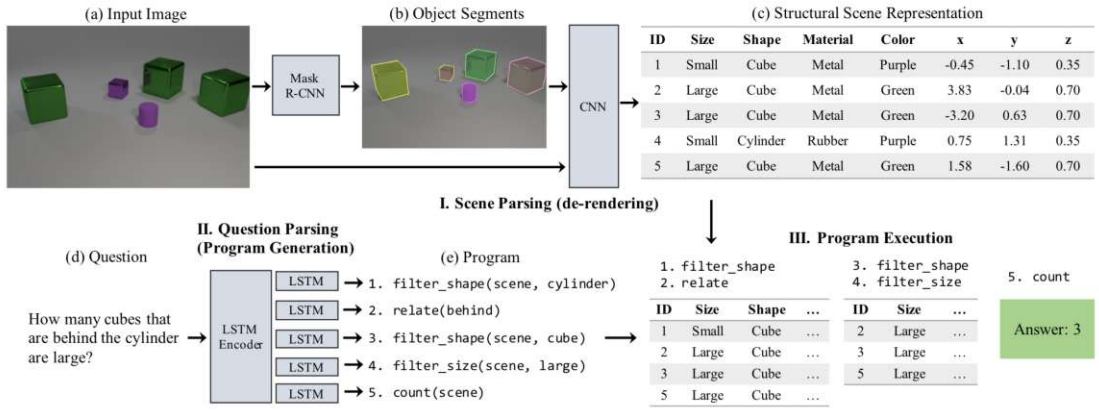
Neural-symbolic approaches for VQA mix neural networks for object detection, classification, and question parsing with symbolic evaluation for the reasoning process. In contrast to approaches based purely on multi-modal deep learning (which will be discussed in Section 2.1.2), the reasoning process is (to varying degrees) explainable: if the neural-symbolic pipeline arrives at the wrong answer, one can determine whether and how the

Figure 2.1: A VQA example demonstrating the requirements perception (red), reasoning (blue), and external information retrieval (orange) (Image Source: artbma.org)

pipeline misinterpreted the question or misclassified a certain entity in the image. Of this paradigm, two distinct flavours have emerged: one extracts a symbolic representation from both image and question and performs reasoning purely symbolically, while the other builds only a symbolic representation of the reasoning steps implied by the question and implements some or all of these reasoning steps with neural networks. Since our work falls into the former category, we consider it under the name "neural-symbolic VQA" and discuss it in this section, while we name the latter category "question-symbolic VQA" and discuss it in Section 2.1.2 afterwards.

The term "neural-symbolic VQA" was established by Yi et al. [YWG+18] in 2018 with a VQA pipeline of the same name. An illustration of the pipeline is given in Figure 2.2. Their NS-VQA pipeline extracts a "structural scene representation" (a list of all objects detected in the image—commonly also called scene—, together with their attributes and position) from the input image. It then translates the provided question into a structured representation of the reasoning steps needed to answer the question in the form of a functional program, and executes this program on the structural scene representation to obtain an answer. The authors show excellent results on the CLEVR [JHvdM+17a] dataset, a benchmark dataset with complex questions involving multiple reasoning steps about synthetic scenes generated in Blender[1]. However, the visually uncluttered scenes of CLEVR, intentionally crafted to simplify visual perception, hide a drawback of the approach of Yi et al.: its reasoning process is not able to deal well with imperfect detection and classification of objects in the scene. For example, given the question *"Which color does the object next to the small blue box have?"*, if the box in question is identified as large rather than small by NS-VQA's scene parser, the reasoning will not find an answer.

[1] https://www.blender.org/



ID	Size	Shape	Material	Color	x	y	z
1	Small	Cube	Metal	Purple	-0.45	-1.10	0.35
2	Large	Cube	Metal	Green	3.83	-0.04	0.70
3	Large	Cube	Metal	Green	-3.20	0.63	0.70
4	Small	Cylinder	Rubber	Purple	0.75	1.31	0.35
5	Large	Cube	Metal	Green	1.58	-1.60	0.70

ID	Size	Shape	...
1	Small	Cube	...
2	Large	Cube	...
3	Large	Cube	...
5	Large	Cube	...

ID	Size	...
2	Large	...
3	Large	...
5	Large	...

Figure 2.2: The original Neural-Symbolic VQA pipeline with symbolic question and image representations and a purely symbolic reasoning process (Source: [YWG+18, Figure 2])

Since then, Amizadeh et al. [APP+20] and Eiter et al. [EHO+22] have improved on NS-VQA's approach with logic-based reasoning processes, the former based on what the authors call "Differentiable First-Order Logic (∇-FOL)", the latter based on ASP, a formalism that is discussed in detail in Section 2.3. These reasoning processes are able to consider not just the most probable prediction of a scene object's class, attributes, and relations, but rather the entire vector of probabilities as output by the object detection and attribute/relation classifier networks that form the VQA pipeline's visual perception component. Continuing the example from above, these approaches would not simply view the detected box as "large", but, e.g., as "large" with a likelihood of 0.8, and "small" with a likelihood of 0.2. Together with all the other information gathered from the scene, they might therefore still conclude that the box is in fact the "small blue box" referenced by the question.

The GS-VQA pipeline presented in this thesis builds on this benefit of the ASP-based reasoning approach of Eiter et al., but improves on the approach in multiple ways. First and foremost, while the object detection in Eiter et al.'s pipeline requires training on images from the benchmark dataset (CLEVR in their case), GS-VQA is zero-shot: through the use of pre-trained VLMs (discussed in more detail in Section 2.4), no training is required to detect objects and extract their classes, attributes, and relations from the scene. Second, as Eiter et al. have shown that even for the small set of object classes of CLEVR, considering all options for each detected object in the ASP reasoning process can yield excessive runtimes, we implement a question-driven partial scene graph extraction method to allow the ASP reasoning approach to scale to the much larger space of object classes, attributes, and relations in datasets like GQA [HM19] (scene graphs are discussed in Section 2.5, while our approach to partial scene graph extraction is outlined in Section 3.3).

2.1.2 Other Approaches

Apart from neural-symbolic ones, a large variety of other approaches to VQA exist, which we categorize into end-to-end networks, question-symbolic approaches, and zero-shot approaches. Note that this overview should not be seen as a complete taxonomy: We refer to a survey by Zakari et al. [ZOW+22] for a more detailed categorization of the published approaches to VQA.

End-to-End Networks

The most straightforward approach to tackling the VQA task is to encode both image and question into feature vectors that are then used as input to some neural network that produces the answer to the question. In the literature dealing with this end-to-end machine learning approach, the image is most commonly encoded as output features of a convolutional neural network (CNN), though the features of objects as detected by a network like Faster R-CNN [RHG+15] are also widely used. The question is usually encoded by the hidden states of an uni- or bi-directional Long Short-Term Memory (LSTM) [HS97] network. The primary difference between models lies in the architecture of the final answer-generating network. The shape of the output also varies from multi-class classification of a known list of answer candidates, over binary classification of (question, image, answer)-triplets, to generation of sentence-long natural language answers, depending on personal choice of the researchers and the characteristics of the used benchmark dataset.

For the answer-generating network, early approaches either simply concatenate the image and question features or use some more elaborate form of combination like Multimodal Compact Bilinear pooling (MCB) [FPY+16], and then process these combined inputs with a fully-connected network [AAL+15] or some form of attention-based network [LYB+16; YHG+16].

Unfortunately, these early approaches have been shown to struggle with questions that have long reasoning chains or require short-term memory (for example, for an attribute comparison between two objects). Instead, they often exploit biases in the training data to derive their answers [ABP16]. This manifests itself in models ignoring large parts of the input question, not changing their answer across different input images, and experiencing a significant drop in performance when evaluated on a dataset that controls for biases [JHvdM+17a].

To improve the reasoning capabilities for VQA, multiple architectures have emerged that more explicitly integrate the multi-step nature of the reasoning process into their architecture. One such architecture, Memory, Attention, and Composition (MAC) [HM18], is built on a recurrent cell architecture where each MAC cell represents one reasoning step. Each of these steps updates a control state representing the reasoning action to perform with the question as context, selectively (based on control state and memory) extracts information from the image, and integrates this information with the cell's memory to

form a new partial result. The final cell memory is then plugged into a fully-connected classifier to determine the network's answer.

Another recent approach, which will be discussed in more detail in Section 2.4, is the direct use of VLMs for VQA. These models consist of multiple uni- and multi-modal Transformers [VSP+17] and use a variety of pre-training tasks on large amounts of data to gain strong multi-model understanding, which can then be used to perform a variety of vision-language tasks, VQA among them [TB19; LLX+22]. Through the multi-layered use of self- and cross-attention present in the Transformer architecture, VLMs are able to focus on different parts of the input at different times and thereby perform multi-step reasoning.

Question-Symbolic Approaches

Rather than extracting a symbolic representation from both the input question and the input image and reasoning on those, question-symbolic VQA approaches only extract the former, usually in the form of some programmatic specification of the reasoning steps needed to arrive at the question's answer.

Johnson et al. [JHvdM+17b] take inspiration from the concept of neural network modules [ARD+16], and use a functional program specification extracted from the input question to assemble a question-specific network from smaller modules. Each possible reasoning operation in the functional program has its own module, which consist of a small number of convolutional layers with input- and output-shapes designed to allow the modules to be chained together. The leaves of the assembled tree of modules accept the input image, while a classifier at the root produces the final answer.

Instead of performing all reasoning through composable neural network modules, ViperGPT [SMV23] and CodeVQA [SNK+23] "out-source" only some reasoning steps. Both use a large language model (LLM) to translate the input question into a valid Python program, which enables them to represent many reasoning operations like comparisons, counting, and negation, through their respective Python primitives. Operations relating to the input image (finding all objects of a certain class, checking if an attribute applies to an object in the scene, etc.) are delegated to VLMs (these models are discussed further in Section 2.4.3).

Zero-Shot Models

Most of the models discussed above require the training of at least some component in their architecture on a training dataset to achieve good results on the test dataset, and do not generalize well to unseen object classes, attributes, or relation types. This severely limits their usability, since they will likely encounter unseen concepts and thus under-perform in general-purpose use. Also, training data for special-purpose tasks takes a lot of effort to generate.

The exception are VLMs like BLIP-2 [LLS+23] and SimVLM [WYY+22] that have gained a sufficiently strong vision-language understanding through their pre-training regimes to generalize well to multiple different datasets, as well as approaches that use these VLMs as components, for example ViperGPT [SMV23], CodeVQA [SNK+23], and PnP-VQA [TLL+22].

2.1.3 Datasets

Since the release of the original VQA dataset [AAL+15], a large variety of alternative benchmark datasets have been introduced. These broadly fall into two categories: generalistic VQA datasets that seek to alleviate the original dataset's flaws and improve on its scope and challenge, and those that specialize on a certain "flavour" of VQA.

Generalist Datasets

The original VQA [AAL+15] dataset uses the MS COCO [LMB+14] dataset as an image source, which contains visually complex real-world photos with multiple objects. Questions are written by Amazon Mechanical Turk[2] workers, and for each question, 10 answers from unique workers are collected to capture the uncertainty ingrained in the task (for the color of an object, "white", "eggshell", and "off-white" might all be correct). Unfortunately, the dataset suffers from strong language biases that can be exploited by models to answer questions without properly attending to the input image, leading to an inflated sense of the amount of reasoning that these models are actually able to perform [ABP16; GKS+17]. These biases often arise from an imbalanced distribution of the image data (for example, 41% of questions starting with "What sport is" can be correctly answered with "tennis", implying that in the images depicting sport scenes in MS COCO, tennis is over-represented), or flaws in the question collection process (for example, blindly answering "yes" to questions starting with "Do you see a ..." yields 87% accuracy, since, if people are tasked to write questions about a given image, they tend to question about things present, rather than absent, in the image).

The second iteration of the VQA dataset (VQA v2.0 [GKS+17]) attempts to balance the original dataset by finding for each $(\text{image}, \text{question}, \text{answer})$ triple (I, Q, A) from the original dataset a "similar" image I' that leads to a different answer A'. As an indicator of the similarity of two images, the ℓ_2-distance between their embedding vectors as returned by a VGGNet [SZ15] CNN is used (i.e., for two embedding vectors $\vec{x}, \vec{y}$, $\ell_2 = \sqrt{\sum_i (x_i - y_i)^2}$). As a result, the dataset has roughly 1.1M $(\text{image}, \text{question})$ pairs, approximately twice the number of pairs in the original version. While this approach perfectly balances the answer distribution for binary question types, it leaves the distributions of open questions largely unbalanced [HM19].

By following a rigorously analytical approach, the CLEVR [JHvdM+17a] dataset takes avoiding question-conditional bias even further. Instead of using real-world images,

[2] https://www.mturk.com/

CLEVR renders simple artificial scenes of objects of 3 shapes, 2 sizes, 2 materials, and 8 colors from generated scene graphs (the concept of a scene graph is discussed in more detail in Section 2.5). For those scenes, questions are then constructed by instantiating 90 different question families, each of which consists of a functional program template that describes the steps of reasoning required to answer the question, and multiple text templates representing different ways of wording the question in natural language. Intelligent sampling ensures that each question family (i.e., questions sharing the same linguistic structure) has a roughly uniform answer distribution, and that ill-posed or degenerate questions are avoided. The composition of questions from functional reasoning operations enables the efficient construction of long and complex questions, and the analysis of a model's performance for specific modes of reasoning (spatial, counting, etc.).

While CLEVR provides a bias-controlled dataset with questions requiring multi-step reasoning, its challenge is somewhat artificial: question-answering skills demonstrated on the rendered scenes of CLEVR, with their visual simplicity, and their limited number of object attributes and relations between them, may not map reliably to "real-world" use-cases. By drawing on the scene graph annotated images of the Visual Genome project [KZG+17], the GQA [HM19] dataset improves on these deficiencies. GQA also instantiates question families represented by a functional program specification and multiple text templates, but the scenes on which they are instantiated, being annotated real-world images, contain a much richer variety of object classes, attributes, and relations. Also, the real-world images contain significantly more visual noise, which makes the detection of objects and attributes in the scene far less trivial. Since GQA is the dataset of choice for this thesis, it is discussed in more detail in Section 2.2.

Specialist Datasets

A number of datasets modify the general "reasoning about images" premise from the original VQA dataset and focus on particular modes of reasoning, visual or textual comprehension skills, or areas of application.

In addition to reasoning about the information contained in the input images themselves, questions in the Outside-Knowledge VQA (OK-VQA) [MRF+19] and Fact-Based VQA (FVQA) [WWS+18] datasets require a model to draw on "common-sense" knowledge to arrive at the correct answer. Models therefore need to be able to access external knowledge bases, or have factual knowledge ingrained through their training regimes.

Regarding comprehension skills, generalist datasets usually require models to detect objects (trees, cars, people, etc.) and their attributes, and to understand a natural-language question written in English. Models like TextVQA [SNS+19] and DocVQA [MKJ21] break with the first convention by requiring models to also detect and reason about text in the input image, while datasets with questions in a language with drastically different structure like Japanese [SRM18] and Chinese [GMZ+15] break with the second.

Finally, datasets for specific areas of application include ones for radiology images [LGB+18], and ones asking questions about data visualisations [KPC+18].

2.2 The GQA Dataset

As already alluded to in Section 2.1.3, a wide variety of VQA datasets have sprung up since the release of the "original" VQA dataset by Antol et al. [AAL+15], improving on issues of their predecessors or specialising into fragments of the generalist VQA task. For this thesis, we use the GQA [HM19] dataset, a state-of-the-art generalist dataset that has seen wide adoption in the recent literature [APP+20; SMV23; LNR+20; LLS+23]. The dataset contains over 22M open and binary questions, which are complex in structure, involve a wide variety of reasoning skills, and have a large number of possible answers (1,878 to be exact). A reduced set of 1.7M questions is provided that controls for question-conditional biases in both open and binary questions, and thus presents an ideal basis for comparison of zero-shot and trained VQA pipelines, since trained models can exploit those biases to a far lesser degree than in previous datasets [AAL+15; GKS+17]. The questions cover more than 100,000 images from the Visual Genome [KZG+17] dataset that present real-world scenes with a wide variety of object classes, attributes, and relations. Unlike the synthetic and overly simplistic images used in datasets like CLEVR [JHvdM+17a], these natural images provide a realistic testing ground for the capabilities of zero-shot VLMs in the context of VQA.

Apart from these general beneficial properties of the dataset, GQA comes with two types of supplementary data that greatly aid in the development of the GS-VQA pipeline: First, each natural-language question from the test/validation/test-dev splits comes with a structured representation of its required reasoning steps, referred to as a "semantic representation" by the authors. This enables us to focus on the visual perception and reasoning aspects of the VQA task and leave the zero-shot translation of natural language into structured specifications to future research (look at OpenAI Codex [CTJ+21] for a recent example of LLMs for code generation and at ViperGPT [SMV23] for an application of the model to VQA). Second, a Visual Genome scene graph is provided for every image in the test/validation splits of the dataset, which allows us to verify soundness of our ASP encoding under perfect visual information during development, as has been done by Amizadeh et al. [APP+20] for their ∇-FOL. While we discuss the semantic question representation below, scene graphs and their generation are described in more detail in Section 2.5.

2.2.1 Semantic Question Representation

While in principle natural-language questions could be directly translated into an ASP encoding, we exclude this translation from the focus of this thesis. Instead, we use the already structured semantic question representation of GQA as a pipeline input and translate it into the ASP Question Encoding. For this reason, we give here a brief overview over the format of this semantic question representation, illustrated the example question: "Do the umpire and the person holding the green baseball bat have the same pose?". The semantic representation of this question is depicted textually and graphically in Figure 2.3.

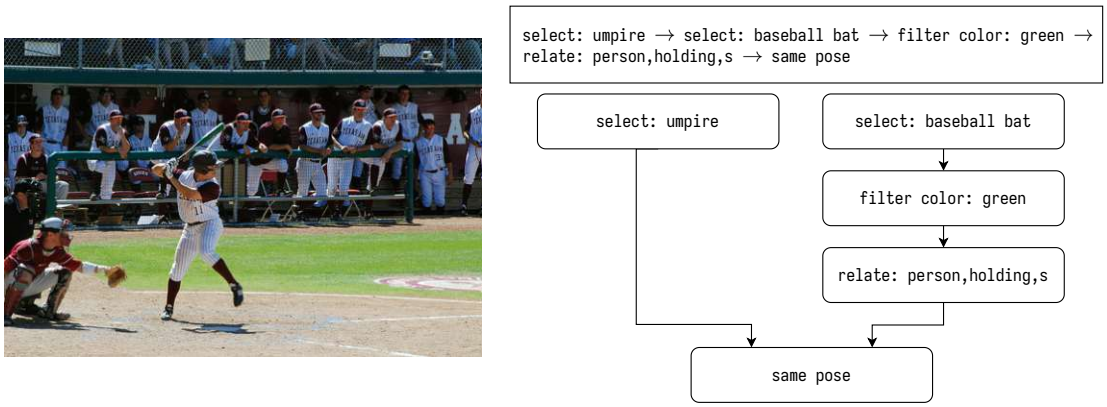


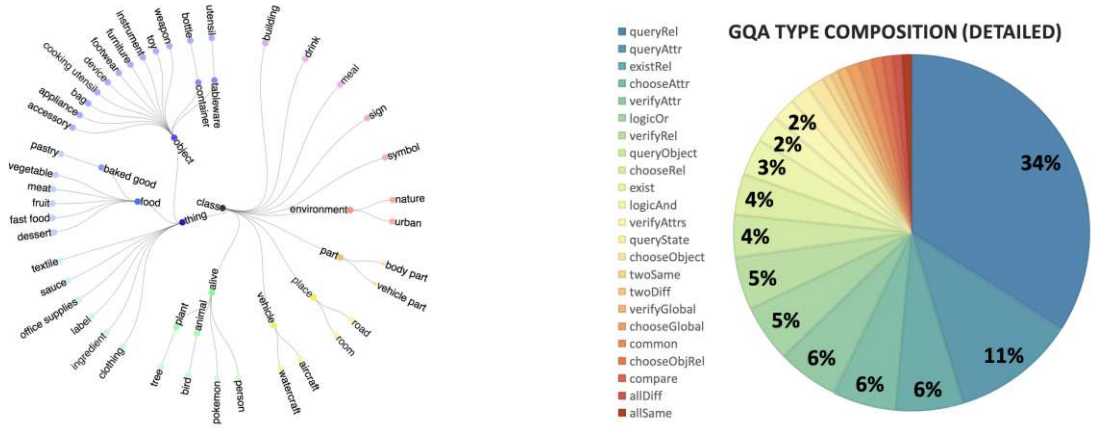
Figure 2.3: The GQA semantic representation for the question "Do the umpire and the person holding the green baseball bat have the same pose?"

From this example, it is apparent that questions reference three kinds of concepts: object classes (helmet, umpire, person), attributes (pose, standing), and relations (wearing, to the left of). Note that classes form an ontological hierarchy (e.g., an umpire is also a person), and that we distinguish attribute categories (e.g., pose, color) from their concrete attribute values (e.g., standing, red).

The semantic representation itself can be seen as a pre-order traversal of a tree of reasoning operations. Most of these operations, such as `select`, `filter`, and `relate`, take as input a set of objects (or all objects in the scene, if no operation precedes them) and return another set of objects. Terminal operations such as `query`, `choose`, and `same` take as input one or multiple sets of objects and return a concrete value, which might be an attribute value, a relation, or a boolean. Some of these terminal operations implicitly expect only a single object as input. The complete set of operations will be discussed in more detail in Section 4.4.1.

