Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Evaluation Environment for Vision-Based Relational Systems

Scientific Students' Association Report

Author
Petra Huszti

Advisor
dr. Oszkár Semeráth

November 1, 2022

Contents

Kivonat

A képfeldolgozáson alapuló gépi tanulás egyre nagyobb teret hódít; az egészségügytől kezdve az önvezető járművekkel bezáróan számos területen előszeretettel alkalmazzák. Ezzel számos új lehetőség és kihívás tárul a mérnökök elé. Megjelentek továbbá Mesterséges Intelligencián (MI) alapuló komponensek biztonságkritikus rendszerekben (például önvezető járművekben) is, ahol a hibák életekbe kerülhetnek, ezért kiemelten fontos a tesztelés és a megbízhatóság elemzése.

Számos tesztelésre és teljesítménymérésre képes környezet létezik a képfeldolgozó MI-k képességeinek értékelésére, azonban gyakran kell szembesülünk azzal, hogy módszereik részrehajlók és képesek anélkül magas pontosságú eredményeket elérni, hogy teljesítenék a tényleges követelményét. Ebből kifolyólag olyan mérési eredménnyel szolgálnak, ami nem tükrözi az MI tényleges képességeit, gyengeségeit.

A probléma elkerülése érdekében szükség van tehát egy olyan kiértékelési környezetre, ami képes pártatlan teljesítménymérést végezni számos képfeldolgozó MI-n. Ezáltal olyan mérési környezetet javaslunk, amelyben összehasonlíthatóvá válnak különböző gépi tanulás modellek, és ezek ellenőrzési/tesztelési módszerei.

A beszámoló egy olyan újszerű módszert mutat be, ami bemeneti adatokat generál képfeldolgozó MI-k számára, amit ezek után fel lehet használni a különböző MI modelek és ellenőrzési technikák tesztelésre és tanítására. Az elkészült keretrendszer célja, hogy az MI-k képességeit több különböző statisztikai és diverzitás metrika szerint legyen képes mérni. Mintaillesztés felhasználásával képes tehát megfelelően sokrétű, releváns kérdéseket generálni és ellenőrizni a rájuk adott választ.

A fentebb említett környezet segítségével lehetőség nyílik arra, hogy számos MI teljesítményét mérjük, ezeket összehasonlítsuk, erősségeket és hiányosságaikat összegezzük.

Abstract

Recently, vision-based machine learning applications are getting more and more attention; they are increasingly applied to multiple domains from healthcare to autonomous vehicles. This presents many novel opportunities and challenges to engineers. As many of those AI-based components operate in safety-critical environments, where errors can harm human life, the evaluation and testing are high priorities.

Many testing and benchmark environments already exist to measure vision-based AI performance, however they often use biased methods for evaluation, where high accuracy can be achieved without satisfying their requirements. Therefore, giving an incorrect assessment of the AIs capabilities.

To avoid this, we need to create an environment that is capable of serving as a non-biased benchmark for a multitude of vision-based AI components. The goal of the report is to present the measurement environment that I have created.

In this report I propose a novel way of generating input data for vision-based AI to evaluate the testing and training performance of different AI models and verification approaches. I have developed a cross-platform environment that aims to measure the capabilities of an AI, while avoiding forming biases. It is capable of generating appropriately varied, relevant questions with the usage of pattern matching and giving verified answers to them.

With the use of the aforementioned environment, it becomes possible to measure the capabilities of different AIs, compare them, assess their strengths and weaknesses.

Chapter 1

Intoduction

1.1 Context

Vision-based AI is becoming more and more prevalent in a multitude of domains. Many of these AI-based components already used in a safety-critical environment; we could take a look at the newest car on the market or at expensive medical equipment. In these systems a single error can cost a life or a lot of money. This is the point where the evaluation and testing of these systems become vital. Through evaluation and testing we can discover fault and understand the limitations and capabilities of the AI, gaining the ability to assign them correctly or improve on them.