2.2.2 Variety & Structure

Compared to its spiritual successor CLEVR [JHvdM+17a], GQA's scenes contain a vastly increased range of object classes, attributes, and relation types: Whereas CLEVR has 3 object classes (cubes, spheres, and cylinders), objects in GQA are part of 1,740 classes (camera, burger, etc.), which are themselves organised into a hierarchy of 60 categories, or higher-order classes (device, food, etc.). The full range of categories can be seen in Figure 2.4a. Instead of CLEVR's 12 different attribute values grouped into 3 attribute categories (size, color, and material), GQA objects have 620 different attribute values grouped into more than 30 attribute categories (color, tone, cleanliness, texture, etc.). Finally, rather than having only four elementary spatial relations (left, right, in front, behind), GQA contains 330 different relations, both spatial ones (near, next to, etc.) and semantic ones (wearing, holding, etc.).



(a) (b)

Figure 2.4: **(a)** The hierarchy of GQA's 60 object categories **(b)** The distribution of GQA's questions across 23 question types (Source: [HM19, Supp. Material, Figure 1&2])

The questions in GQA are instantiated from 117 template question groups categorized into 23 types, each group containing a semantic representation, a set of natural-language rephrases that express those same semantics, and a pair of long and short answers. Figure 2.4b shows the distribution of the instantiated question across the 23 question types.

2.2.3 Other Works evaluated on GQA

Due to the popularity of GQA, a number of recent papers spanning a wide range of approaches to VQA have evaluated their work on the dataset. These include other neural-symbolic approaches [APP+20], question-symbolic approaches [LNR+20; SMV23; SNK+23], end-to-end networks [HM18; TB19; NDT+22], and VLMs [JCS+22; LLS+23]. Their performance will be used as a basis for comparison in Chapter 6.

2.3 Answer Set Programming

ASP is a declarative, non-monotonic logic programming (LP) formalism. The idea of a declarative problem solving approach is succinctly described by Gebser et al. as follows:

> *"Rather than solving a problem by telling a computer* how *to solve the problem, the idea is simply to describe* what *the problem is and leave its solution to the computer."* [GKK+12]

A problem description in ASP is a *program*, a finite collection of *rules* that, in their most basic, propositional variant, have the form [BET11]:

$$a :- b_1, ..., b_m, not\ c_1, ..., not\ c_n. \tag{2.1}$$

Here, a, b_i, c_j are *atoms*, elementary propositions that may be true or false, while $not\, c_j$ denotes the *weak negation* of atom c_j. Atoms and their weak negations form the set of *literals*. a is also called the *head* of rule r (denoted $head(r)$), while the literals to the right of :– represent the *body* of the rule ($body(r)$, split into $body^+(r)$ and $body^-(r)$ for the positive and negative (weak-negated) literals).

A rule r is the justification to derive the truth of $head(r)$ if all literals in $body(r)$ can be derived as true. A weak negation $not\, c_j$ is true if c_j cannot be derived from the rules of the program (hence weak negation is often also referred to as "negation as failure"). A rule without a body (i.e., of the form d :– .) is called a *fact*, and communicates that d can always be derived.

A solution to an ASP program $\mathcal{P}$, as the name would already suggest, is called an *answer set*. Let S be an *interpretation*, i.e., a subset of the atoms of $\mathcal{P}$. If $\mathcal{P}$ does not contain weak negation (i.e., for all rules $r \in \mathcal{P}$, $body^-(r) = \emptyset$), then S is the answer set of $\mathcal{P}$ if S is the minimal closed set under $\mathcal{P}$. That is, S is the minimal set such that:

$$\forall r \in \mathcal{P}: \quad body^+(r) \in S \Rightarrow head(r) \in S \tag{2.2}$$

Intuitively, S contains all atoms that can be derived from the information in program $\mathcal{P}$. If $\mathcal{P}$ does contain weak negation though, this bottom-up construction of the answer set does not work, since we do not know which atoms can be eventually derived, and thus not verify the conditions for applying any of the rules containing weak negation [BET11]. Instead, a certain interpretation S is assumed (which places an assumption on which atoms can, and importantly also cannot, be derived). Under this assumption, all rules that contain $not\, c_j$ for some $c_j \in S$ cannot be invoked, and can thus be removed from $\mathcal{P}$. For all remaining rules with some weak negation $not\, c_j$, the negated atom c_j must not be in S, since otherwise the rule would have been discarded in the previous step. It can therefore be safely assumed that $not\, c_j$ evaluates to true, and the literal $not\, c_j$ can be removed from the rule without affecting its usability. The resulting program without weak negation is called the Gelfond-Lifschitz reduct $\mathcal{P}^S$ of $\mathcal{P}$.

An interpretation S is then the answer set of $\mathcal{P}$ if it is the minimal closed set under the Gelfond-Lifschitz reduct $\mathcal{P}^S$. Intuitively, S containing all atoms derivable from the information in $\mathcal{P}^S$ verifies the assumptions in S about which atom can/cannot be derived.

It is important to realise that the answer sets of an ASP program behave non-monotonically, that is, previously drawn conclusions may have to be changed as new facts or rules are added to the program. This is a direct consequence of weak negation, which becomes apparent with the following program as a minimal example:

$$a :- b, not\, c. \tag{2.3}$$
$$b. \tag{2.4}$$

This program trivially has the answer set $\{a, b\}$. However, if the fact $c.$ is added to the program, the rule used to establish the truth of a can no longer be applied, so this previously established conclusion has to be retracted, leaving $\{b, c\}$ as the new answer set.

2.3.1 Programs with Predicates

A crucial extension of the just introduced formalism is to allow for predicates rather than just atoms. In this variant of ASP, rules have the form:

$$A :- B_1, ..., B_m, not\, C_1, ..., not\, C_n. \tag{2.5}$$

where A, B_i, C_j are atomic formulas in the language [BET11], meaning they take the form $R(s_1, ..., s_k)$, where R is a predicate of arity k, and each s is either a constant (often denoted by $a, b, c, ...$) or a variable (often written as $X, Y, Z, ...$). Answer sets of such a program $\mathcal{P}$ with predicates are then defined in terms of its grounding $grnd(\mathcal{P})$ over the Herbrand universe (i.e., the program obtained by replacing each rule with variables with all its instantiations over the constants). This then leads to the ASP solving process depicted in Figure 2.5.

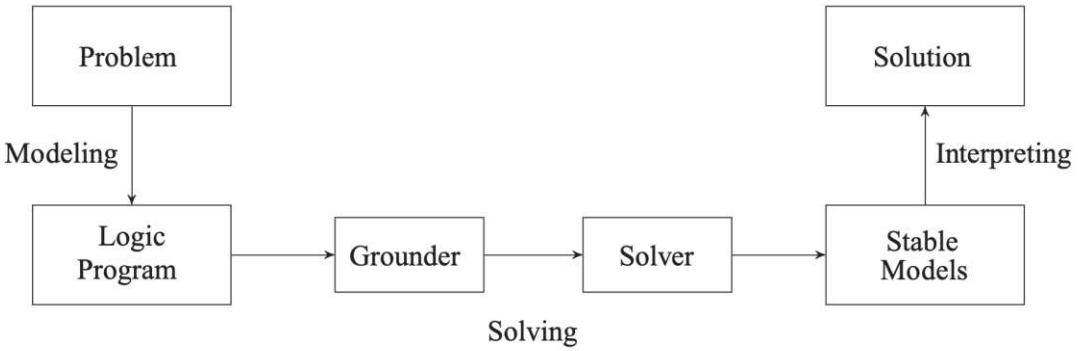


Figure 2.5: The ASP solving process with predicates (Source: [GKK+12, p. 3])

In practice, ASP solvers do not replace $\mathcal{P}$ by $grnd(\mathcal{P})$, since this generally leads to an exponential blow-up of the program size. Rather, they employ various optimization techniques to produce *"a possibly small propositional program, not necessarily a subset of $grnd(\mathcal{P})$, that is equivalent to $\mathcal{P}$, that is, has the same answer sets"* [BET11].

To then solve the propositional grounding of $\mathcal{P}$, ASP solvers commonly employ techniques introduced in SAT solving (backtracking search, clause learning, etc.), but extended by the additional considerations implied by the foundedness condition of answer sets (i.e., that every atom that is true must be derived by a rule in the program).

2.3.2 Extensions

Depending on the chosen solver, various extensions to the ASP formalism are available. In the following, only those that are most relevant to this thesis will be introduced. A precise explanation of the semantics and the translation of these constructs into simpler ASP variants are provided by Gebser et al. [GKK+12; GKK+19], and Leone et al. [LPF+06].

Choice & Cardinality Rules

Falling back to the propositional variant of ASP for simplicity, a choice rule has the form:

$$\{a_1, a_2, ..., a_o\} :- b_1, ..., b_m, not\, c_1, ..., not\, c_n. \tag{2.6}$$

This rule expresses that any subset of $\{a_1, a_2, ..., a_o\}$ may be included in an answer set, provided the body literals are true. Cardinality rules are a natural extension of this concept, stipulating that at least l and at most k of the atoms in the rule head must be included in an answer set if the rule body is satisfied:

$$l\ \{a_1, a_2, ..., a_o\}\ k :- b_1, ..., b_m, not\, c_1, ..., not\, c_n. \tag{2.7}$$

Integrity Constraints

For some problems, it may be desirable to enforce that a scenario does not occur. This can be achieved with integrity constraints, which differ from regular rules simply by their lack of a rule head:

$$:- b_1, ..., b_m, not\, c_1, ..., not\, c_n. \tag{2.8}$$

A constraint like that of Equation 2.8 enforces that all interpretations that satisfy its rule body are ruled out as answer sets.

Weak Constraints & Optimization

In optimization, the goal is not simply to find any (or all) interpretations S that are answer sets, but to find those that are *optimal*, with respect to the weak constraints of the program. In contrast to the strong integrity constraints just presented, weak constraints may be violated by an answer set, but this violation contributes a term tuple $(t_1, t_2, ..., t_o)$ with an associated weight w to a cost function:

$$:\sim b_1, ..., b_m, not\, c_1, ..., not\, c_n.\ \ [w, (t_1, t_2, ..., t_o)] \tag{2.9}$$

An answer set is then optimal if the sum of weights of all contributed tuples is minimal (note that set semantics apply, i.e., if the same tuple is contributed twice, its weight is counted only once).

2.3.3 Other Works using ASP for VQA

Using logic-based formalisms for neural-symbolic VQA is still a relatively unexplored area of research. As such, only two other approaches have been proposed: most closely related, Eiter et al. [EHO+22] use ASP to tackle the CLEVR dataset, while Amizadeh et al. [APP+20] perform reasoning in what the authors call "Differentiable First-Order Logic (∇-FOL)" to answer the questions of GQA. Looking beyond the VQA task, other neural-symbolic formalisms integrating neural network output into ASP [YIL20] or ProbLog [MDK+18] have also been introduced.

2.4 Vision-Language Models

In the field of natural language processing (NLP), the training of models on vast amounts of raw text with so-called pre-train objectives that are agnostic to any specific downstream task has enabled immense improvements in language understanding [RKH+21]. This development has reached a point where LLMs perform competitively to or even better than purpose-built models when applied with minimal fine-tuning [DCL+19] or even zero-shot [BMR+20] to a variety of downstream tasks (translation, named entity recognition, causal language modeling, text classification, question answering, etc.).

VLMs, as their name would suggest, adapt the approach of task-agnostic pre-training on large quantities of data to (image, text) pairs, enabling multi-modal understanding that can again be applied to a wide range of tasks (open-vocabulary image classification, object detection, image-text matching (ITM), image captioning, VQA, etc.). Among the state-of-the-art models in this space are CLIP [RKH+21], SimVLM [WYY+22], and BLIP(-2) [LLX+22; LLS+23].

In the following sub-sections, we discuss the two VLM tasks that are used by the GS-VQA pipeline (ITM, open-vocabulary object detection), along with the models we employ for them (CLIP, OWL-ViT [MGS+22]).

2.4.1 CLIP for Open-Vocabulary Image Classification

CLIP [RKH+21] consists of an image and a text encoder that encode their respective inputs into a joint embedding space (which is 512- to 1024-dimensional, depending on the implementation). The image encoder either follows the ResNet [HZR+16] architecture or that of the Vision Transformer (ViT) [DBK+21], while the text encoder is a Transformer [VSP+17].

To train CLIP, Radford et al. used only one contrastive pre-training objective: When presented with a batch of n (image, text) pairs from the training data, CLIP is tasked to determine the correct n pairings of images and texts from the n^2 possible ones of the batch. To pair an image to a text (and vice-versa), it encodes the images and texts from the batch into their shared embedding space and then determines for each image/text embedding the respective text/image embedding that is "closest" in the embedding space

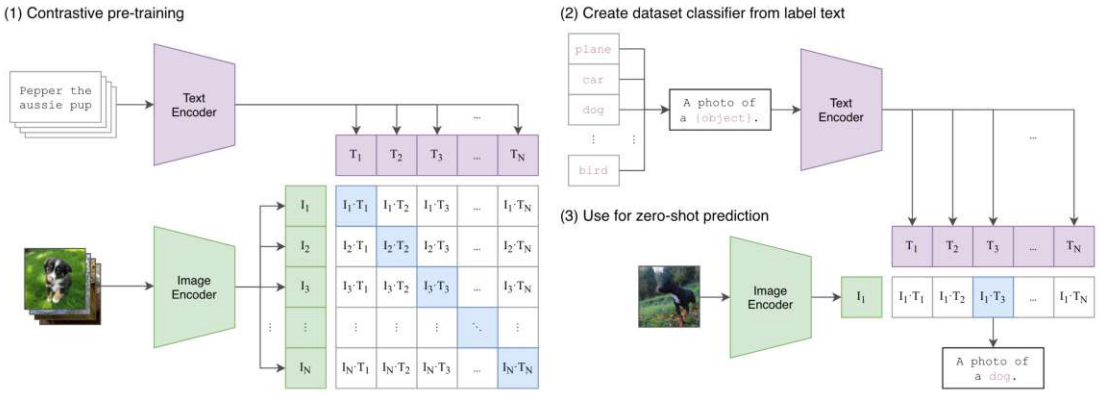


Figure 2.6: **(1)** A summary of CLIP's architecture and pre-training approach **(2-3)** Using CLIP for open-vocabulary image classification (Source: [RKH+21, Figure 1])

by cosine similarity. Pre-training then optimizes a symmetric (i.e. considering both pairing of text to image and the other way around) cross-entropy loss on these similarities. The process is illustrated in step (1) of Figure 2.6.

This pre-training procedure produces the desirable property that semantically similar images and texts, i.e., images that show the same thing that the texts describe, have a high cosine similarity in CLIP's embedding space, while dissimilar images and texts, i.e., those that depict and describe different concepts, have a comparatively low cosine similarity.

To utilize this property for open-vocabulary image classification (i.e., classification in which the classes are not hard-wired into the model architecture), one can compute the cosine similarity between an image and a textual prompt for each class of interest, e.g., "a photo of a `{class}`", where `{class}` is instantiated to "plane", "car", "dog", etc., and pick the class which maximizes this similarity. This approach is already described by Radford et al. [RKH+21] and shown in steps (2-3) of Figure 2.6.

2.4.2 OWL-ViT for Open-Vocabulary Object Detection

Vision Transformer for Open-World Localization (OWL-ViT) [MGS+22] is both a model in its own right and recipe for adapting VLMs trained with image-text contrastive pre-training (like CLIP) to the task of open-vocabulary object detection.

The architecture of OWL-ViT is depicted in Figure 2.7. It has as its basis the text and image encoders of a contrastively pre-trained VLM, though with the restriction that the image encoder must be based on the Vision Transformer to allow for the modifications that follow. A ViT splits its input image into a sequence of small (originally 16×16 pixel) patches, runs them through a linear projection, and concatenates the resulting patch embeddings with a learnable position embedding. This sequence of embedding vectors is used as input to a Transformer encoder, which produces a sequence of output

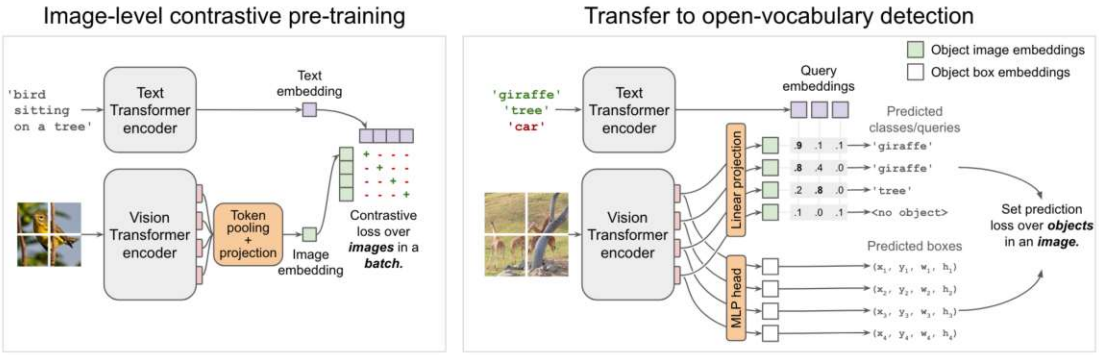


Figure 2.7: OWL-ViT's approach of adapting the text and image encoders of a contrastively pre-trained VLM (left) to perform open-vocabulary object detection (right) (Source: [MGS+22, Figure 1])

vectors of the same length. For the encoder in contrastively trained VLMs like CLIP, a final pooling and projection layer is then used to combine these output vectors into an embedding for the entire image.

OWL-ViT removes these final pooling and projection layers, and instead transforms the sequence of output vectors from the ViT encoder in two ways: First, each vector is projected into the embedding space shared with the text encoder. This enables OWL-ViT to determine the similarity of each ViT output vector to the embedding of one or multiple textual descriptions of object classes ("cat", "dog", "car", etc.), using cosine similarity as CLIP does for image-level classification. Second, each vector is projected into a bounding box. The maximum number of detected objects is thus limited by the sequence length of the ViT (which is not a problem in practice, since 224×224 images split into 16×16 patches already yield a sequence length of 196). An object is detected when the maximum (softmax-normalized) cosine similarity between it and one of the object classes exceeds a user-specified threshold.

2.4.3 Other Works using VLMs for VQA

Instead of CLIP's contrastive pre-training objective, many VLMs use objectives that require text generation: both BLIP [LLX+22] and SimVLM [WYY+22] train with some form of Language Modeling (LM) objective that maximizes the likelihood of an image-conditioned Transformer decoder generating the corresponding image caption autoregressively. Such VLMs can, sometimes with minor architectural adjustments, be used directly for VQA.

This capability is built upon by ViperGPT [SMV23], which uses an LLM to transform a natural-language question into Python code that adheres to a provided API. Some of the API's functions are implemented using VLMs like X-VLM [ZZL22] and BLIP-2 [LLS+23] for object detection, classification, and VQA with sub-questions of the original

question (using an example from ViperGPT's website[3], the question "What color is the counter?" is translated into code that first uses object detection to find the bounding box of a counter in the image, which is then presented to BLIP as an image crop with the question "What color is this?").

PnP-VQA [TLL+22] also uses LLMs and VLMs, though in a different way: in a multi-step process, it employs BLIP and a modified variant of GradCAM [SCD+17] to sample patches of the input image that are most relevant to the question, and generate textual captions for those patches. These captions, along with the question, are then used as input to an LLM that generates the answer.

Combining the approaches of ViperGPT and PnP-VQA, CodeVQA [SNK+23] also uses an LLM to transform a question into Python code (though with a much more concise API than ViperGPT), but employs the technique of PnP-VQA to implement its API's function for answering sub-questions.

2.5 Scene Graph Generation

A lot of research in visual perception has focused on the *objects* in an image, be it the classification of an entire image according to its principal depicted object [DDS+09], or the detection of objects and their location in the scene, either as bounding boxes [RHG+15; RF18] or more detailed segmentation masks [RFB15; BKC17]. In recent years, foundation models have been introduced that show strong zero-shot performance on these tasks [JHvdM+17a; MGS+22; KMR+23].

However, for the purpose of neural-symbolic VQA, information about the class and position of the objects in the scene is not sufficient. Rather, as we have seen in Section 2.2, questions in state-of-the-art VQA datasets like GQA [HM19] additionally require an understanding of the attributes of objects (a "*green* baseball bat") and the relations between them (a "person *holding* a baseball bat"). Scene graphs, originally introduced under this term in 2015 by Johnson et al. [JKS+15] for the purpose of image retrieval, model the object classes, attributes, and relations depicted in an image as a directed, sparse graph. Formally, Johnson et al. define a scene graph as follows: *"Given a set of object classes $\mathcal{C}$, a set of attribute types $\mathcal{A}$, and a set of relationship types $\mathcal{R}$, we define a scene graph G to be a tuple $G = (O, E)$ where $O = \{o_1, ..., o_n\}$ is a set of objects and $E \subseteq O \times \mathcal{R} \times O$ is a set of edges. Each object has the form $o_i = (c_i, A_i)$ where $c_i \in \mathcal{C}$ is the class of the object and $A_i \in \mathcal{A}$ are the attributes of the object."* [JKS+15]. An exemplary scene graph is shown in Figure 2.8.

Since the introduction of the scene graph concept, benchmark datasets with scene-graph-annotated images and models for scene graph generation (SGG) have been developed. Of the datasets, Visual Genome [KZG+17] is of particular note for its scope (108,077 images annotated with more than 3.8M objects, more than 2.8M attributes, and more than 2.3M

[3] `https://viper.cs.columbia.edu/`, accessed 08.07.2023

relations) and variety (33k different object categories, 68k unique attributes, 42k unique relations). It also forms the basis for the visual components of the GQA [HM19] dataset.

Models for SGG mostly focus on relation prediction, using already established approaches for object detection and often disregarding object attributes entirely [XZC+17; LKB+16]. Lu et al. [LKB+16] use R-CNN to generate object proposals, for which they then use an object and a relation classification CNN to determine the visual likelihood of (subject, relation, object) triplets applying to pairs of the proposed objects. A language module is then used to alter these likelihoods based on semantic plausibility (e.g., dampening the likelihood of (dog, drive, car) because it is unlikely to occur). Yang et al. [YLL+18] also use the idea of semantic plausibility, first constructing a full graph of all objects detected by Faster R-CNN [RHG+15], then pruning the edges between objects that are unlikely to be related. This sparse graph is processed by an attentional graph convolution network (aGCN) to obtain the final scene graph. Xu et al. [XZC+17] too extract object proposals from the image via Faster R-CNN, but process them using recurrent neural networks (RNNs). Additionally, their model learns to iteratively improve its predictions via message passing.

While—to the best of our knowledge—no foundation model for zero-shot SGG exists yet, Kan et al. [KCY21] have recently explored integrating commonsense knowledge contained in an external knowledge base to improve performance on unseen (subject, relation, object) triplets.

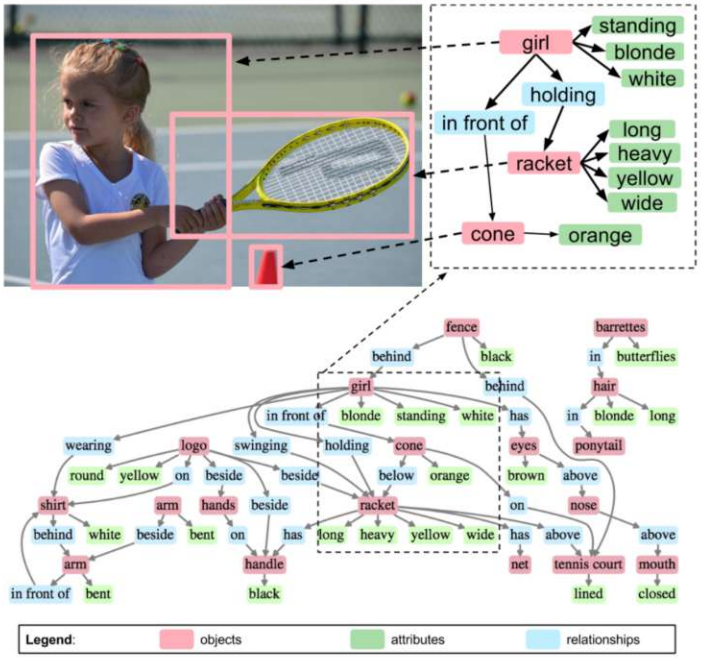


Figure 2.8: An example of a scene graph (Source: [JKS+15, Figure 2])

2.5.1 Other Works using SGG for VQA

While we are not aware of any VQA model that generates and uses a symbolic scene graph as generated by the models discussed above, multiple models use intermediary representations of the outputs eventually produced by SGG. Liang et al.'s LRTA [LNR+20] and Amizadeh et al.'s pipeline [APP+20] both run existing object detection models on the input image and then determine attributes and relations with classifiers taking single objects or object pairs as input. LRTA directly uses the vector outputs of its detection and classification models as input to its neural execution engine based on graph convolution, and only converts them to a symbolic form for human readability. Amizadeh et al. integrate the scene graph information as "visual oracles" into their ∇-FOL formalism, which return the likelihood of a predicate representing class-, attribute-, or relation-membership applying to an object or a pair of objects.

Related are also the structural scene representations extracted from an input image by Yi et al. [YWG+18] and Eiter et al. [EHO+22]. These too contain the objects in the scene, their attributes, and their relations among each other. However, since objects in the CLEVR [JHvdM+17a] dataset used by these papers have a small number of classes and attributes, and are related only via fundamental spatial relations ("left", "in front of", etc.), the papers can mostly rely on object detection by encoding all attribute combinations as classes and determining the relations from the object's bounding boxes.

CHAPTER 3

Design: The GS-VQA Pipeline

This chapter gives a birds-eye view on the architecture of the GS-VQA pipeline for zero-shot VQA. The pipeline is depicted in Figure 3.1 with a concrete example: a 500×334 image showing a scene of a baseball game, and the question "Is the umpire to the right or to the left of the standing person that is wearing a helmet?". In the following sections, the four components of the pipeline—Concept Extraction, Scene Processing, ASP Encoding, and ASP Solving, by order of execution—will be introduced using this example. The full implementation details of each component will then be elaborated in Chapter 4.

3.1 Pipeline Inputs

As expected for the VQA task, the inputs to the pipeline are an image and a question that is to be answered with the image as context. We focus on the visual perception and reasoning aspects of the VQA task and thus directly use the semantic question representation of GQA (introduced in Section 2.2.1) as an input to the GS-VQA pipeline. It is obvious that real-world use-cases or even other VQA benchmark datasets would not come with such a semantic representation for each question, in which case various trained models [JHvdM+17b; HAR+17; YWG+18] or zero-shot models [SMV23] could be used to generate it.

Looking at the example from Figure 3.1, the pipeline inputs are the depicted image, along with the following semantic representation: `select: helmet → relate: person,wearing,s → filter pose: standing → choose rel: umpire, to the left of|to the right of,s`.

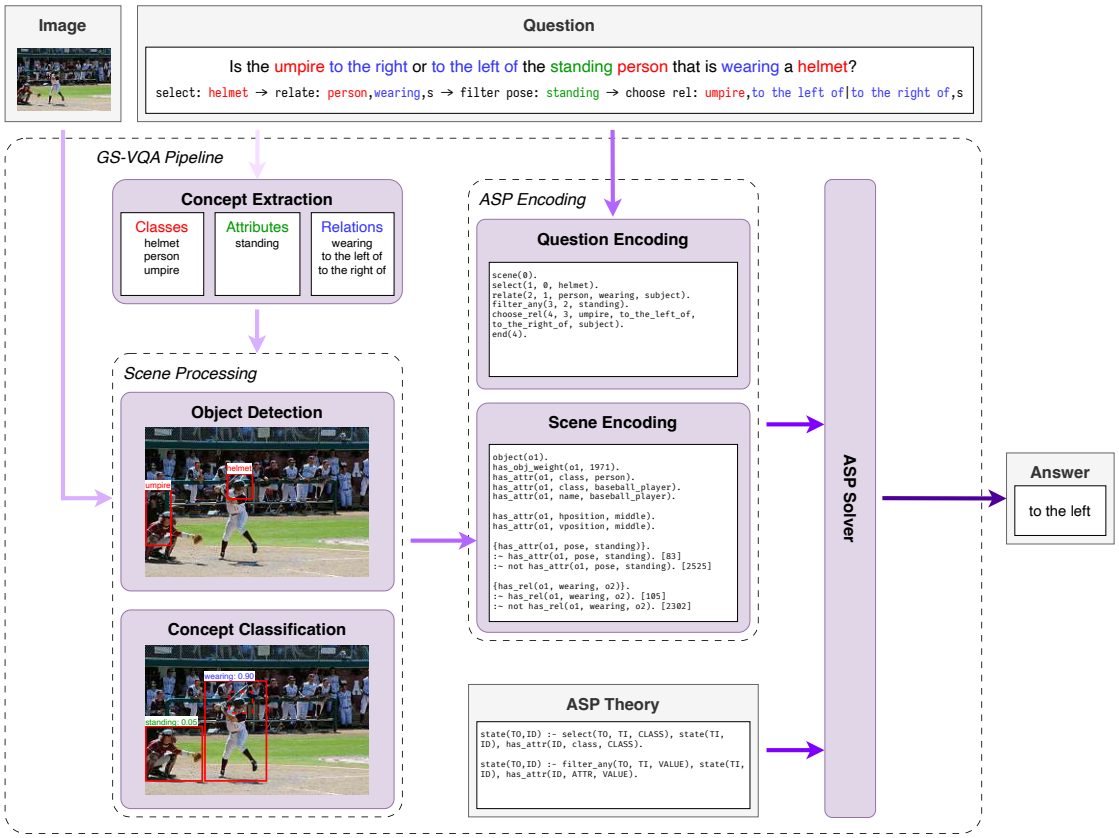


Figure 3.1: An overview over the full GS-VQA pipeline

3.2 Concept Extraction

In keeping with the paradigm of neural-symbolic VQA, the GS-VQA pipeline encodes both the input question and the input image symbolically. Encoding the relevant information contained in the input scene requires extracting it from that scene in the first place though. The obvious solution to this problem is SGG, variations of which have already been used by multiple neural-symbolic models (see Section 2.5.1 for further details).

However, in contrast to the works discussed in Section 2.5.1, the zero-shot nature of the GS-VQA pipeline introduces a problem for constructing a complete scene graph of visually complex (or, less euphemistically, cluttered) scenes: due to their general-purpose nature, inference with the VLMs that GS-VQA uses for SGG is far more resource-intensive than with the purpose-built trained models used by, e.g., Yi et al.'s NS-VQA [YWG+18] or Amizadeh et al.'s pipeline [APP+20]. As such, constructing a full scene graph in which likelihoods for all possible classes, attributes, and relations in the dataset are present for every detected object in the scene is untenable if we want to maintain an inference time that a human user might find acceptable.

To resolve this issue, the GS-VQA pipeline resorts to question-driven partial scene graph extraction, extracting only the information from the scene that is relevant for answering the question at hand. Determining which object classes, attributes, and relations are relevant from the semantic representation of the input question is precisely the role of the Concept Extraction component. Conceptually, the component outputs a tuple (C, A, R), where C is a set of classes, A a set of attribute categories, and R a set of relations (the actual format is a bit more intricate to cover certain edge cases and will be described in Section 4.2). For the example question from Figure 3.1, the tuple is `({helmet, umpire, person}, {pose}, {wearing, to the left of, to the right of})`.

3.3 Scene Processing

Using the (C, A, R) output tuple from the Concept Extraction component, the Scene Processing component (shown in Figure 3.2) has the task of extracting a question-driven partial scene graph. We deviate here a bit from the formal scene graph representation introduced in Section 2.5, and represent the graph as a list O of objects $o_i = (id, s, B, c, A_o, R_o)$, each having a unique identifier id, a score s between 0 and 1 denoting the Scene Processing's confidence in the object detection, a bounding box B, and a class c that is either in C or a sub-class of maximal specificity of a class in C. This last property ensures that each object cannot, for example, be detected as just a "person", but must be detected as a maximally specific class like "baseball player". It also means that the scene graph contains only objects of classes that are deemed relevant to answering the question (hence the description as a "partial" scene graph). In addition to this information, for each possible value of each attribute category in A, the objects include a likelihood between 0 and 1 of that attribute value applying to the object (attribute likelihoods A_o). Finally, for each relation $r \in R$ and each other detected object o_j, o_i includes a likelihood between 0 and 1 that r applies with o_i as the subject and o_j as the object (relation likelihoods R_o).

Illustrating this with the example from Figure 3.1, the Scene Processing might produce a list $O = [o_1, o_2, ..., o_n]$ with one object o_1 being the striking player. Its example scene graph entry is shown in Listing 3.1.

The Scene Processing component consists of two sub-components that both make use of VLMs and run one after the other to build the output described above. First, the Object Detection sub-component uses OWL-ViT [MGS+22] to detect the objects in O with their scores s, bounding boxes B, and classes c, and assigns each of them a unique id. The Concept Classification sub-component then takes this list of detected objects and uses CLIP [RKH+21] to supplement the attribute and relation likelihoods A_o, R_o.

```
(
   "o1",
   0.1392,
   {"x": 150, "y": 80, "w": 110, "h": 210},
   "baseball player",
   {"pose": {"standing": 0.92, "sitting": 0.003, ...}},
   {
      "o2": {
         "wearing": 0.900,
         "to the left of": 0.523,
         "to the right of": 0.542
      },
      ...
   }
)
```

Listing 3.1: An example scene graph entry for the striking baseball player from Figure 3.1

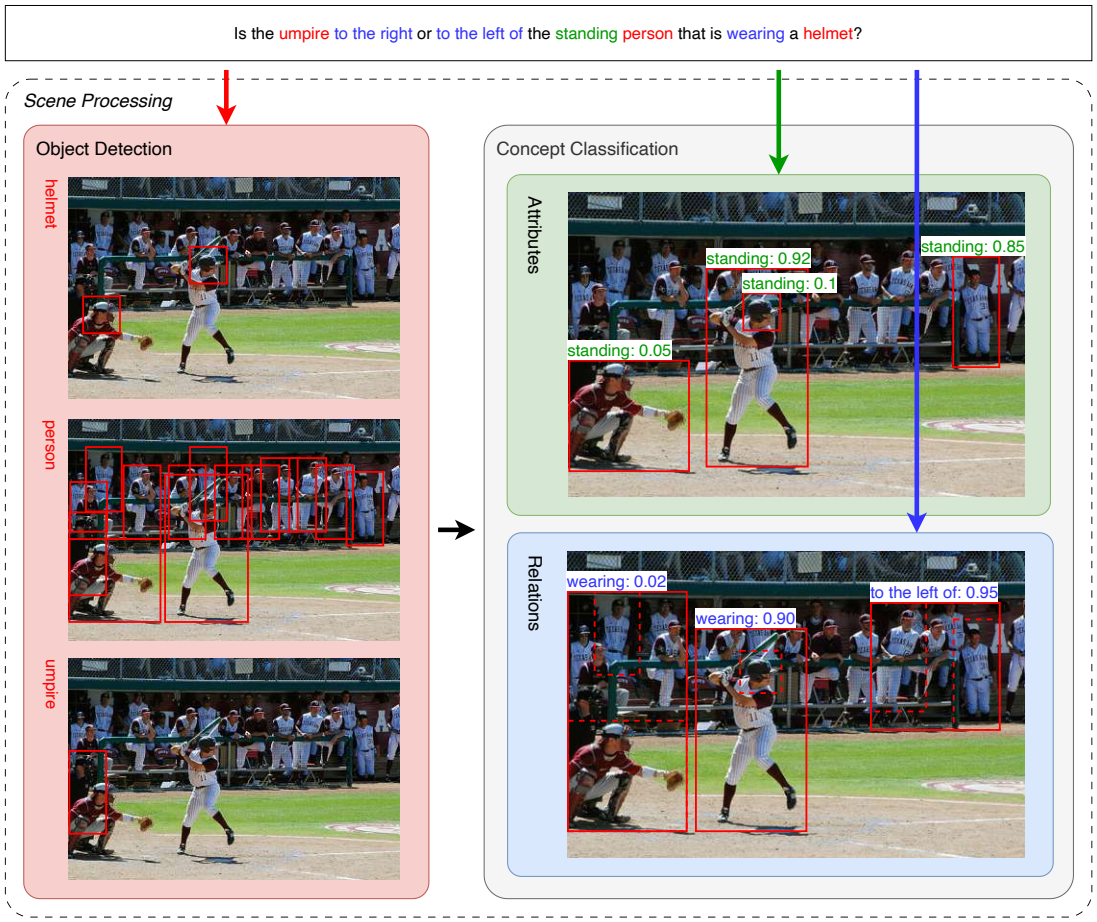


Figure 3.2: An overview over the Scene Processing component

3.4 ASP Encoding

As already mentioned in Section 3.2, the GS-VQA pipeline constructs symbolic encodings of both the input question and the input image, which will respectively be called Question Encoding and Scene Encoding hereafter. Like the VQA pipeline by Eiter et al. [EHO+22], GS-VQA uses ASP as the symbolic formalism for the encodings, which not only provides a mature ecosystem of tooling and solvers (in contrast to custom logic formalisms like ∇-FOL [APP+20]), but more importantly, allows us to capture the uncertainty in the class, attribute, and relation predictions of the Scene Processing component.

So while the Question Encoding is a rather straightforward translation of the input question's semantic representation into a sequence of ASP facts, the Scene Encoding makes use of choice rules and weak constraints to allow the ASP Solver to take the Scene Processing's confidence in detected objects and the likelihoods of attributes and relations into account. With choice rules, the ASP Solver can include or omit object attributes and relations from an answer set. For either of these options, a weak constraint then adds a weight inversely proportional to the likelihood of that attribute/relation applying or not applying to the object, respectively. This way, it is possible to consider negative cases, i.e., an object *not* having a certain attribute/relation.

3.5 ASP Solving

Equipped with an ASP Theory containing the rules that describe the semantics of the reasoning operations in the Question Encoding, the ASP Solving component takes the Question and Scene Encodings as inputs to determine the answer in the most likely model. Since weak constraints in the Scene Encoding turn the ASP evaluation into an optimization problem, a time limit is imposed, after which the current best answer is returned. The answers returned are succinct ASP atoms, in contrast to some VQA approaches that can generate full-sentence answers (see, for example, LRTA [LNR+20]). In the case that no answer can be derived (which can only happen if no object is found by the Object Detection for one of the classes required by the input question), UNSAT is returned.

CHAPTER 4

Implementation

In this chapter, we describe the full implementation details of the components of the GS-VQA pipeline that were introduced conceptually in the previous chapter. Since the general flow of information has already been laid out in Chapter 3, and it is difficult to cover the edge-cases of all components in one continuous example, each section in this chapter will use one or multiple disjoint examples to illustrate its explanations.

4.1 Preprocessing

Like other works in the literature [APP+20], we ignore a particular set of questions in GQA that are not answerable with the information contained in the ground-truth scene graph of the corresponding image. These questions refer to the image as a whole, rather than objects within it, and all start with a `select: scene` operation. Examples for this set of questions are: "Which place is it?", "Is it an outdoors scene?", and "What is the image showing?".

Furthermore, we singularize plural object classes since the semantic representation of GQA uses singular/plural inconsistently: for example, the question "Is there a pepper or a potato that is not red?" has the operation `select: potatoes` in its semantic representation.

4.2 Concept Extraction

As noted in the architectural overview in Section 3.2, the Concept Extraction component's job is to produce a tuple (C, A, R) with the classes, attributes and relations relevant to answering the question. Table 4.1 contains all operations that can occur in the GQA semantic question representation and the classes, attributes, or relations extracted from

them. However, we have to explain some particularities that complicate the simplified picture presented in Section 3.2.

First, we have to fix some notation: Let $\mathcal{C}$ be the set of all classes in the current dataset (GQA in our case). Accordingly, let $\mathcal{A}$ be the set of all attribute categories, $\mathcal{V}$ be the set of all attribute values, and $\mathcal{R}$ be the set of all relations in the dataset. Furthermore, let $cv : \mathcal{A} \to 2^{\mathcal{V}}$ be the mapping of attribute categories to the attribute values included in that category.

Regarding classes, certain questions reference objects whose class is not restricted by any operation of the question. For example, for the question "What is the man holding?", the object in question is only restricted by its `holding` relation to an object of class `man`, but could be of any class. In GQA's semantic question representation, these cases all contain a `relate` operation without class restriction (e.g., `relate: _,holding,o`). To signify to the Object Detection component that objects of all possible classes in the scene could be of relevance, a special `all` class is introduced and added to C in these cases. As such, strictly speaking $C \subseteq \mathcal{C} \cup \{\texttt{all}\}$, though we will include `all` in $\mathcal{C}$ for simplicity in the remainder of this thesis. Note that GQA does not have questions like "What is blue?" that do not contain a relation operation but still require all objects in the scene, though it would be easy to add support for them with a `select: _` operation for which the special `all` class is added to C as well.

For the attributes, the distinction between attribute categories and attribute values introduced in Section 2.2 comes into play. Usually, operations in GQA's semantic question representation that work with attributes contain both a category and the relevant value(s) of that category (if any). However, GQA contains some attribute values that are not associated with any category (Table 4.1 contains some examples: `burnt`, `healthy`, `dried`). For these, operation variants without a category exist (e.g., `filter: burnt` instead of `filter color: red`). To handle these *standalone* values, the set A is actually split into two sets A_c, A_v, where attribute categories extracted from the semantic question representation are added to A_c, while relevant standalone values are added to A_v. The information passed to the Scene Processing component is therefore actually (C, A_c, A_v, R), with $A_c \subseteq \mathcal{A}$, $A_v \subseteq \mathcal{V}$, and $R \subseteq \mathcal{R}$. To declutter the notation in the succeeding sections, we consider standalone values to be part of the `any` attribute category, which is considered included in $\mathcal{A}$.

Similar to the situation for classes, there is one operation that requires all attribute categories to be considered. The `common` operation, which appears in questions like "What do the plate and the silver watch have in common?", has as its answer the attribute category for which its two input objects have the same value. If one such operation occurs in the input question, we set $A_c = \mathcal{A}$.

We note that the extraction is not yet as efficient as it could be: for certain operations like `filter color: red`, we know that only a specific value (`red`) of the attribute category (`color`) is relevant to the question, but add the entire category to A_c regardless.

Category	Operation	Extracted Concepts
Select	`select: person`	`person` (C)
Filter	`filter color: red`	`color` (A_c)
	`filter color: not(red)`	`color` (A_c)
	`filter: burnt`	`burnt` (A_v)
	`filter: not(burnt)`	`burnt` (A_v)
Relate	`relate: person,holding,s`	`person` (C), `holding` (R)
	`relate: _,holding,o`	`all` (C), `holding` (R)
	`relate: person,same_pose,_`	`person` (C), `pose` (A_c)
Query	`query: color`	`color` (A_c)
	`query: name`	–
Verify	`verify color: red`	`color` (A_c)
	`verify: burnt`	`burnt` (A_v)
	`verify rel: shorts,wearing,o`	`shorts` (C), `wearing` (R)
Choose	`choose color: red|blue`	`color` (A_c)
	`choose: dried|wet`	`dried` (A_v), `wet` (A_v)
	`choose rel: bat,holding|wearing,o`	`bat` (C), `holding` (R), `wearing` (R)
	`choose less healthy` (analogous for `more`)	`healthy` (A_v)
	`choose healthier`	`healthy` (A_v)
Exist	`exist`	–
Different/ Same	`same color` (analogous for `different`)	`color` (A_c)
	`same: color` (analogous for `different`)	`color` (A_c)
Common	`common`	$\mathcal{A}$ (A_c)
And	`and`	–
Or	`or`	–

Table 4.1: The extracted concepts from each operation in GQA's semantic question representation. Classes are denoted with (C), attribute categories with (A_c), standalone values with (A_v), and relations with (R)

4.3 Scene Processing

Having identified the concepts relevant to answering a question as a (C, A_c, A_v, R) tuple in the Concept Extraction component, we now have to actually construct the partial scene graph that contains this information from the input image. Refining the informal notions of Section 3.3, this partial scene graph is represented as a list O of objects $o_i = (id, s, B, c, A_o, R_o)$, each having a unique identifier $id \in \mathcal{I}$ for some set of alphanumeric identifiers $\mathcal{I}$, a detection confidence score $s \in [0, 1]$, a bounding box

$B = (x, y, w, h) : \mathbb{R}^4_{\geq 0}$, a class $c \in \mathcal{C}$, attribute likelihoods $A_o : \mathcal{A} \to (\mathcal{V} \to [0, 1])$, and relation likelihoods $R_o : \mathcal{I} \to (\mathcal{R} \to [0, 1])$. As already alluded to in Section 3.3, constructing the graph is done in two stages.

First, the Object Detection sub-component detects objects and their bounding boxes in the scene that conform to one of the classes in C. Second, the Concept Classification sub-component takes these objects as input and supplements the likelihoods for attributes and relations applying to them.

4.3.1 Object Detection

For object detection, we use OWL-ViT [MGS+22], which is capable of operating in an open-vocabulary manner, i.e., the classes for which it should detect objects can vary from invocation to invocation without any modifications to the model. Given a list of class labels to detect, OWL-ViT returns a list of object detections, each with the class label $c \in \mathcal{C}$ it was detected as, a detection confidence score $s \in [0, 1]$, and a bounding box $B = (x, y, w, h) : \mathbb{R}^4_{\geq 0}$, where (x, y) are the coordinates of the top-left corner of the bounding box (with the top-left corner of the image having coordinates $(0, 0)$), and w and h are the width and height of the bounding box.

We process the classes in C in a multi-staged manner, in the order of class specificity according to a given hierarchy, here GQA's as depicted in Figure 2.4a. As an example to illustrate the explanations below, we assume that we have extracted the following set C of relevant classes from an input question: $C = \{\texttt{vehicle}, \texttt{van}, \texttt{driver}, \texttt{all}\}$.

1. As a first step, OWL-ViT is used to detect objects with classes of maximal specificity (i.e., classes without sub-classes in the hierarchy), in this case `van` and `driver`. We directly use the class names as labels for OWL-ViT, and do not transform them to, e.g., "a photo of a `{class}`". As a clean-up step, objects whose bounding boxes overlap by more than a threshold t_{bbox1} and have the same class c are combined, i.e., they are returned as one with the smallest bounding box enclosing both original boxes and the higher of the two object's detection confidence scores. The number of objects returned is limited in two ways: First, only objects for which OWL-ViT's detection confidence score exceeds a certain threshold t_s are returned. Second, if more than k_1 objects are detected for a class with a confidence score exceeding t_s, only the top k_1 objects (sorted by confidence score) are returned. The result is a list of object proposals $O_1 = [(s_1, B_1, c_1), (s_2, B_2, c_2), ...]$.

2. In the second step, for each class higher up in the class hierarchy, objects of all sub-classes of maximal specificity of that class are detected. Like in step 1, the threshold t_s applies, and the maximum number of objects returned for one higher-order class is limited by a separate threshold k_2. While OWL-ViT would also directly accept the higher-order class for detection, we require for each detected object a class of maximal specificity in case that it becomes the target of a `query: name` operation (for example, GQA expects questions like "What thing is the man holding?" to be

answered with "burger" and not simply "food"). Another benefit to detecting the sub-classes of maximal specificity is that some higher-order classes are rather vague and therefore difficult for OWL-ViT to detect (for example, OWL-ViT struggles with detecting objects of class "watercraft", but is rather adept at detecting a "boat"). Note that this approach requires the class hierarchy for the current dataset, which we deem an acceptable compromise, since regular SGG or open-vocabulary object detection for all classes in the dataset would require at least the complete list of possible classes in any case.

For our example, OWL-ViT would be run with all sub-classes of maximal specificity of the higher-order class `vehicle`: `minivan`, `jeep`, `locomotive`, `bus`, `van`, etc. The list of returned object proposals O_2^* is merged with the list of objects O_1 detected in the first step to yield the step output list O_2 in the following way: if the bounding boxes of an object in O_1 and O_2^* overlap by more than a certain percentage threshold t_{bbox2}, and the classes of the two objects have a common ancestor/higher-order class, the object from O_1 is included in O_2, and the object from O_2^* is discarded. This ensures that objects are not considered multiple times, and that objects that might feasibly belong to multiple classes of maximal specificity (e.g., `van` and `minivan`) are considered under a class of maximal specificity implied by the question.

3. The third and final step runs only if the special `all` class is included in C. In this case, OWL-ViT is run once more, this time with all classes of maximum specificity in the hierarchy. Again the threshold t_s applies, and the length of the list O_3^* of object proposals returned by this run is limited by a threshold k_3. The list of new objects O_3^* is merged with the existing list O_2 in exactly the same manner as was done in step 2, yielding the final list of object proposals O_3.

Finally, each object proposal in O_3 is assigned a unique identifier $id \in \mathcal{I}$ to yield the list of partial scene graph objects $O = [(id_1, s_1, B_1, c_1), (id_2, s_2, B_2, c_2), ...]$.

4.3.2 Concept Classification

Taking the list of partial scene graph objects O from the Object Detection sub-component, the Concept Classification sub-component uses CLIP [RKH+21] to supplement for each object the attribute and relation likelihoods A_o, R_o according to the question-relevant attribute categories A_c, attribute values A_v, and relations R from the Concept Extraction component.

Attribute Likelihoods

To determine the attribute likelihoods $A_{o,i}$ for each object $o_i = (id_i, s_i, B_i, c_i) \in O$, a crop of the image area under its bounding box B_i is created. Since especially small objects often need context to be identified even by humans (e.g., an image crop of the bounding box of a wooden table leg would just show an indeterminate brown block),

we add padding to all four sides of the bounding box before creating the image crop. The padding p_w added to the left and right sides of the bounding boxes is determined from the box width w_{bbox} and the image width w_{image} as follows, with the top/bottom padding determined analogously:

$$p_w = \left[1 - \tanh\left(2\frac{w_{\text{bbox}}}{w_{\text{image}}}\right)\right] * w_{\text{bbox}} \tag{4.1}$$

This formulation ensures that the bounding boxes of small objects (relative to the image size) are padded by a large amount relative to their size (up to $w_{\text{bbox}}/h_{\text{bbox}}$ on each side in the limit) to provide the required context around the objects, but the amount of padding decreases quickly as object size increases. To build the attribute likelihoods $A_{o,i}$, we need for each attribute value in A_v and each attribute value of each attribute category in A_c (so each attribute value $a \in A_v \cup \{a' \mid a_c \in A_c, a' \in cv(a_c)\}$) a number in $[0, 1]$ that represents the likelihood of that attribute value applying to object o_i. As introduced in Section 2.4.1, we can calculate the cosine similarity between CLIP's embedding of the padded object crop and the embedding of a textual prompt to get an indication of the semantic similarity between these two: prompts "fitting" the image crop will have higher cosine similarities than prompts that don't. However, this concept of semantic similarity requires a point of reference: what constitutes a "high" cosine similarity with a prompt embedding varies from image to image.

Taking inspiration from a paper by Sarri and Rodriguez-Fernandez [SR21], we use a modified version of their "target vs. neutral" approach to introduce this frame of reference: We introduce a neutral prompt, i.e., one that we are sure applies to the current object o_i. The neutral prompt follows the schema "a blurry photo of {a/an} $\{c_i\}$" (for example, if $c_i =$ `van`, the neutral prompt would be "a blurry photo of a van"). We then build for each attribute value a a modified version more specific to that value, the target prompt, which follows the schema "a blurry photo of {a/an} $\{a\}$ $\{c_i\}$". For example, for $a =$ `red`, the target prompt would be "a blurry photo of a red van".

If the additional information added to the target prompt actually applies to the object, CLIP should return a higher cosine similarity between its embedding and the object crop than between the neutral prompt embedding and the object crop. Conversely, if the additional information does not apply to the object, the cosine similarity should be lower than for the neutral prompt. We can therefore frame the computation of the likelihood of attribute value a applying to object o_i as a binary classification problem between "applies" and "does not apply", represented by the target and neutral prompts, respectively. To obtain values in $[0, 1]$, we simply compute the softmax over the cosine similarity values between the two prompts and the object crop. More formally, let $\mathbf{f}_{\text{CLIP}}(x)$ be the embedding vector of an image or a text prompt x as returned by CLIP's image or text encoder (which are already normalized). Furthermore, let p_t, p_n be the target and neutral prompts, and I the object crop. We compute the likelihood of attribute a applying as $\text{softmax}\left(\mathbf{f}_{\text{CLIP}}(I) \cdot \mathbf{f}_{\text{CLIP}}(p_t), \mathbf{f}_{\text{CLIP}}(I) \cdot \mathbf{f}_{\text{CLIP}}(p_n)\right)_0$.

Relation Likelihoods

For the relation likelihoods $R_{o,i}$ of each object o_i, we use the same "target vs. neutral" approach as for attributes. Between o_i and every other object $o_j = (id_j, s_j, B_j, c_j) \in O \setminus \{o_i\}$, we determine the smallest bounding box enclosing the two objects' original boxes, and use the crop of the image area under that bounding box. For each relation $r \in R$, the neutral prompt follows the schema "$\{c_i\}$ and $\{c_j\}$", while the target prompt adheres to the schema "$\{c_i\}$ $\{r\}$ $\{c_j\}$". For example, if c_i = `woman`, c_j = `van`, and r = `driving`, then the neutral prompt would be "woman and van", and the target prompt would be "woman driving van".

Alternative Approaches

Another approach discussed by Sarri and Rodriguez-Fernandez [SR21] is "target vs. contrary". Rather than using a neutral prompt that applies to the current object with certainty, a prompt containing the opposite of the current concept, i.e., its antonym, is used. For example, for the target prompt "a blurry photo of a wet dog", the contrary prompt would be "a blurry photo of a dry dog". In theory, this approach emphasises the difference between the two prompts more than "target vs. neutral", which should lead to a greater deviation in cosine similarities. However, it is difficult to apply to our use-case, since the opposite of a concept is not always clear. For many attribute categories like `material`, `pose`, etc., there is no conceptual opposite (e.g., what would the opposite of `brick` as in "a brick house" be?), or the opposite is unclear (e.g., what is the opposite of `standing`: `sitting` or `walking`?). For relations, the opposite would have to be formed by prepending "not", as in "woman driving van" vs. "woman not driving van". In many cases, this makes little sense as a legitimate image caption and relies on the assumption that CLIP has a good understanding of negation.

In their ViperGPT model, Surís et al. [SMV23] use an approach for attribute classification that we will call "target vs. ensemble". Instead of computing the likelihood of an attribute value applying to an object with a single binary classification, this approach uses multiple binary classifications, each between a target prompt and a contrastive prompt randomly sampled from a set of alternative attribute values. The final score reported for the target attribute value is the mean of these classification results. For example, with the target attribute value `red`, the target prompt might be "a red van", and the contrastive prompts might be "a blue van", "a burning van", "a large van", and so on. Since most of the attribute values used as contrastive samples likely do not apply to the current object, and should therefore be further away from the object crop than a neutral "a van" prompt in CLIP's embedding space, this approach again has the benefit of emphasising the difference between each pair of prompts. It also does not require an opposite to every target attribute, but the obvious disadvantage is computational cost, since the number of prompts that need to be processed is considerably higher.

Prompt Engineering

To experiment with multiple prompt schemas and find the optimal one for the "target vs. neutral" prompting approach, we build evaluation datasets in the following way:

1. We randomly sample 50,000 questions from GQA's validation set. We then extract from these questions, by the same principles as followed in the Concept Extraction component, the question-relevant attribute values and relations.

2. We sample an image crop for each attribute value and relation as follows:

   - For each attribute value, we randomly choose one object to which this value applies from the ground-truth scene graph of the image associated with the question. For this object, we create a padded image crop from the image area under its bounding box as is done in the Concept Classification sub-component.
   - For relations, we similarly sample two objects from the ground-truth scene graph that are connected by that relation and obtain an image crop of the combined bounding box.

   These pairs of (attribute value, image crop) and (relation, image crop) form our datasets of positive samples, where we know that the attribute value/relation applies to the object(s) depicted in the image crop.

3. We shuffle the two positive datasets: For each (attribute value, image crop) pair we randomly select an image crop from a different pair where the underlying object does not have the attribute value in the associated ground truth scene graph; the positive (relation, image crop) pairs are treated accordingly. Through this, we obtain for each of the two positive datasets a negative one of equal size, where the attribute value/relation of each sample does not apply to the object(s) depicted in the image crop.

4. Finally, we combine the positive and negative datasets, yielding a total of 27,470 samples for attribute classification, and 47,634 samples for relation classification.

We test multiple schemas for target (representing the positive class, i.e., the attribute value/relation applying to the object depicted in the image crop) and neutral (representing the negative class) prompts for classification with CLIP [RKH+21] as described in Section 2.4.1 and choose the one obtaining the highest accuracy for the remainder of our evaluations. For attributes, the best-performing target prompt schema is "a blurry photo of {a/an} {a} {c_i}", while for relations, the best schema is "{c_i} {r} {c_j}". The results for all explored schemas for attribute and relation classification are shown in the Tables 4.2 and 4.3, respectively. As can be seen from the tables, the impact of prompt choice for our use case is small, especially for attribute classification: from the lowest- to the highest-performing prompt schema, the difference in accuracy is 1.25 percentage points for attributes, and 2.52 percentage points for relations. Curiously, for some prompt schemas

we observe a higher-than-average precision value coupled with a lower-than-average recall value, and vice-versa. This would indicate that for these prompt schemas, the threshold for classifying an attribute/relation as "applies" needs to be adjusted away from 0.5 to achieve optimal accuracy.

Schema	Accuracy	Precision	Recall
"a bad photo of {a/an} {a} {c_i}"	0.6583	0.6728	0.6165
"a blurry photo of {a/an} {a} {c_i}"	**0.6590**	0.6620	0.6495
"a pixelated photo of {a/an} {a} {c_i}"	0.6512	0.6492	0.6577
"a low resolution photo of {a/an} {a} {c_i}"	0.6555	0.6572	0.6499
"a photo of {a/an} {a} {c_i}"	0.6522	0.6621	0.6215
"{a/an} {a} {c_i}"	0.6533	0.6650	0.6178
"{a} {c_i}"	0.6465	**0.6776**	0.5591
"itap of {a/an} {a} {c_i}"	0.6529	0.6741	0.5918
"a bad picture of {a/an} {a} {c_i}"	0.6561	0.6673	0.6227
"a blurry picture of {a/an} {a} {c_i}"	0.6549	0.6564	0.6501
"a pixelated picture of {a/an} {a} {c_i}"	0.6551	0.6518	**0.6660**
"a low resolution picture of {a/an} {a} {c_i}"	0.6571	0.6593	0.6502
"a picture of {a/an} {a} {c_i}"	0.6560	0.6568	0.6534

Table 4.2: Results of the evaluation of various attribute prompt schemas, with a being the current attribute value and c_i the class of the current object o_i

As a point of reference, using the same method we extract from our 50,000 sampled questions a dataset of 97,246 (class, image crop) entries, again split 50/50 into positive/negative samples. Using the target prompt schema "a photo of {a/an} {c_i}" and the neutral prompt schema "a photo of an object", CLIP achieves an accuracy of 0.8287, i.e., it is able to correctly identify class membership or non-membership for 82.87% of samples. These experiments highlight an important finding that has already been made elsewhere in the literature [ZZL22; ZZZ+23]: current state-of-the-art VLMs have a strong understanding of object classes, but less so of attributes, and even less of relations between objects. In Chapter 5, we explore multiple approaches for alleviating this performance gap.

Performance Considerations

The fact that image and text embeddings in CLIP can be computed in isolation allows for some optimization to improve the runtime of the Scene Processing component. During both attribute and relation likelihood computation, we compute the embeddings of the image crops and the neutral prompts for each object or pair of objects only once, and then re-use them with the embeddings of different target prompts for binary classification.

Schema	Accuracy	Precision	Recall
“a bad photo of {a/an} {c_i} {r} {a/an} {c_j}”	0.5469	0.5384	0.6569
“a blurry photo of {a/an} {c_i} {r} {a/an} {c_j}”	0.5380	0.5290	0.6939
“a pixelated photo of {a/an} {c_i} {r} {a/an} {c_j}”	0.5533	0.5495	0.5922
“a low resolution photo of {a/an} {c_i} {r} {a/an} {c_j}”	0.5482	0.5476	0.5542
“a photo of {a/an} {c_i} {r} {a/an} {c_j}”	0.5544	0.5430	0.6880
“{a/an} {c_i} {r} {a/an} {c_j}”	0.5493	0.5367	**0.7213**
“{c_i} {r} {c_j}”	**0.5632**	**0.5530**	0.6590
“itap of {a/an} {c_i} {r} {a/an} {c_j}”	0.5415	0.5403	0.5562
“a bad picture of {a/an} {c_i} {r} {a/an} {c_j}”	0.5431	0.5359	0.6429
“a blurry picture of {a/an} {c_i} {r} {a/an} {c_j}”	0.5395	0.5316	0.6656
“a pixelated picture of {a/an} {c_i} {r} {a/an} {c_j}”	0.5524	0.5493	0.5834
“a low resolution picture of {a/an} {c_i} {r} {a/an} {c_j}”	0.5499	0.5495	0.5541
“a picture of {a/an} {c_i} {r} {a/an} {c_j}”	0.5583	0.5469	0.6799

Table 4.3: Results of the evaluation of various relation prompt schemas, with r being the current relation and c_i, c_j the classes of the current objects o_i, o_j

For the computation of relation likelihoods, we apply one further optimization: since the number of object pairs (o_i, o_j) is quadratic in the number of objects in O, the number of combined object bounding boxes whose corresponding image crops we would have to embed is also quadratic in $|O|$. However, many object pairs have highly overlapping combined bounding boxes (as an example, consider the input image from Figure 3.2: the combined bounding box of the umpire and the striking player, and the combined box of the umpire and the player's helmet, will be nearly identical). We therefore merge object pair bounding boxes that overlap by more than a certain percentage threshold t_{bbox3} and re-use them for all affected object pairs.

4.4 ASP Encoding

4.4.1 Question Encoding

The Question Encoding is strongly inspired by the one that Eiter et al.'s pipeline [EHO+22] uses as a translation of the semantic representations of natural-language questions from the CLEVR [JHvdM+17a] dataset. Each operation in the semantic representation of the question is—in general—translated into one ASP fact, although additional facts might be inserted to enforce certain properties that are implicit in the semantic representation of GQA (e.g. uniqueness, inverted criteria).

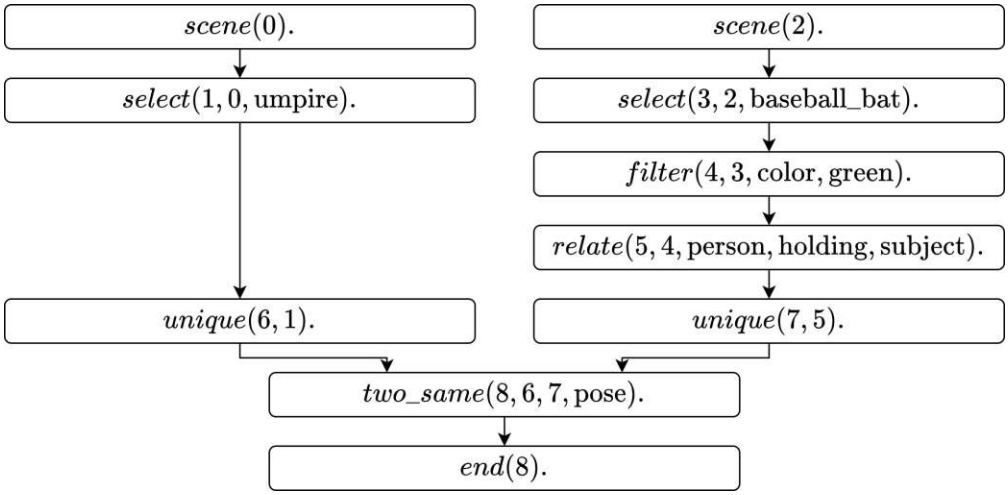


Figure 4.1: The Question Encoding for the question "Do the umpire and the person holding the green baseball bat have the same pose?"

We illustrate the encoding with the example question "Do the umpire and the person holding the green baseball bat have the same pose?", whose semantic representation is shown in Figure 2.3. The corresponding ASP Question Encoding is presented in Figure 4.1.

From the example it can be seen that the structure of the tree of reasoning operations in the semantic representation is encoded using step indices. Each predicate encoding an operation takes as first argument the index of its output step, then one or more indices of its input step(s), followed by the remaining arguments specific to the operation. If an operation has no predecessor in the semantic representation, a *scene*() fact is inserted, and an *end*() fact is always added for the last step index. The mapping of all operations from the GQA semantic representation to their corresponding Question Encoding facts is shown in Table 4.4.

4.4.2 Scene Encoding

As explained in detail in Section 4.3, the Scene Processing component outputs a partial scene graph represented as a list O of objects $o_i = (id_i, s_i, B_i, c_i, A_{o,i}, R_{o,i})$, each having a unique identifier $id_i \in \mathcal{I}$, a score $s_i \in [0, 1]$, a bounding box $B_i = (x, y, w, h) : \mathbb{R}^4_{\geq 0}$, a class $c_i \in \mathcal{C}$, attribute likelihoods $A_{o,i} : \mathcal{A} \to (\mathcal{V} \to [0, 1])$, and relation likelihoods $R_{o,i} : \mathcal{I} \to (\mathcal{R} \to [0, 1])$. The information for each object o_i is converted into a set of ASP facts and constraints as described in the following paragraphs. As an example, we use the scene graph entry of a striking baseball player presented in Listing 3.1 from Section 3.3.

Objects: To start off, we add an *object*(id_i) fact to the encoding to establish the existence of the object in the scene. To be able to select the object that the Object Detection component has the most confidence in when filtering a set of qualifying

objects with the *unique*() operation, we add a fact $has\_obj\_weight(id_i, w_o)$, where $w_o = \lfloor \min(-1000 \cdot \ln(s_i), 5000) \rfloor$. The formulation of w_o is taken from NeurASP [YIL20] and ensures that the "higher-is-better" score s_i is correctly adapted to a "lower-is-better" weight for the *unique*() operation (see Equation 4.51). So, for our example object, the added facts would be:

$$object(\text{o1}). \tag{4.2}$$
$$has\_obj\_weight(\text{o1}, 1971). \tag{4.3}$$

Classes: For the object's class c_i and all of its parent classes (if any), we add a fact $has\_attr(id_i, \text{class}, c_i)$. This allows the object to be considered not just for operations filtering on its specific class, but also the parent classes it belongs to. Additionally, we add a fact $has\_attr(id_i, \text{name}, c_i)$. Questions in GQA that inquire the type of a certain object (e.g., "Who is wearing a shirt?") always have `query: name` as the terminal operation, and the expected answer is the object's most specific class. Having $has\_attr(id_i, \text{name}, c_i)$ in the encoding allows us to easily identify the most specific class of the object for these cases. For our example object, the added facts are thus:

$$has\_attr(\text{o1}, \text{class}, \text{alive}). \tag{4.4}$$
$$has\_attr(\text{o1}, \text{class}, \text{person}). \tag{4.5}$$
$$has\_attr(\text{o1}, \text{class}, \text{baseball\_player}). \tag{4.6}$$
$$has\_attr(\text{o1}, \text{name}, \text{baseball\_player}). \tag{4.7}$$

Attributes: For each attribute value v in category a in $A_{o,i}$, we add the choice rule $\{has\_attr(id_i, a, v)\}.$, which ensures that the ASP solver can consider both the case where the attribute value applies to the object, and the case where it doesn't. To include the likelihoods of those two cases into the consideration, we add the weak constraints $:\sim has\_attr(id_i, a, v).\ [w_{v^+}, (id_i, a, v)]$ and $:\sim not\ has\_attr(id_i, a, v).\ [w_{v^-}, (id_i, a, v)]$, where $w_{v^+} = \lfloor \min(-1000 \cdot \ln(s_v), 5000) \rfloor$ and $w_{v^-} = \lfloor \min(-1000 \cdot \ln(1 - s_v), 5000) \rfloor$, s_v being the likelihood of v applying to the object according to $A_{o,i}$. For the example object and the `standing` attribute value of the `pose` category, we therefore add the following rules and constraints to the encoding:

$$\{has\_attr(\text{o1}, \text{pose}, \text{standing})\}. \tag{4.8}$$
$$:\sim has\_attr(\text{o1}, \text{pose}, \text{standing}).\ [83, (\text{o1}, \text{pose}, \text{standing})] \tag{4.9}$$
$$:\sim not\ has\_attr(\text{o1}, \text{pose}, \text{standing}).\ [2525, (\text{o1}, \text{pose}, \text{standing})] \tag{4.10}$$

Additionally, we add facts for the two attribute categories that are explicitly excluded from Scene Processing, `hposition` and `vposition`, whose value for each object is

directly derived from the bounding box B_i. We divide the input image horizontally and vertically into thirds. For `hposition`, the object gets the value `left`, `middle`, or `right` if the center of its bounding box falls within the left, middle, or right horizontal third of the input image. Analogously for `vposition`, the object gets the value `top`, `middle`, or `bottom` if the center of its bounding box falls within the upper, middle, or lower vertical third of the input image. For our example object, given its bounding box center of $(205, 185)$ and the 500×334 dimensions of its corresponding input image, this yields the following facts:

$$has\_attr(\text{o1}, \text{hposition}, \text{middle}). \tag{4.11}$$
$$has\_attr(\text{o1}, \text{vposition}, \text{middle}). \tag{4.12}$$

Relations: For each other object $o_j = (id_j, s_j, B_j, c_j, A_{o,j}, R_{o,j}) \in O \setminus \{o_i\}$ and relation r in $R_{o,i}$, we proceed like we do for attribute values. We add the choice rule $\{has\_rel(id_i, r, id_j)\}.$, and the weak constraints $:\sim has\_rel(id_i, r, id_j).\,[w_{r^+}, (id_i, r, id_j)]$ and $:\sim not\, has\_rel(id_i, r, id_j).\,[w_{r^-}, (id_i, r, id_j)]$. Here, $w_{r^+} = \lfloor \min(-1000 \cdot \ln(s_r), 5000) \rfloor$ and $w_{r^-} = \lfloor \min(-1000 \cdot \ln(1 - s_r), 5000) \rfloor$, s_r being the likelihood of r applying with o_i as subject and o_j as object according to $R_{o,i}$. For the example object as subject, o_2 as object, and the `wearing` relation, we therefore extend our encoding as follows:

$$\{has\_rel(\text{o1}, \text{wearing}, \text{o2})\}. \tag{4.13}$$
$$:\sim has\_rel(\text{o1}, \text{wearing}, \text{o2}).\,[105, (\text{o1}, \text{wearing}, \text{o2})] \tag{4.14}$$
$$:\sim not\, has\_rel(\text{o1}, \text{wearing}, \text{o2}).\,[2302, (\text{o1}, \text{wearing}, \text{o2})] \tag{4.15}$$

4.4.3 ASP Theory

The ASP Theory consists of a set of rules that—in contrast to the Question and Scene Encoding—do not change from question to question and encode the semantics of each of the reasoning operations that can appear as part of the Question Encoding. To keep the rules compact, we introduce a couple of variable short-hands: T_i and T_o are variables representing input/output step references, I represents an object id, C a class, A an attribute category, V an attribute value, and R a relation.

Base Operations

Scene: The *scene*() operation simply returns all objects in the scene as encoded by the Scene Encoding. Its single ASP rule is defined as follows:

$$state(T_o, I) :- scene(T_o), object(I). \tag{4.16}$$

End: The *end*() operation converts the different possible answer types coming from its input step into a common *ans*() predicate. Its rules are simple in structure:

$$ans(V) :- end(T_o), attr\_value(T_o, V). \tag{4.17}$$
$$ans(A) :- end(T_o), attr(T_o, A). \tag{4.18}$$
$$ans(R) :- end(T_o), rel(T_o, R). \tag{4.19}$$
$$ans(B) :- end(T_o), bool(T_o, B). \tag{4.20}$$
$$\tag{4.21}$$

We add an integrity constraint that forbids solutions in which no *ans*() predicate can be derived:

$$:- not\ ans(\_). \tag{4.22}$$

Intermediary Operations

Select: The *select*() operation restricts its input set of objects to those that are members of a certain class. Its ASP rule is as follows:

$$state(T_o, I) :- select(T_o, T_i, C), state(T_i, I), has\_attr(I, \text{class}, C). \tag{4.23}$$

Filter: Similarly to the *select*() operation, the *filter*() operation restricts its input objects to those that have a specific value for a certain attribute category. For cases in which it is not known which category the filtered value belongs to, the analogous *filter_any*() operation is used. We present only the rule for the former, since the latter is identical except for the restriction to an attribute category:

$$state(T_o, I) :- filter(T_o, T_i, A, V), state(T_i, I), has\_attr(I, A, V). \tag{4.24}$$

Relate: Relate operations return objects connected to some input object through either a relation or a common attribute value. The *relate*() operation covers the case of connection through a relation, and additionally requires that the connected objects belong to a certain class. We only present the rule variant in which the input object is the object of the relation, and the connected object is the subject; the variant with swapped subject/object positions is analogous:

$$\begin{aligned} state(T_o, I_2) :- relate(T_o, T_i, C, R, \text{subject}), state(T_i, I_1), \\ has\_attr(I_2, \text{class}, C), has\_rel(I_2, R, I_1). \end{aligned} \tag{4.25}$$

The *relate_any*() operation does not restrict the class of the connected objects, and its rules are the same as those for *relate*() except for the omission of the $has\_attr(I_2, \text{class}, C)$ restriction.

The case of connection through a common attribute value is covered by the *relate_attr*() operation, whose sole rule is presented below:

$$\begin{aligned} state(T_o, I_2) :- & \; relate\_attr(T_o, T_i, C, A), state(T_i, I_1), \\ & has\_attr(I_1, A, V), has\_attr(I_2, \text{class}, C), \\ & has\_attr(I_2, A, V), I_1 \neq I_2. \end{aligned} \tag{4.26}$$

Compare: The *compare*() operation takes two steps, an attribute value, and a mode as input. It checks if one of the input objects from the two input steps (respectively assumed unique) has the attribute value, and the other one does not. If so, it returns the object having the attribute value if the mode is `true`, and the object not having the attribute value if the mode is `false`. We present the rules for the mode `true` below, the rules for the mode `false` are analogous:

$$\begin{aligned} state(T_o, I_1) :- & \; compare(T_o, T_{i1}, T_{i2}, V, \text{true}), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & has\_attr(I_1, \_, V), not\; has\_attr(I_2, \_, V). \end{aligned} \tag{4.27}$$

$$\begin{aligned} state(T_o, I_2) :- & \; compare(T_o, T_{i1}, T_{i2}, V, \text{true}), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & not\; has\_attr(I_1, \_, V), has\_attr(I_2, \_, V). \end{aligned} \tag{4.28}$$

Terminal Operations

Query: The *query*() operation returns the value of a specific attribute category for its input object (it implicitly assumes only one input object is present). Its rule is accompanied by a cardinality rule (shown in simplified form in Equation 4.30) that ensures that the optimization must assign exactly one value for the attribute category to the object. As with other operation-specific cardinality rules that follow, we do not enforce this constraint in general, since it may be beneficial to consider multiple similar attribute values (e.g., `brown`, `beige`) to apply to an object, for example for filter operations. The added rules are shown below:

$$attr\_value(T_o, V) :- query(T_o, T_i, A), state(T_i, I), has\_attr(I, A, V). \tag{4.29}$$

$$\{has\_attr(I, A, V) : is\_attr\_value(A, V)\} = 1 :- query(T_o, T_i, A), state(T_i, I). \tag{4.30}$$

Verify: Verify operations return a boolean that indicates whether the input object (assumed unique) is connected through a specific relation to some object with a certain class ($verify\_rel()$), or has a specific attribute value ($verify\_attr()$). Of the rules for

$verify\_rel()$, we present only the case in which the input object is in the object position of the relation; the case for the subject position is analogous:

$$bool(T_o, \text{yes}) :- verify\_rel(T_o, T_i, C, R, \text{subject}), state(T_i, I_1), has\_attr(I_2, \text{class}, C), has\_rel(I_2, R, I_1). \tag{4.31}$$

$$bool(T_o, \text{yes}) :- verify\_attr(T_o, T_i, A, V), state(T_i, I), has\_attr(I, A, V). \tag{4.32}$$

For all of these rules, a dual one exists that forces the operation output to `no` if the conditions for `yes` are not fulfilled. The one for $verify\_attr()$ is shown below:

$$bool(T_o, \text{no}) :- verify\_attr(T_o, T_i, A, V), not\ bool(T_o, \text{yes}). \tag{4.33}$$

Choose: Choose operations are similar to Verify operations, but rather than asking *if* a certain relation or attribute value is present for the input object (again assumed unique), they ask *which* of two options is present. Of the rules for $choose\_rel()$, we again show only the case in which the input object is in the object position. Also, only the rule for one of the two options is presented, since the other (R_2 or V_2) is analogous:

$$rel(T_o, R_1) :- choose\_rel(T_o, T_i, C, R_1, R_2, \text{subject}), state(T_i, I_1), has\_attr(I_2, \text{class}, C), has\_rel(I_2, R_1, I_1). \tag{4.34}$$

$$attr\_value(T_o, V_1) :- choose\_attr(T_o, T_i, A, V_1, V_2), state(T_i, I), has\_attr(I, A, V_1). \tag{4.35}$$

For both $choose\_rel()$ and $choose\_attr()$, we again use cardinality rules to enforce that only one of the two options can apply:

$$\{has\_rel(I_2, R_1, I_1) : has\_attr(I_2, \text{class}, C); has\_rel(I_2, R_2, I_1) : has\_attr(I_2, \text{class}, C)\} = 1 :- choose\_rel(T_o, T_i, C, R_1, R_2, \text{subject}), state(T_i, I_1). \tag{4.36}$$

$$\{has\_attr(I, A, V_1); has\_attr(I, A, V_2)\} = 1 :- choose\_attr(T_o, T_i, A, V_1, V_2), state(T_i, I). \tag{4.37}$$

Exist: The Exist operation returns whether or not an input object exists:

$$bool(T_o, \text{yes}) :- exist(T_o, T_i), state(T_i, I). \tag{4.38}$$

Like for the Verify operations, a dual rule is added that outputs `no` if the rule outputting `yes` does not apply.

Different/Same: The Different/Same operations come in two distinct variations: The *all_different*() and *all_same*() operations take one step and an attribute category as input and check if across all objects from the input step, their sets of attribute values for the input category are pairwise disjoint (for *all_different*()), or identical (for *all_same*()):

$$\begin{aligned} bool(T_o, \text{no}) :- & \; all\_different(T_o, T_i, A), state(T_i, I_1), state(T_i, I_2), \\ & has\_attr(I_1, A, V), has\_attr(I_2, A, V), I_1 \neq I_2. \end{aligned} \tag{4.39}$$

$$\begin{aligned} bool(T_o, \text{no}) :- & \; all\_same(T_o, T_i, A), state(T_i, I_1), state(T_i, I_2), \\ & has\_attr(I_1, A, V), not\; has\_attr(I_2, A, V), I_1 \neq I_2. \end{aligned} \tag{4.40}$$

The other variant of the Different/Same operations, *two_different*() and *two_same*(), have two input steps and check whether for the input objects from those steps (respectively assumed unique), their sets of attribute values for an attribute category are disjoint (for *two_different*()), or identical (for *two_same*()):

$$\begin{aligned} bool(T_o, \text{no}) :- & \; two\_different(T_o, T_{i1}, T_{i2}, A), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & has\_attr(I_1, A, V), has\_attr(I_2, A, V). \end{aligned} \tag{4.41}$$

$$\begin{aligned} bool(T_o, \text{no}) :- & \; two\_same(T_o, T_{i1}, T_{i2}, A), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & has\_attr(I_1, A, V), not\; has\_attr(I_2, A, V). \end{aligned} \tag{4.42}$$

$$\begin{aligned} bool(T_o, \text{no}) :- & \; two\_same(T_o, T_{i1}, T_{i2}, A), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & not\; has\_attr(I_1, A, V), has\_attr(I_2, A, V). \end{aligned} \tag{4.43}$$

Like for the other terminal operations returning booleans, a dual rule exists for each of the rules above.

Common: The Common operation has two input steps and returns an attribute category for which the input objects from those steps (respectively assumed unique) have a common value:

$$\begin{aligned} attr(T_o, A) :- & \; common(T_o, T_{i1}, T_{i2}), state(T_{i1}, I_1), state(T_{i2}, I_2), \\ & has\_attr(I_1, A, V), has\_attr(I_2, A, V). \end{aligned} \tag{4.44}$$

Utility Operations

Boolean: There are two Boolean operations that combine the boolean output(s) of terminal operation steps as expected according to boolean arithmetic, *and*():

$$bool(T_o, \text{yes}) :- and(T_o, T_{i1}, T_{i2}), bool(T_{i1}, \text{yes}), bool(T_{i2}, \text{yes}). \tag{4.45}$$

$$bool(T_o, \text{no}) :- and(T_o, T_{i1}, T_{i2}), not\; bool(T_o, \text{yes}). \tag{4.46}$$

And $or()$:

$$bool(T_o, \text{yes}) :- or(T_o, T_{i1}, T_{i2}), bool(T_{i1}, \text{yes}). \tag{4.47}$$
$$bool(T_o, \text{yes}) :- or(T_o, T_{i1}, T_{i2}), bool(T_{i2}, \text{yes}). \tag{4.48}$$
$$bool(T_o, \text{no}) :- or(T_o, T_{i1}, T_{i2}), not\ bool(T_o, \text{yes}). \tag{4.49}$$

Unique: The Unique operation forces only one of its input objects objects to be returned, the one with the lowest weight:

$$\{state(T_o, I) : state(T_i, I)\} = 1 :- unique(T_o, T_i). \tag{4.50}$$
$$:\sim unique(T_o, T_i), state(T_o, I), has\_obj\_weight(I, W).[W, (T_o, I)] \tag{4.51}$$

Negate: The Negate operation has two input steps and returns all those input objects from the first step that are not present in the second one:

$$state(T_o, I) :- negate(T_o, T_{i1}, T_{i2}), state(T_{i1}, I), not\ state(T_{i2}, I). \tag{4.52}$$

4.5 ASP Solving

To actually solve the problems represented by the ASP programs that the ASP Encoding component produces, we use the `clingo` system of the Potsdam Answer Set Solving Collection (Potassco) [GKK+19], which consists of the `gringo` grounder and the `clasp` solver. We use the system in its standard configuration, i.e., with no changes to the optimization mode & strategy, heuristics, etc. Since the use of weak constraints turns our ASP programs into optimization problems, we enforce a timeout t_{ASP}, after which optimization is stopped and the current best answer set is returned.

Category	Operation	Resulting Fact(s)
Select	`select: person`	$select(T_o, T_i, \text{person})$.
Filter	`filter color: red`	$filter(T_o, T_i, \text{color}, \text{red})$.
	`filter color: not(red)`	$filter(T_{o1}, T_i, \text{color}, \text{burnt})$. $negate(T_{o2}, T_i, T_{o1})$.
	`filter: burnt`	$filter\_any(T_o, T_i, \text{burnt})$.
	`filter: not(burnt)`	$filter\_any(T_{o1}, T_i, \text{burnt})$. $negate(T_{o2}, T_i, T_{o1})$.
Relate	`relate: person,holding,s`	$relate(T_o, T_i, \text{person}, \text{holding}, \text{subject})$.
	`relate: _,holding,o`	$relate\_any(T_o, T_i, \text{holding}, \text{object})$.
	`relate: person,same_pose,_`	$relate\_attr(T_o, T_i, \text{person}, \text{pose})$.
Query	`query: color`	$unique(T_{o1}, T_i)$. $query(T_{o2}, T_{o1}, \text{color})$.
Verify	`verify color: red`	$unique(T_{o1}, T_i)$. $verify\_attr(T_{o2}, T_{o1}, \text{color}, \text{red})$.
	`verify: burnt`	$unique(T_{o1}, T_i)$. $verify\_attr(T_{o2}, T_{o1}, \text{any}, \text{burnt})$.
	`verify rel: shorts,wearing,o`	$unique(T_{o1}, T_i)$. $verify\_rel(T_{o2}, T_{o1}, \text{shorts}, \text{wearing}, \text{object})$.
Choose	`choose color: red|blue`	$unique(T_{o1}, T_i)$. $choose\_attr(T_{o2}, T_{o1}, \text{color}, \text{red}, \text{blue})$.
	`choose: dried|wet`	$unique(T_{o1}, T_i)$. $choose\_attr(T_{o2}, T_{o1}, \text{any}, \text{dried}, \text{wet})$.
	`choose rel: bat,near|in,o`	$unique(T_{o1}, T_i)$. $choose\_rel(T_{o2}, T_{o1}, \text{bat}, \text{near}, \text{in}, \text{object})$.
	`choose less healthy` (analogous for `more`)	$unique(T_{o1}, T_{i1})$. $unique(T_{o2}, T_{i2})$. $compare(T_{o3}, T_{o1}, T_{o2}, \text{healthy}, \text{false})$. $query(T_{o4}, T_{o3}, \text{name})$.
	`choose healthier`	$unique(T_{o1}, T_{i1})$. $unique(T_{o2}, T_{i2})$. $compare(T_{o3}, T_{o1}, T_{o2}, \text{healthy}, \text{true})$. $query(T_{o4}, T_{o3}, \text{name})$.
Exist	`exist`	$exist(T_o, T_i)$.
Different/ Same	`same color` (analogous for `different`)	$unique(T_{o1}, T_{i1})$. $unique(T_{o2}, T_{i2})$. $two\_same(T_{o3}, T_{o1}, T_{o2}, \text{color})$.
	`same: color` (analogous for `different`)	$all\_same(T_o, T_i, \text{color})$.
Common	`common`	$unique(T_{o1}, T_{i1})$. $unique(T_{o2}, T_{i2})$. $common(T_{o3}, T_{o1}, T_{o2})$.
And	`and`	$and(T_o, T_{i1}, T_{i2})$.
Or	`or`	$or(T_o, T_{i1}, T_{i2})$.

Table 4.4: The Question Encoding for each operation from GQA's semantic representation

CHAPTER 5

Extensions

One of the key benefits of the neural-symbolic approach to VQA is compositionality. Since symbolic information rather than neural network embeddings is used as an interface between the components of the pipeline, those components can be modified, exchanged, and augmented without having to reconfigure or retrain all others. In this chapter, we discuss three such modifications to the core GS-VQA pipeline as described in Chapters 3 and 4, which all aim to improve the pipeline's handling of object attributes and relations.

5.1 Fine-tuning of CLIP

As we have seen in Section 4.3.2, state-of-the-art VLMs have a far weaker understanding of attributes and relations than they have of object classes. With this extension, we fine-tune the CLIP model that we use for concept classification on image crops depicting objects, and captions containing attributes or relations applying to the depicted objects. Instead of adding a new classification head on top of CLIP's vision and text encoders, we tune CLIP with the same constrastive pre-training objective that was used in its initial training and continue using the "target vs. neutral" prompting technique that we describe in Section 4.3.2. While training one or more classification heads on top of CLIP's encoders would likely result in a higher accuracy, it would essentially equate the construction of a purpose-built concept classification model through transfer learning. Combining purpose-built concept classification models with symbolic reasoning in a probabilistic logic formalism has already been implemented and evaluated on GQA by Amizadeh et al. [APP+20], so we know what answer accuracy to expect. By directly fine-tuning CLIP with its original pre-training objective, we get an indication of the performance improvements for VQA that could be achieved if current VLMs added more attribute and relation captions to their training data, without any changes to the architecture of the VLM or the GS-VQA pipeline.

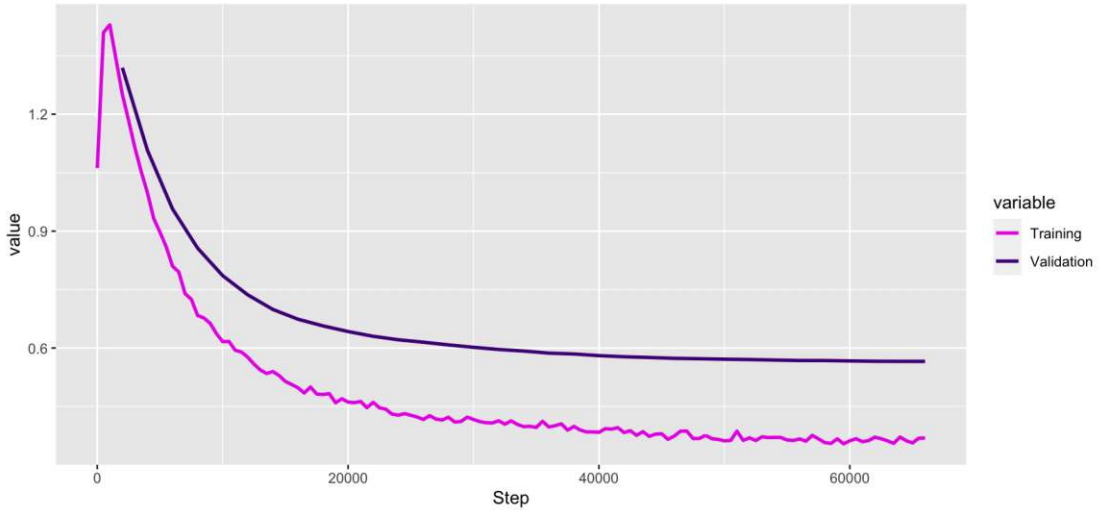


Figure 5.1: The development of training and validation loss over 3 epochs of fine-tuning CLIP (ViT-B/32)

To obtain the data for fine-tuning, we proceed similarly as in Section 4.3.2 for prompt engineering. We extract from all questions in GQA's balanced training dataset, by the same principles as followed in the Concept Extraction component, the attribute values and relations relevant to the question. For each attribute value, we randomly choose one object in the ground-truth scene graph of the image associated with the question and create an image crop from the image area under its bounding box. For relations, we similarly sample two objects from the ground-truth scene graph that are connected by that relation and obtain an image crop of the combined bounding box. Each of the resulting image crops is paired with a textual label, either "$\{a\}$ $\{c_i\}$" (e.g., "red van") for crops obtained from attributes or "$\{c_i\}$ $\{r\}$ $\{c_j\}$" (e.g., "woman driving wan") for crops obtained from relations. The result is a training dataset of 708,469 (image, text) pairs, in which the texts correctly describe either an attribute or a relation of the object(s) depicted in the image, and correspond to concepts represented in the questions of GQA's training set. We equally pre-process a random sample of 10,000 questions from GQA's validation set to obtain a small validation dataset for frequent evaluation during the fine-tuning process.

We then fine-tune CLIP on the training dataset with the same symmetric cross entropy loss that was used for pre-training the model. We use a linear learning-rate schedule with a maximum learning rate of $2 \cdot 10^{-7}$ and a warm-up ratio of 25%, and train the model for three epochs with a mini-batch size of 32. These values were obtained through manual experimentation: for higher learning rates, we observed overfitting on the training set, while lower learning rates yielded higher loss values on both training and validation set. Figure 5.1 shows the development of the training and validation loss over the course of training.

To have the tuned model perform at its best for the "target vs. neutral" prompting schema, we again evaluate the model for attribute and relation classification using multiple prompt schemas as in Section 4.3.2. Table 5.1 shows the performance improvement with the respective highest-accuracy prompt schema between the baseline and the fine-tuned model.

Concept	Type	Highest-Accuracy Schema	Acc	P	R	Acc ↑
Attributes	Base	"a blurry photo of {a/an} {a} {c_i}"	65.90%	0.662	0.650	3.05%
	FT	"a pixelated picture of a {a/an} {a} {c_i}"	68.95%	0.684	0.705	
Relations	Base	"{c_i} {r} {c_j}"	56.32%	0.553	0.659	3.52%
	FT	"{a/an} {c_i} {r} {a/an} {c_j}"	59.84%	0.578	0.733	

Table 5.1: The accuracy (Acc), precision (P), and recall (R) on the prompt-engineering datasets from Section 4.3.2 for the highest-accuracy prompt schemas of the base and fine-tuned (FT) CLIP models

5.2 Explicit Handling of Spatial Relations

As can be seen in Figure 5.2, the overwhelming majority of the most frequently occurring relations in GQA are of a spatial nature. In this extension, we take inspiration from ViperGPT [SMV23], whose LLM-based code generation maps many spatial relations to pure-Python computations on the bounding boxes of the affected objects. For the spatial relations occurring in more than 10,000 questions, we omit the "target vs. neutral" prompting used in the Concept Classification sub-component and instead directly determine the relation likelihoods from the bounding boxes of the object pairs. Since we do not have a three-dimensional representation of the objects in the scene, but only their two-dimensional bounding boxes, this is of course a stark simplification of the semantics of these relations, since they depend not only on the relative object positions in the image plane, but also on the camera viewpoint and context [JHvdM+17a]. However, given the baseline performance of CLIP for relation classification (see [ZZZ+23] and Section 4.3.2), this change might still bring an improvement to the performance of the pipeline for VQA.

We split the relevant spatial relations into two types: proximity relations (`on`, `near`, `of`, `in`, `in front of`, `behind`, `next to`, `with`) and directional relations (`to the left of`, `to the right of`, `on top of`, `above`, `below`). For proximity relations, we determine the likelihood of relation r applying between objects o_i, o_j by the ℓ_2-distance of the centers of their bounding boxes. That is, for bounding boxes $B_i = (x_i, y_i, w_i, h_i)$ and $B_j = (x_j, y_j, w_j, h_j)$ and image dimensions (i_w, i_h), we compute:

$$1 - \frac{\sqrt{((x_i + w_i/2) - (x_j + w_j/2))^2 + ((y_i + h_i/2) - (y_j + h_j/2))^2}}{\sqrt{i_w^2 + i_h^2}} \tag{5.1}$$

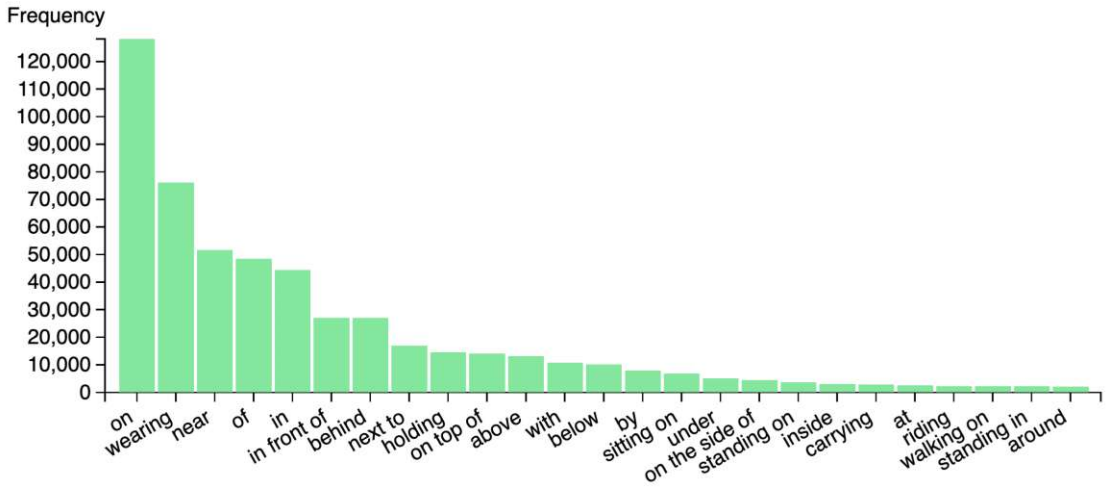


Figure 5.2: The most frequent relations in the GQA dataset, excluding `to the left/right of` (Source: [HM19, Supp. Material, Figure 1])

For directional relations, we set the likelihood of relation r applying from object o_i to object o_j to 1 if the center of the bounding box of o_i is further in the direction indicated by the relation than the center of the bounding box of o_j, otherwise to 0.

5.3 Relation Classification with LLMs

For many (o_i, r, o_j) triples, we can derive a lot of information about the likelihood of the relation holding between the two objects just from the textual representation of the relation and the objects' classes. As a human, we assign a low likelihood to the relation "a dog driving a suitcase" without ever looking at the image crop of the two objects in question. LLMs pre-trained on enormous corpora of text allow us to capture this intuitive notion of a relation between two objects "making sense" and to combine it with the multi-modal understanding of VLMs to achieve more exact relation classifications.

To obtain a numeric representation of how likely (or rather, unlikely) an LLM judges a given $(o_i,\ r,\ o_j)$ triple to apply, we use perplexity (PPL), which for auto-regressive language models is defined as the exponential of the average negative log likelihood of each token given its preceding context [Mie19], or more formally:

$$\mathrm{PPL} = 2^{-\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(x_i|x_{<i})} \tag{5.2}$$

where $x = (x_1, x_2, ..., x_n)$ is the tokenization of the input string and p_θ is the likelihood our model assigns to a token x_i given the preceding tokens $x_{<i} = \{x_1, x_2, ..., x_{i-1}\}$. For fixed-length causal language models like those based on the Transformer [VSP+17] Decoder (e.g., GPT-3 [BMR+20]), $p_\theta(x_i|x_{<i})$ has to be approximated by $p_\theta(x_i|x_{i-1}, x_{i-2}, ..., x_{i-l_\omega})$, where l_ω is the maximum context length of the model. When using perplexity as an

evaluation metric on an an entire test corpus that far exceeds the maximum context length of the model, a sliding window approach is used. For our purpose, where x is always a single sentence fragment, this does not present a problem.

For each (o_i, r, o_j) triple $(o_i = (id_i, s_i, B_i, c_i), o_j = (id_j, s_j, B_j, c_j))$ processed during relation likelihood computation in the Scene Processing component, we compute the perplexity $\mathrm{PPL}_{i,r,j}$ of the phrase "{a/an} {c_i} {r} {a/an} {c_j}" with the OPT [ZRG+22] causal language model. As with cosine similarities between CLIP-embeddings of images and text, the perplexity value of a text requires a frame of reference to judge if it is high or low for the used language model.

For this use-case, we cannot use the "target vs. neutral" prompt scheme that we employ for CLIP, since a generic relation like "and" in a neutral prompt like "a man and a car" would produce a lower perplexity in many cases even when the relation in the target prompt is actually plausible for the given objects, e.g. "a man driving a car". With any noun as the preceding context ("a man"), predicting the word "and" as the next token is a valid choice in almost all cases, so the language model will assign a relatively high likelihood to it.

Instead, we use the "target vs. ensemble" scheme: we sample n_{llm} relations from all possible relations in the dataset. For each pair of object classes c_m, c_n appearing on objects in the scene graph, we then pre-compute the perplexity of the phrase "{a/an} {c_m} {r_k} {a/an} {c_n}" for each relation r_k in the n_{llm} sampled ones, and determine the mean of these perplexities $\overline{\mathrm{PPL}}_{m,n}$. For the triple (o_i, r, o_j), we then determine the relation likelihood l_{llm} as judged by the LLM using the pre-computed mean perplexity for the classes of o_i, o_j: $l_{\mathrm{llm}} = 1 - \mathrm{softmax}(\mathrm{PPL}_{i,r,j}, \overline{\mathrm{PPL}_{i,j}})_0$. The combined relation likelihood is then $p_{\mathrm{llm}} \cdot l_{\mathrm{llm}} + (1 - p_{\mathrm{llm}}) \cdot l_{\mathrm{vlm}}$, where l_{vlm} is the original relation likelihood as determined with the approach described in Section 4.3.2, and p_{llm} is a mixing parameter.

CHAPTER 6

Evaluation

In this chapter, we perform evaluations on the GS-VQA pipeline and its extensions to answer the research questions posed in Section 1.1 in the introduction of this thesis. We first describe the modalities under which the evaluations are performed, and then discuss the results in the context of each research question individually.

6.1 Evaluation Modalities

We evaluate the GS-VQA pipeline and its extensions presented in Chapter 5 on the balanced test-dev set of GQA [HM19] ("testdev_balanced_questions.json" in version 1.2 of the GQA questions download[1]), which contains 12,578 questions. Of those, we exclude 157, or 1.248%, for the reasons given in Section 4.1. While GQA also provides a regular test set, it does not contain the semantic question representations that we take as input instead of the natural language questions. Fortunately, for this reason among others, models are commonly evaluated on the test-dev set in the literature [APP+20; SMV23; TLL+22; LLS+23; SNK+23], which allows us to put our results into context with the state-of-the-art. Like for the test set, the images in the test-dev are disjoint from those in the Visual Genome [KZG+17] dataset, ensuring that models using Visual Genome as part of their training data (like OWL-ViT [MGS+22]) do not achieve artificially good results.

All evaluation runs use the larger ViT-L/14 variant of OWL-ViT for object detection and the smaller ViT-B/32 variant of CLIP [RKH+21], which we have found to work better than the ViT-L/14 CLIP variant for our use-case during optimization on the GQA validation set. The exact values for all parameters introduced in Chapters 4 and 5 that were used during evaluation are listed in Table 6.1. If no other reason is specified

[1] https://downloads.cs.stanford.edu/nlp/data/gqa/questions1.2.zip

in the remainder of this chapter, the parameter values were obtained through manual optimization to maximize the answer accuracy on the GQA validation set.

As a host system for the evaluations, we use a workstation with an Intel Core i7-12700K CPU, 32GB of RAM, and an NVIDIA GeForce RTX 3080 Ti GPU with 12GB of video memory.

Parameter	Description	Value
t_{bbox1}	Bounding box overlap threshold for merging objects in the results of object detection model $m_{\text{detection}}$	0.6
t_{bbox2}	Bounding box overlap threshold for considering objects as duplicates in step 2 and 3 of the Object Detection sub-component	0.7
t_{bbox3}	Bounding box overlap threshold for combining object pair boxes in the relation likelihood computation of the Concept Classification sub-component	0.7
t_s	Confidence score threshold for the object detection model $m_{\text{detection}}$	0.03
k_1	Maximum number of objects returned per class in step 1 of the Object Detection sub-component	5
k_2	Maximum number of objects returned per higher-order class in step 2 of the Object Detection sub-component	5
k_3	Maximum number of objects returned in step 3 of the Object Detection sub-component	25
$m_{\text{detection}}$	Model used for the Object Detection sub-component	OWL-ViT (ViT-L/14)
$m_{\text{classification}}$	Model used for the Concept Classification sub-component	CLIP (ViT-B/32)
t_{ASP}	Time-out for the ASP solver	10s
n_{llm}	Number of sampled relations for the "target vs. ensemble" scheme used in the extension described in Section 5.3	25
p_{llm}	Mixing parameter between the LLM and VLM likelihoods of a relation in the extension described in Section 5.3	0.5

Table 6.1: Settings for the evaluations performed with GS-VQA$_{\text{base}}$

6.2 (RQ) Zero-Shot Pipeline Accuracy

To answer our main research question, *"What accuracy can be achieved on a current VQA benchmark dataset with a zero-shot neural-symbolic VQA pipeline that uses VLMs for visual perception?"*, we process the test-dev set of GQA with the base version of the GS-VQA pipeline without any of the modifications described in Chapter 5 (GS-VQA$_{\text{base}}$). The accuracy of this pipeline variant, i.e., the percentage of questions for which the pipeline generates exactly the right answer, is presented in Table 6.2 in comparison with state-of-the-art fine-tuned and zero-shot models. Note that with the exception of the

model by Amizadeh et al. [APP+20], the results are not perfectly comparable, since the other models directly take natural language questions as input, and the translation from natural language into the structured question representation that GS-VQA uses presents an additional potential source of inaccuracy.

	Model	**Category**	**GQA Accuracy**
Fine-tuned	Amizadeh et al. [APP+20]	Neural-Symbolic	51.9%
	MAC [HM18]	End-to-End	55.4%
	LXMERT [TB19]	End-to-End	60.0%
	CRF [NDT+22]	End-to-End	**72.1**%
Zero-shot	FewVLM [JCS+22]	End-to-End	29.3%
	GS-VQA$_{\text{base}}$ (Ours)	Neural-Symbolic	39.5%
	PnP-VQA [TLL+22]	Semi-Symbolic†	42.3%
	BLIP-2 [LLS+23]	End-to-End	44.7%
	ViperGPT [SMV23]	Question-Symbolic	48.1%
	CodeVQA▲ [SNK+23]	Question-Symbolic	**49.0**%

Table 6.2: Comparison of GS-VQA's accuracy on the test-dev set of GQA with that of state-of-the-art approaches for VQA

†PnP-VQA neither fully fits the "Neural-Symbolic" category, since it doesn't perform its reasoning purely symbolically, nor the Question-Symbolic category, since it extracts a symbolic representation of the image, not the question.

▲Since CodeVQA provides some expert-annotated sample programs to its code generation component, it is technically few-shot. However, since our pipeline excludes the translation of the input question into a structured representation entirely, we group it with the zero-shot models.

GS-VQA answers 39.5% of all questions of GQA's test-dev set correctly, with the current best zero-shot VQA model, CodeVQA [SNK+23], obtaining an accuracy of 49.0%. So while GS-VQA does not achieve state-of-the-art performance, it establishes the combination of VLMs and symbolic reasoning as a viable approach for zero-shot VQA. For context, we also note that CodeVQA and ViperGPT [SMV23] both translate input questions into Python code that may contain queries to another VQA model. The performance of the model used for this purpose (PnP-VQA [TLL+22] and BLIP-2 [LLS+23], respectively) should therefore be considered as their baseline. To the best of our knowledge, no other zero-shot model performing purely symbolical reasoning has been evaluated on GQA yet.

In Table 6.3, we present the accuracy of the GS-VQA pipeline and its variants on fragments of the GQA test-dev that either belong to a specific question type (binary, i.e., answered with "yes"/"no", or open), or include a specific reasoning operation. Looking at the data for GS-VQA$_{\text{base}}$, we see that the accuracy on binary questions is considerably higher than that on open questions. This result is both consistent with the literature [APP+20] and expected, since the space of possible answers for binary questions is significantly smaller. From the accuracy results of all questions containing a certain operation, we

can similarly derive the expected result that the operations with the highest number of possible outputs (*common*(): all possible attribute categories; *relate_any*(): all possible objects in the scene; *query_attr*(): all possible values for an attribute category) generally lead to the lowest accuracy values.

	GS-VQA$_{\text{base}}$	**GS-VQA$_{\text{ft}}$**	**GS-VQA$_{\text{sr}}$**	**GS-VQA$_{\text{llm}}$**
By Question Type				
All	39.50%	41.46%	40.55%	39.84%
Binary	55.13%	57.76%	56.87%	55.02%
Open	30.64%	32.23%	31.31%	31.24%
By Operation				
select()	39.50%	41.46%	40.55%	39.84%
filter()	41.56%	43.71%	42.65%	41.69%
filter_any()	40.64%	40.22%	40.92%	38.56%
relate()	36.88%	38.58%	37.82%	37.58%
relate_any()	8.59%	7.61%	8.77%	7.73%
relate_attr()	24.00%	24.00%	24.00%	26.00%
compare()	39.13%	30.43%	39.13%	47.83%
query_attr()	23.43%	23.61%	23.99%	23.43%
query_name()	33.75%	38.51%	33.64%	35.96%
verify_attr()	56.81%	61.46%	56.48%	56.06%
verify_rel()	53.84%	55.66%	62.94%	55.53%
choose_attr()	62.49%	65.70%	61.71%	62.71%
choose_rel()	49.76%	47.87%	66.35%	52.61%
exist()	55.52%	58.01%	56.85%	55.52%
two_same()	50.00%	42.48%	50.00%	49.35%
two_different()	41.30%	40.58%	41.30%	41.30%
all_same()	76.19%	80.95%	76.19%	76.19%
all_different()	60.00%	60.00%	60.00%	60.00%
common()	6.25%	14.58%	5.21%	7.29%
and()	53.62%	56.46%	53.17%	52.80%
or()	52.80%	56.03%	52.80%	52.80%

Table 6.3: The accuracy of the GS-VQA pipeline on the GQA test-dev set, shown split by question type and operation occurrence. The values stated are for the pipeline without extensions (GS-VQA$_{\text{base}}$), with fine-tuning of the Scene Processing component (GS-VQA$_{\text{ft}}$), with explicit handling of spatial relations (GS-VQA$_{\text{sr}}$), and with LLM integration for relation classification (GS-VQA$_{\text{llm}}$)

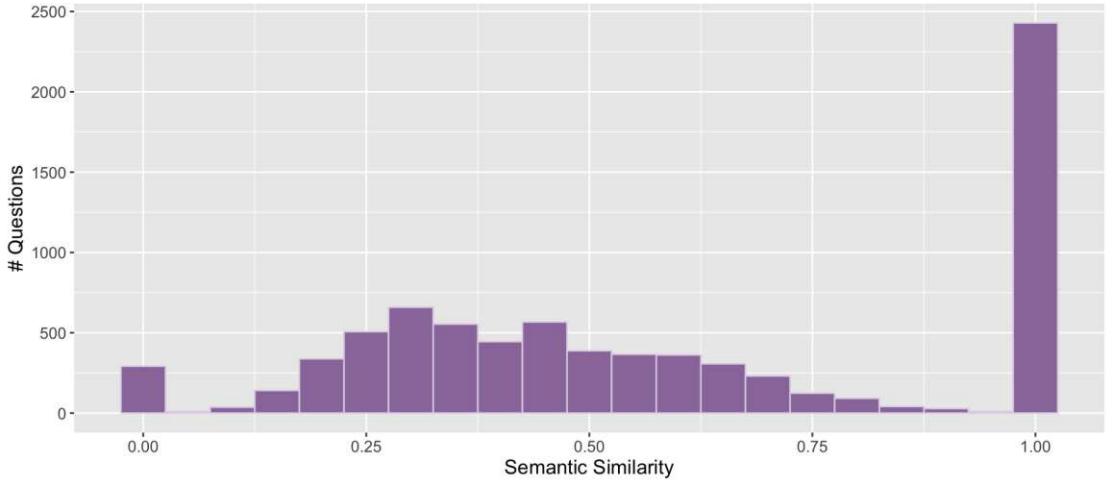


Figure 6.1: Semantic similarity (cosine similarity of MPNet text embeddings) between the official answers of GQA's test-dev set and the answers generated by GS-VQA$_{\text{base}}$

6.2.1 Answer Similarity

One limitation of the GQA dataset, in contrast to datasets like VQA v2.0 [GKS+17], is that it only contains one valid answer per question. In combination with the granularity of GQA's set of possible object classes, attribute values, etc., and the nuance of natural language, this limitation means that a model might generate answers that are not exactly equal to the official one, but semantically similar enough to be judged valid by a human. To approximately quantify this effect, we compute for each open question in GQA's test-dev set the cosine similarity between the embeddings of the answer generated by GS-VQA$_{\text{base}}$ and the question's ground-truth answer (to compute the embeddings, we use the MPNet [STQ+20] model provided by the Sentence-Transformers [RG19] library). The distribution of semantic similarity between pipeline and ground-truth answers is shown in Figure 6.1.

Of all 7,928 open questions, 2,429 have a semantic similarity of exactly 1.00: these are the 30.64% of open questions for which GS-VQA$_{\text{base}}$ gives the exact answer. For the 294 questions that the pipeline is unable to answer (and therefore generates the answer "UNSAT"), we manually set the similarity value to 0.00. The semantic similarity between pipeline and ground-truth answers for the remaining questions is widely spread between these two extremes.

Through manual inspection of the results, we determine the semantic similarity threshold above which a human would judge the generated answers as equal or at least very close to the ground-truth answer to be approximately 0.65. Similarity values below this threshold are achievable by answers that simply lie in the same category as the ground-truth one (e.g. white/blue, concrete/aluminium). One undesirable type of answer remains above this threshold however: antonyms, i.e., semantic opposites of the ground-truth answer

(narrow/wide, clean/dirty, etc.). After filtering these with the NLTK[2] Python library, we arrive at 602 questions (7.59% of all open questions) whose pipeline answers are synonyms of (small/little, purse/handbag, skateboarder/skater, etc.), close in meaning to (kitten/cat, cauliflower/broccoli, brown/beige, etc.), or more specific versions of (car/suv, table/dining table, game controller/wii controller, etc.) the ground-truth answer. We note though that a threshold on semantic similarity is not a perfect technique for separating "similar" answers from obviously wrong ones, and produces a few outliers: for example, gown/dress fall below the threshold, while office chair/desk exceed it. Together with the fact that equality in meaning is an intuitive concept, the results of this analysis should serve only as an estimate of the extent to which answers similar to the ground-truth one are generated.

6.3 (ARQ1) Runtime Performance

To answer our first auxiliary research question, *"Is the runtime performance of such a pipeline on consumer hardware suitable for human interactive use?"*, we analyze the evaluation run of GS-VQA$_{\text{base}}$ on GQA's test-dev set whose accuracy results are discussed in the previous section. The system it was performed on, as described in Section 6.1, uses only hardware readily available to enthusiast consumers. On this machine, the evaluation run of GS-VQA$_{\text{base}}$ takes 3 hours and 26 minutes. The median pipeline runtime per question is 719 milliseconds, which would be more than sufficient for using the pipeline in systems with human interaction, e.g., chat-bots or assistance systems.

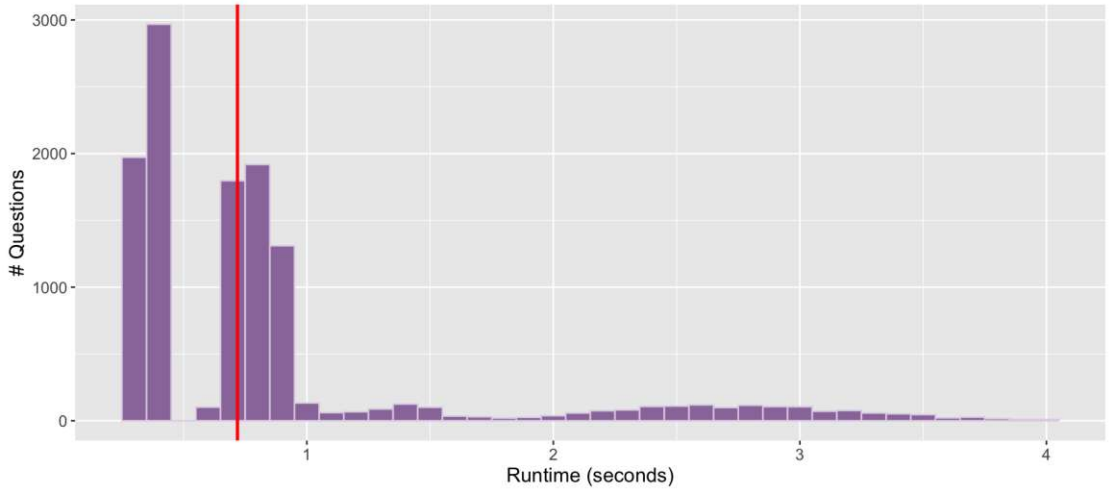


Figure 6.2: The runtimes for evaluating the questions of GQA's test-dev set; outliers with a runtime of $> 4s$ are not shown; the median on the full test-dev set (including outliers) is shown in red

[2] https://www.nltk.org/howto/wordnet.html#synsets

A histogram of the runtimes on the test-dev questions is depicted in Figure 6.2, with the median runtime highlighted in red. We have omitted 266 outliers with a runtime greater than 4 seconds from the graph, 33 of which run into the ASP solving timeout t_{ASP} of 10 seconds. Without exception, these outliers with excessive runtimes either include the *relate_any*() or the *common*() operation. This behaviour is explained by the fact that the two operations require either the detection of objects of all possible classes or the classification of all possible attribute values for each detected object. Apart from the longer processing time for the Scene Processing component, these requirements lead to significantly larger ASP Scene Encodings and thus a longer solving time. These results highlight the importance of the partial scene graph extraction of GS-VQA for making the use of zero-shot VLMs feasible, as without it, all questions would experience the same runtimes as the outliers for which the number of question-relevant classes or attributes cannot be restricted.

6.4 (ARQ2) Improved VLM Understanding of Attributes and Relations

For our second auxiliary research question, *"How much would the answer accuracy of the pipeline improve if the used VLMs were more thoroughly pre-trained for understanding attributes and relations?"*, we process GQA's test-dev set with the GS-VQA pipeline modified as described in Section 5.1 (GS-VQA$_{\text{ft}}$). We leave the architecture of the pipeline unchanged, but replace the CLIP model used in GS-VQA$_{\text{base}}$ with one that was fine-tuned on 708,469 (image, text) pairs extracted from GQA's training set. The texts in those pairs describe object attributes or relations depicted in the image. As the fine-tuning is done with the same pre-training objective as used in CLIP's original construction, and the model and pipeline architecture remain unchanged, we get an indication of the effect that improved attribute and relation understanding of the used VLM has on the actual answer accuracy of the pipeline.

From Table 5.1, we know that the fine-tuned CLIP model is 3.05%/3.52% more accurate at judging if an attribute applies to an object or a relation applies between two objects, respectively. In the results on the test-dev set shown in Table 6.3, we see this improved understanding result in a 1.96% increase in answer accuracy for the entire pipeline. This indicates that the pipeline performance scales well with improvements of the underlying VLMs used for visual perception. If future efforts on vision-language pre-training explicitly include better understanding of object attributes and relations in their training targets (as is already explored by models like X-VLM [ZZL22]), the performance of the pipeline would improve even further, as Section 5.1 showed that fine-tuning the model on a small (compared to the overall amount of training data for state-of-the-art VLMs), albeit task-specific, attribute and relations dataset still leaves a lot of room for improvement.

6.5 (ARQ3) Spatial Relation and LLM Extensions

In this section, we answer our third auxiliary research question, *"How is the accuracy of the pipeline affected by (1) the explicit computation of spatial relations between objects, and (2) the integration of LLMs to judge the plausibility of object relations?"*, by evaluating the GS-VQA pipeline with the extensions described in Sections 5.2 and 5.3.

6.5.1 Explicit Handling of Spatial Relations

From the data in Table 6.3, we see that directly determining the likelihoods for all spatial relations occurring more than 10,000 times in the GQA dataset through bounding box computations (GS-VQA$_{\text{sr}}$) yields a 1.05% increase in overall answer accuracy compared to the baseline GS-VQA$_{\text{base}}$. Looking at just those questions of GQA's test-dev set that contain at least one relation operation (*relate*(), *relate_any*(), *verify_rel*(), or *choose_rel*()), the accuracy improves by 2.05% from 32.87% to 34.92%. Including all spatial relations, and not just the most common ones, would increase this performance improvement further, but it would most likely not change it from a small, though noticeable, to a drastic one. The composition of questions into multiple reasoning operations comes into play here: Improving on just the handling of relations might mean that the right object is identified by a *relate*() operation, but if the successive *query_attr*() operation returns the wrong attribute value, the answer to the question is still wrong. In this sense, the overall performance of the pipeline is dependent on its "weakest link".

We note that we chose not to integrate the explicit handling of spatial relations into the base version of the pipeline, even though it involves no training or fine-tuning of models on the current dataset, for the following reason: In its current form, the behaviour for every handled spatial relation is hard-coded. Therefore, for evaluating the pipeline on a new dataset, the pipeline logic would have to be adjusted to cover the spatial relations that commonly occur in that dataset. To use the explicit handling of spatial relations in a dataset-agnostic manner, we could either add implementations for the finite set of all spatial relations in the corpus of the English language (which might be too large a set to be practical), or have a model tasked with the extraction of a semantic representation from the natural-language input question translate the various different wordings for spatial relations into a reduced set (e.g., left/right/above/below for directional ones, near/far for proximity ones).

6.5.2 Relation Classification with LLMs

Finally, in Table 6.3 we see that integrating LLMs as a second component to judge the likelihood of a relation applying between two objects (GS-VQA$_{\text{llm}}$) only improves the answer accuracy on the test-dev set slightly by 0.34% compared to GS-VQA$_{\text{base}}$ (or 0.47% when only questions containing a relation operation are considered). We suspect one reason for this to be the fact that the most frequent relations in GQA are largely spatial in nature (see Figure 5.2), which are plausible between almost any pair of

object classes. A second might be the quality of the LLM used: choosing a model by optimizing the accuracy achieved on the validation set of GQA, we observed models with more than 1 billion parameters (OPT-1.3b [ZRG+22], GPT-Neo 1.3b [BGW+21], etc.) achieving a $1-2\%$ higher accuracy than smaller models like GPT2 124m [RWC+19]. State-of-the-art LLMs regularly exceed 40 billion parameters (which we couldn't evaluate with our restricted hardware setup), and so could show improved results.

In its current state though, the performance gains do not justify the added architectural complexity and increased runtime of this extension. Regarding the latter point, while the median runtime increases from 719 milliseconds to 897, the mean runtime jumps from 993 milliseconds to 3,533. Since the "target vs. ensemble" scheme used in this extension requires the processing of a significant number of prompts for every combination of object classes occurring in the detected objects for the current question, questions that require a large number of object classes to be detected (or in the extreme case of *relate_any*(), all of them) are doubly affected: once by their large scene graphs, and a second time by the processing overhead induced by the LLM extension. Outliers therefore become more extreme, and the pipeline runtime less consistent.

CHAPTER 7

Conclusion and Future Work

In this chapter, we summarize the contributions of this thesis and the answers obtained for the research questions introduced in Chapter 1. We then address the limitations of our implementation, and propose topics for potential future work.

7.1 Summary

This thesis documents the design and implementation of the GS-VQA pipeline for zero-shot neural-symbolic VQA on the GQA [HM19] benchmark dataset. For visual perception, the pipeline uses the OWL-ViT [MGS+22] VLM for open-vocabulary object detection and the CLIP [RKH+21] VLM for attribute and relation understanding to build a partial scene graph of the input image, containing only those objects, attributes, and relations relevant to the input question (as determined from the question's semantic representation). We use a multi-stage approach to object detection to handle the hierarchy among object classes (e.g., "food" vs. "burger"), and "target vs. neutral" prompting to determine the likelihood of an attribute or relation applying to an object or between a pair of objects.

The partial scene graph and the semantic question representation are then encoded into an ASP formulation, making use of weak constraints to model the uncertainty of visual perception. Together with an ASP Theory formalizing the semantics of the various reasoning operations that can occur in an input question and its derived Question Encoding, the thereby created ASP program is solved with the Potsdam Answer Set Solving Collection [GKK+19], yielding the pipeline's answer to the input question.

We evaluate this implementation of the GS-VQA pipeline on the test-dev set of the GQA dataset to obtain the following answers to the research questions set out in the introduction of the thesis.

(RQ) *What accuracy can be achieved on a current VQA benchmark dataset with a zero-shot neural-symbolic VQA pipeline that uses VLMs for visual perception?*

The GS-VQA pipeline answers 39.5% of all questions of GQA's test-dev set correctly (55.13% of the binary questions, 30.64% of the open ones), with the current best zero-shot VQA model, CodeVQA [SNK+23], obtaining an accuracy of 49.0%. GS-VQA therefore does not achieve state-of-the-art performance, but establishes the combination of VLMs and symbolic reasoning as a viable approach for zero-shot VQA that can be built upon in the future.

(ARQ1) *Is the runtime performance of such a pipeline on consumer hardware suitable for human interactive use?*

In an evaluation run on the test-dev set of GQA, performed on a workstation with an Intel Core i7-12700K CPU, 32GB of RAM, and an NVIDIA GeForce RTX 3080 Ti GPU, the pipeline achieved a median runtime per question of 719 milliseconds, which would be more than sufficient for using the pipeline in systems with human interaction, e.g., chat-bots or assistance systems.

(ARQ2) *How much would the answer accuracy of the pipeline improve if the used VLMs were more thoroughly pre-trained for understanding attributes (color, shape, etc.) and relations (holding, to the left of, etc.)?*

We fine-tune CLIP on 708,469 (image, text) pairs in which the texts describe object attributes or relations depicted in the image, extracted from GQA's training set. Fine-tuning is done with the same contrastive pre-training objective as used in CLIP's original construction. This way, we are able to simulate the improvements for VQA that could be achieved if current VLMs added more attribute and relation captions to their training data, without any changes to the architecture of the VLM or the GS-VQA pipeline.

The fine-tuned CLIP model is 3.05%/3.52% more accurate at judging if an attribute applies to an object or a relation applies between two objects, respectively, using the "target vs. neutral" classification scheme employed by the GS-VQA pipeline. This improved understanding results in a 1.96% increase in answer accuracy on GQA's test-dev set when the tuned CLIP model is integrated into the otherwise unchanged GS-VQA pipeline. This indicates that the pipeline performance scales well with improvements of the underlying VLMs used for visual perception.

(ARQ3) *Making use of the compositionality of neural-symbolic VQA approaches, how can the visual perception component be modified/extended to improve the pipeline? In particular, how is the accuracy of the pipeline affected by (1) the explicit computation of spatial relations between objects, and (2) the integration of LLMs to judge the plausibility of object relations?*

Directly determining the likelihoods for all spatial relations occurring more than 10,000 times in the GQA dataset through bounding box computations yields a 1.05% increase

in overall answer accuracy compared to the base version of the GS-VQA pipeline. The accuracy on questions containing at least one relation operation improves by 2.05% from 32.87% to 34.92%. If the translation of a natural language input question into its semantic representation were implemented in such a way that all possible wordings for spatial relations are converted to a reduced set (e.g., left/right/above/below for directional relations, near/far for proximity ones), this extension could be practical to implement and provide a noticeable boost to answer accuracy.

In contrast, integrating LLMs into the GS-VQA pipeline as a second component to judge the likelihood of a relation applying between two objects only improves the answer accuracy on the test-dev set slightly by 0.34% (or 0.47% when only questions containing a relation operation are considered). These minimal performance gains do not justify the added architectural complexity and increased runtime of this extension.

7.2 Limitations

Our results come with a few limitations that we address here, along with suggestions for removing them.

Semantic question representation as input. As we focus on the visual perception and reasoning aspects of the VQA task, we directly use the semantic question representation of GQA as an input to the GS-VQA pipeline. This introduces two restrictions.

First, as already discussed in Chapter 6, it limits the comparability of the performance results of the pipeline with the results from the literature, since the translation from natural language question to semantic representation adds an additional source of errors.

Second, it constrains the real-world use or the evaluation of the pipeline on other VQA benchmark datasets that do not come with this semantic representation or a similar one. However, various works from the literature have already presented both approaches that require training [JHvdM+17b; HAR+17; YWG+18; APP+20], in some cases on only a few hundred sample questions, or zero-shot approaches [SMV23; SNK+23] that perform the translation from natural language to structured representation with high accuracy and could be adopted to remove this limitation.

Class and attribute ontology as context. To avoid inconsistent class assignments to objects and to cover open-ended operations like *relate_any*() and *common*(), the GS-VQA pipeline requires the hierarchy of possible object classes and the list of possible attribute values and their categories for the current task as context. We view this limitation as rather minor, since much of this information would be required in an open-vocabulary setting anyway (e.g., the list of classes for object detection), and the required information can be obtained far more easily than the effort required to train purpose-built models for object detection and attribute/relation classification from scratch for the task at hand. It does however limit the use of the pipeline for unconstrained, "anything goes" VQA. For

this open-ended use-case, one approach that could be pursued is using LLMs to generate the required information, e.g., sub-classes for higher-order classes, and attribute values for attribute categories.

Reasoning operations of GQA. As the GS-VQA pipeline has been developed with GQA as a benchmark dataset in mind, its supported reasoning operations are tailored to this dataset. The pipeline therefore misses some reasoning operations that one would commonly expect an intelligent agent to be able to perform in the VQA context, like counting (or mathematical reasoning in general), comprehending text visible in the input image, and drawing on external knowledge. Thanks to the modular nature of the pipeline though, many of these capabilities could be added in the future. For example, Eiter et al. [EHO+22] implement counting in their ASP encoding for the CLEVR [JHvdM+17a] dataset, where this operation is present.

7.3 Future Work

Apart from removing the limitations discussed in the previous section, we present here an outlook on further enhancements to the GS-VQA pipeline that could be explored in the future.

Newer VLMs. With the astounding pace of research in vision-language models in recent years, the OWL-ViT and CLIP models used for object detection and concept classification have already been surpassed by newer models like BLIP-2 [LLS+23], X-VLM [ZZL22], and F-VLM [KCG+23] in many vision-language understanding tasks. Adopting them into the Scene Processing component could yield more accurate partial scene graphs and thereby improve VQA accuracy.

Optimization of operations. Especially the *relate_any*() and *common*() operations cause performance issues and inaccurate answers due to the large search space that they induce. Potential improvements could involve intelligently reducing the size of the partial scene graphs by restricting the considered object classes and attribute categories, or improving the ASP encoding for faster optimization.

Further evaluation. Apart from the intuitive answer accuracy, additional evaluation modalities have been introduced in the VQA literature. GQA's consistency, validity, and plausibility metrics indicate the degree to which the model contradicts itself with the answers to related questions, produces answers in the question scope (e.g., gives any color as answer to a color question), and gives answers that "make sense" (e.g., does not answer "purple" when asked about the color of an apple). VQA [AAL+15] and VQA v2.0 [GKS+17] provide 10 answers per question from unique Mechanical Turk workers, allowing for multiple similar answers to be considered correct. Evaluating the GS-VQA pipeline under these modalities would give even more insight into its strengths and weaknesses.

APPENDIX A

Complete ASP Theory

This appendix presents the ASP theory that is discussed in Section 4.4.3 in its entirety. Listing A.1 follows the ASP syntax accepted by the tools of the Potsdam Answer Set Solving Collection [GKK+19].

Listing A.1: The complete ASP theory in Potassco ASP syntax

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

% ========== 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(A) :- end(TO), attr(TO,A).
ans(R) :- end(TO), rel(TO,R).
ans(B) :- end(TO), bool(TO,B).

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


```

```
29 % ========== Intermediary Operations ==========
30 % ---------- select ----------
31 #defined select/3.
32
33 state(TO,ID) :- select(TO, TI, CLASS), state(TI, ID), has_attr(ID, class,
       ↪ CLASS).
34
35 % ---------- filter ----------
36 #defined filter/4.
37
38 state(TO,ID) :- filter(TO, TI, ATTR, VALUE), state(TI, ID), has_attr(ID,
       ↪ ATTR, VALUE).
39
40 #defined filter_any/3.
41
42 state(TO,ID) :- filter_any(TO, TI, VALUE), state(TI, ID), has_attr(ID, _,
       ↪ VALUE).
43
44 % ---------- relate ----------
45 #defined relate/5.
46
47 state(TO, ID') :- relate(TO, TI, CLASS, REL, subject), state(TI, ID),
       ↪ has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
48 state(TO, ID') :- relate(TO, TI, CLASS, REL, object), state(TI, ID),
       ↪ has_attr(ID', class, CLASS), has_rel(ID, REL, ID').
49
50 % relate_any
51 #defined relate_any/4.
52
53 state(TO, ID') :- relate_any(TO, TI, REL, subject), state(TI, ID),
       ↪ has_rel(ID', REL, ID).
54 state(TO, ID') :- relate_any(TO, TI, REL, object), state(TI, ID), has_rel
       ↪ (ID, REL, ID').
55
56 % relate_attr
57 #defined relate_attr/4.
58
59 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), ID!=ID'.
60
61 % ---------- compare ----------
62 #defined compare/5.
63
64 state(TO,ID) :- compare(TO, TI0, TI1, VALUE, true), state(TI0, ID), state
       ↪ (TI1, ID'), has_attr(ID, _, VALUE), not has_attr(ID', _, VALUE).
65 state(TO,ID') :- compare(TO, TI0, TI1, VALUE, true), state(TI0, ID),
       ↪ state(TI1, ID'), not has_attr(ID, _, VALUE), has_attr(ID', _, VALUE
       ↪ ).
66
67 state(TO,ID') :- compare(TO, TI0, TI1, VALUE, false), state(TI0, ID),
       ↪ state(TI1, ID'), has_attr(ID, _, VALUE), not has_attr(ID', _, VALUE
       ↪ ).
```

```
68  state(TO,ID) :- compare(TO, TI0, TI1, VALUE, false), state(TI0, ID),
        ↪ state(TI1, ID'), not has_attr(ID, _, VALUE), has_attr(ID', _, VALUE
        ↪ ).
69
70
71  % ========== Terminal Operations ==========
72  % ---------- query ----------
73  #defined query/3.
74
75  { 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.
76  attr_value(TO,VALUE) :- query(TO, TI, ATTR), state(TI, ID), has_attr(ID,
        ↪ ATTR, VALUE).
77
78  % ---------- verify ----------
79  % verify_rel
80  #defined verify_rel/5.
81
82  bool(TO, yes) :- verify_rel(TO, TI, CLASS, REL, subject), state(TI, ID),
        ↪ has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
83  bool(TO,no) :- verify_rel(TO, TI, CLASS, REL, subject), not bool(TO,yes).
84
85  bool(TO, yes) :- verify_rel(TO, TI, CLASS, REL, object), state(TI, ID),
        ↪ has_attr(ID', class, CLASS), has_rel(ID, REL, ID').
86  bool(TO,no) :- verify_rel(TO, TI, CLASS, REL, object), not bool(TO,yes).
87
88  % verify_attr
89  #defined verify_attr/4.
90
91  bool(TO, yes) :- verify_attr(TO, TI, ATTR, VALUE), state(TI, ID),
        ↪ has_attr(ID, ATTR, VALUE).
92  bool(TO,no) :- verify_attr(TO, TI, ATTR, VALUE), not bool(TO,yes).
93
94  % ---------- choose ----------
95  % choose_rel
96  #defined choose_rel/6.
97  {has_rel(ID', REL, ID): has_attr(ID', class, CLASS); has_rel(ID', REL',
        ↪ ID): has_attr(ID', class, CLASS)} = 1 :- choose_rel(TO, TI, CLASS,
        ↪ REL, REL', subject), state(TI, ID).
98  rel(TO, REL) :- choose_rel(TO, TI, CLASS, REL, REL', subject), state(TI,
        ↪ ID), has_attr(ID', class, CLASS), has_rel(ID', REL, ID).
99  rel(TO, REL') :- choose_rel(TO, TI, CLASS, REL, REL', subject), state(TI,
        ↪  ID), has_attr(ID', class, CLASS), has_rel(ID', REL', ID).
100
101 {has_rel(ID, REL, ID'): has_attr(ID', class, CLASS); has_rel(ID, REL', ID'
        ↪ ): has_attr(ID', class, CLASS)} = 1 :- choose_rel(TO, TI, CLASS,
        ↪ REL, REL', object), state(TI, ID).
102 rel(TO, REL) :- choose_rel(TO, TI, CLASS, REL, REL', object), state(TI,
        ↪ ID), has_attr(ID', class, CLASS), has_rel(ID, REL, ID').
103 rel(TO, REL') :- choose_rel(TO, TI, CLASS, REL, REL', object), state(TI,
        ↪ ID), has_attr(ID', class, CLASS), has_rel(ID, REL', ID').
104
```

```
105 % choose_attr
106 #defined choose_attr/5.
107 {has_attr(ID, ATTR, VALUE); has_attr(ID, ATTR, VALUE')} = 1 :-
        ↪ choose_attr(TO, TI, ATTR, VALUE, VALUE'), state(TI, ID).
108 attr_value(TO, VALUE) :- choose_attr(TO, TI, ATTR, VALUE, VALUE'), state(
        ↪ TI, ID), has_attr(ID, ATTR, VALUE).
109 attr_value(TO, VALUE') :- choose_attr(TO, TI, ATTR, VALUE, VALUE'), state
        ↪ (TI, ID), has_attr(ID, ATTR, VALUE').
110
111 % ---------- exist ----------
112 #defined exist/2.
113
114 bool(TO,yes) :- exist(TO, TI), state(TI,ID).
115 bool(TO,no) :- exist(TO, TI), not bool(TO,yes).
116
117 % ---------- different, same ----------
118 % all_different
119 #defined all_different/3.
120
121 bool(TO,no) :- all_different(TO, TI, ATTR), state(TI, ID), state(TI, ID'),
        ↪  has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE), ID != ID'.
122 bool(TO,yes) :- all_different(TO, TI, ATTR), not bool(TO,no).
123
124 % all_same
125 #defined all_same/3.
126
127 bool(TO,no) :- all_same(TO, TI, ATTR), state(TI, ID), state(TI, ID'),
        ↪ has_attr(ID, ATTR, VALUE), not has_attr(ID', ATTR, VALUE), ID != ID
        ↪ '.
128 bool(TO,yes) :- all_same(TO, TI, ATTR), not bool(TO,no).
129
130 % two_different
131 #defined two_different/4.
132
133 bool(TO,no) :- two_different(TO, TI0, TI1, ATTR), state(TI0, ID), state(
        ↪ TI1, ID'), has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE).
134 bool(TO,yes) :- two_different(TO, TI0, TI1, ATTR), not bool(TO,no).
135
136 % two_same
137 #defined two_same/4.
138
139 bool(TO,no) :- two_same(TO, TI0, TI1, ATTR), state(TI0, ID), state(TI1,
        ↪ ID'), has_attr(ID, ATTR, VALUE), not has_attr(ID', ATTR, VALUE).
140 bool(TO,no) :- two_same(TO, TI0, TI1, ATTR), state(TI0, ID), state(TI1,
        ↪ ID'), not has_attr(ID, ATTR, VALUE), has_attr(ID', ATTR, VALUE).
141 bool(TO,yes) :- two_same(TO, TI0, TI1, ATTR), not bool(TO,no).
142
143 % ---------- common ----------
144 #defined common/3.
145
146 attr(TO, ATTR) :- common(TO, 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(TO, ATTR): is_attr(ATTR)} = 1 :- common(TO, TI0, TI1).


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

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

% or
#defined or/3.

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

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

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

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

List of Figures

List of Tables

Bibliography

[AAL+15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, et al. VQA: Visual Question Answering. In *2015 IEEE International Conference on Computer Vision(ICCV)*, pages 2425–2433. IEEE Computer Society, 2015.

[ABP16] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the Behavior of Visual Question Answering Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1955–1960. The Association for Computational Linguistics, 2016.

[APP+20] Saeed Amizadeh, Hamid Palangi, Alex Polozov, et al. Neuro-Symbolic Visual Reasoning: Disentangling "Visual" from "Reasoning". In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 279–290. PMLR, 2020.

[ARD+16] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, et al. Neural Module Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48. IEEE Computer Society, 2016.

[BBM+21] Silvio Barra, Carmen Bisogni, Maria De Marsico, et al. Visual question answering: which investigated applications? *Pattern Recognition Letters*, 151:325–331, 2021.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[BGW+21] Sid Black, Leo Gao, Phil Wang, et al. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, version 1.0, March 2021.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, 2020.

[CTJ+21] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating Large Language Models Trained on Code, 2021. arXiv: `2107.03374 [cs.LG]`.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021. arXiv: `2010.11929 [cs.CV]`.

[DCL+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, et al. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE Computer Society, 2009.

[EHO+22] Thomas Eiter, Nelson Higuera, Johannes Oetsch, et al. A Neuro-Symbolic ASP Pipeline for Visual Question Answering. *Theory and Practice of Logic Programming*, 22(5):739–754, 2022.

[FPY+16] Akira Fukui, Dong Huk Park, Daylen Yang, et al. Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 457–468. The Association for Computational Linguistics, 2016.

[GKK+12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, et al. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[GKK+19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, et al. *Potassco User Guide*. English. Version 2.2.0. University of Potsdam. January 2019.

[GKS+17] Yash Goyal, Tejas Khot, Douglas Summers-Stay, et al. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6325–6334. IEEE Computer Society, 2017.

[GMZ+15] Haoyuan Gao, Junhua Mao, Jie Zhou, et al. Are You Talking to a Machine? Dataset and Methods for Multilingual Image Question. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, pages 2296–2304, 2015.

[HAR+17] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, et al. Learning to Reason: End-to-End Module Networks for Visual Question Answering. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 804–813. IEEE Computer Society, 2017.

[HM18] Drew A. Hudson and Christopher D. Manning. Compositional Attention Networks for Machine Reasoning, 2018. arXiv: 1803.03067 [cs.AI].

[HM19] Drew A. Hudson and Christopher D. Manning. GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6700–6709. Computer Vision Foundation / IEEE, 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HZR+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE Computer Society, 2016.

[JCS+22] Woojeong Jin, Yu Cheng, Yelong Shen, et al. A Good Prompt Is Worth Millions of Parameters: Low-resource Prompt-based Learning for Vision-Language Models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2763–2775. Association for Computational Linguistics, 2022.

[JHvdM+17a] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, et al. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997. IEEE Computer Society, 2017.

[JHvdM+17b] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, et al. Inferring and Executing Programs for Visual Reasoning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3008–3017. IEEE Computer Society, 2017.

[JKS+15] Justin Johnson, Ranjay Krishna, Michael Stark, et al. Image retrieval using scene graphs. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3668–3678. IEEE Computer Society, 2015.

[KCG+23] Weicheng Kuo, Yin Cui, Xiuye Gu, et al. F-VLM: Open-Vocabulary Object Detection upon Frozen Vision and Language Models, 2023. arXiv: 2209.15639 [cs.CV].

[KCY21] Xuan Kan, Hejie Cui, and Carl Yang. Zero-Shot Scene Graph Relation Prediction Through Commonsense Knowledge Integration. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, volume 12976 of *Lecture Notes in Computer Science*, pages 466–482. Springer, 2021.

[KMR+23] Alexander Kirillov, Eric Mintun, Nikhila Ravi, et al. Segment Anything, 2023. arXiv: 2304.02643 [cs.CV].

[KPC+18] Kushal Kafle, Brian L. Price, Scott Cohen, et al. DVQA: Understanding Data Visualizations via Question Answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5648–5656. Computer Vision Foundation / IEEE Computer Society, 2018.

[KZG+17] Ranjay Krishna, Yuke Zhu, Oliver Groth, et al. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

[LGB+18] Jason J Lau, Soumya Gayen, Asma Ben Abacha, et al. A dataset of clinically generated visual questions and answers about radiology images. *Sci Data*, 5(1):1–10, 2018.

[LKB+16] Cewu Lu, Ranjay Krishna, Michael S. Bernstein, et al. Visual Relationship Detection with Language Priors. In *Computer Vision - ECCV 2016 - 14th European Conference*, volume 9905 of *Lecture Notes in Computer Science*, pages 852–869. Springer, 2016.

[LLS+23] Junnan Li, Dongxu Li, Silvio Savarese, et al. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models, 2023. arXiv: 2301.12597 [cs.CV].

[LLX+22] Junnan Li, Dongxu Li, Caiming Xiong, et al. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 12888–12900. PMLR, 2022.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, et al. Microsoft COCO: Common Objects in Context. In *Computer Vision - ECCV 2014 - 13th European Conference*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014.

[LNR+20] Weixin Liang, Feiyang Niu, Aishwarya Reganti, et al. LRTA: A Transparent Neural-Symbolic Reasoning Framework with Modular Supervision for Visual Question Answering, 2020. arXiv: 2011.10731 [cs.CL].

[LPF+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, et al. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[LYB+16] Jiasen Lu, Jianwei Yang, Dhruv Batra, et al. Hierarchical Question-Image Co-Attention for Visual Question Answering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, pages 289–297, 2016.

[MDK+18] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, et al. DeepProbLog: Neural Probabilistic Logic Programming. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, pages 3753–3763, 2018.

[MGS+22] Matthias Minderer, Alexey A. Gritsenko, Austin Stone, et al. Simple Open-Vocabulary Object Detection. In *Computer Vision - ECCV 2022 - 17th European Conference*, volume 13670 of *Lecture Notes in Computer Science*, pages 728–755. Springer, 2022.

[Mie19] Sabrina J. Mielke. Can you compare perplexity across different segmentations? April 2019. URL: `https://sjmielke.com/comparing-perplexities.htm` (visited on 08/10/2023).

[MKJ21] Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. DocVQA: A Dataset for VQA on Document Images. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2199–2208. IEEE, 2021.

[MRF+19] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, et al. OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3195–3204. Computer Vision Foundation / IEEE, 2019.

[NDT+22] Binh X. Nguyen, Tuong Do, Huy Tran, et al. Coarse-to-Fine Reasoning for Visual Question Answering. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4557–4565. IEEE, 2022.

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, 2018. arXiv: `1804.02767 [cs.CV]`.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3980–3990. Association for Computational Linguistics, 2019.

[RHG+15] Shaoqing Ren, Kaiming He, Ross B. Girshick, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, pages 91–99, 2015.

[RKH+21] Alec Radford, Jong Wook Kim, Chris Hallacy, et al. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, et al. Grad-cam: visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626. IEEE Computer Society, 2017.

[SMV23] Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual Inference via Python Execution for Reasoning, 2023. arXiv: `2303.08128 [cs.CV]`.

[SNK+23] Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, et al. Modular Visual Question Answering via Code Generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 747–761. Association for Computational Linguistics, 2023.

[SNS+19] Amanpreet Singh, Vivek Natarajan, Meet Shah, et al. Towards VQA Models That Can Read. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8317–8326. Computer Vision Foundation / IEEE, 2019.

[SR21] Arnau Martí Sarri and Victor Rodriguez-Fernandez. An Implementation of the "Guess Who?" Game Using CLIP. In *Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference*, volume 13113 of *Lecture Notes in Computer Science*, pages 415–425. Springer, 2021.

[SRM18] Nobuyuki Shimizu, Na Rong, and Takashi Miyazaki. Visual Question Answering Dataset for Bilingual Image Understanding: A Study of Cross-Lingual Transfer Using Attention Maps. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 1918–1928. Association for Computational Linguistics, 2018.

[STQ+20] Kaitao Song, Xu Tan, Tao Qin, et al. MPNet: Masked and Permuted Pre-training for Language Understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, 2020.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[TB19] Hao Tan and Mohit Bansal. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5099–5110. Association for Computational Linguistics, 2019.

[TLL+22] Anthony Meng Huat Tiong, Junnan Li, Boyang Li, et al. Plug-and-Play VQA: Zero-shot VQA by Conjoining Large Pretrained Models with Zero Training. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 951–967. Association for Computational Linguistics, 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 5998–6008, 2017.

[WWS+18] Peng Wang, Qi Wu, Chunhua Shen, et al. FVQA: Fact-Based Visual Question Answering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(10):2413–2427, 2018.

[WYY+22] Zirui Wang, Jiahui Yu, Adams Wei Yu, et al. SimVLM: Simple Visual Language Model Pretraining with Weak Supervision, 2022. arXiv: 2108.10904 [cs.CV].

[XZC+17] Danfei Xu, Yuke Zhu, Christopher B. Choy, et al. Scene Graph Generation by Iterative Message Passing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3097–3106. IEEE Computer Society, 2017.

[YHG+16] Zichao Yang, Xiaodong He, Jianfeng Gao, et al. Stacked Attention Networks for Image Question Answering. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21–29. IEEE Computer Society, 2016.

[YIL20] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural Networks into Answer Set Programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1755–1762. ijcai.org, 2020.

[YLL+18] Jianwei Yang, Jiasen Lu, Stefan Lee, et al. Graph R-CNN for Scene Graph Generation. In *Computer Vision - ECCV 2018 - 15th European Conference*, volume 11205 of *Lecture Notes in Computer Science*, pages 690–706. Springer, 2018.

[YWG+18] Kexin Yi, Jiajun Wu, Chuang Gan, et al. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, pages 1039–1050, 2018.

[ZOW+22] Rufai Yusuf Zakari, Jim Wilson Owusu, Hailin Wang, et al. VQA and Visual Reasoning: An Overview of Recent Datasets, Methods and Challenges, 2022. arXiv: 2212.13296 [cs.CV].

[ZRG+22] Susan Zhang, Stephen Roller, Naman Goyal, et al. OPT: Open Pre-trained Transformer Language Models, 2022. arXiv: 2205.01068 [cs.CL].

[ZZL22] Yan Zeng, Xinsong Zhang, and Hang Li. Multi-Grained Vision Language Pre-Training: Aligning Texts with Visual Concepts. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 25994–26009. PMLR, 2022.

[ZZZ+23] Tiancheng Zhao, Tianqi Zhang, Mingwei Zhu, et al. VL-CheckList: Evaluating Pre-trained Vision-Language Models with Objects, Attributes and Relations, 2023. arXiv: 2207.00221 [cs.CV].