1.2 Problem Statement

Despite the large amount of existing testing and benchmark environments already exists, the testing of those macine learning models is still an open question. There are several measurement environments that are capable of providing us with measurements of artificial intelligence (AI) models, but their methods are often *biased* and miss **diversity** to achieve high coverage metrics. Therefore, in the context of visual-reasoning AI this means that the AI models can achieve relatively high accuracy without solving the problems rare (but relevant) corner-cases. For example, for a question "is there a pedestrian left to the car in front of this car" the typical answer is "no", an model can achieve high accuracy with a constant "no" answer, but it is critical to give the correct answer in case of a takeover maneuver for self-driving cars. Thus an evaluation environment is needed which can systematically generate questions that are either *statistically relevant*, or follow some *coverage metrics* to discover important corner cases.

1.3 Objectives

To battle the formation of biases we require an environment that discourage the formation of them. The *first goal* of this report is to present the environment, that can serve as a non-biased benchmark for a multitude of vision-based AI components. The *second goal* of this report is to provide a cross-domain benchmark environment, so multiple solutions from different domains can be compared with each other, or existing benchmarks or test cases can be adapted to another domain.

This report focuses on visual question answering AI models, with special focus on relational reasoning.

1.4 Contributions

In this report I propose a novel way of generating input data for vision-based AI to evaluate the testing and training performance of different AI models and verification approaches.

- I have developed a cross-domain environment that aims to measure the capabilities of an AI, while avoiding forming biases.
- I adapted the generation environment for the synthetic domain of Clevr [17] and autonomous vehicle domain Carla [9, 10].
- I provided performance benchmarks to assess the feasibility of my question generation approach.

1.5 Added Value

With the use of the aforementioned environment, it becomes possible to measure the capabilities of different AIs, compare them, assess their strengths and weaknesses. Moreover, with different realistic and coverage metrics, the highly accurate performance and efficient test suites can be generated.

1.6 Struture of the Report

The structure of this report is the following. In the 2. Chapter I discuss the necessary background knowledge for evaluating vision-based AIs. In Chapter 3 we will take a look at the overall concept of the work, like how a verification environment is structured, and what is required from it. In Chapter 4 we discuss the workings of the algorithm behind the environment. Chapter 5 presents the measurement results produced by the environment. Finally, Chapter 6 will conclude this report.

Chapter 2

Preliminaries

This chapter discusses the preliminaries necessary to understand the contributions of this report.

2.1 Vision-Based Relational AIs

Vision-Based Relational Artificial Intelligeces belong to the field of computer vision application that allow machines to analyze images and provide an output based on their observations.

2.1.1 Machine Learning techniques

Machine Learning (ML) techniques are widely used for automatically assessing large datasets where it is difficult or infeasible to provide traditional software components. The field of machine learning is diverse and ever-expanding with multiple different approaches and algorithms to use [19]. A generic taxonomy of machine learning approaches is illustrated in Figure 2.1. In this report, we are focusing mainly on neural networks.

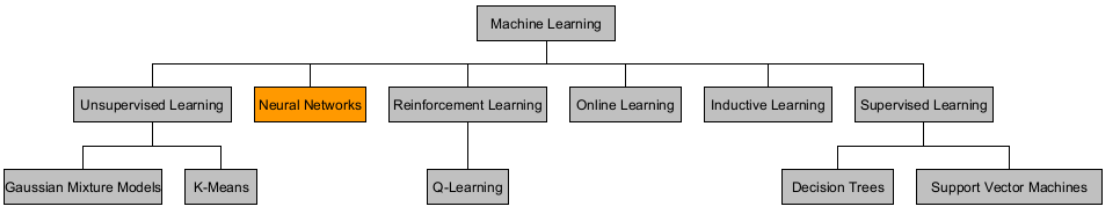

Figure 2.1: Machine Learning Types and Algorithms

Neural Network [1] is a family of algorithms that emulates the workings of neurons of the brain signaling to each other. Nowadays, a wide variety of neural network algorithm is proposed for various tasks, which can be efficiently trained and operated using the computational capacities of modern graphical processors [25].

The use of neural networks allow computers to recognise patterns and solve problems. As illustrated in Figure 2.2, they are composed of an input and output layer separated by one or more hidden layers. Each layer contains nodes, that act as artificial neurons, connected by weighted edges to higher level nodes. If a node activates then information is only allowed to travel on edges with a lower threshold then the value of the output[3].

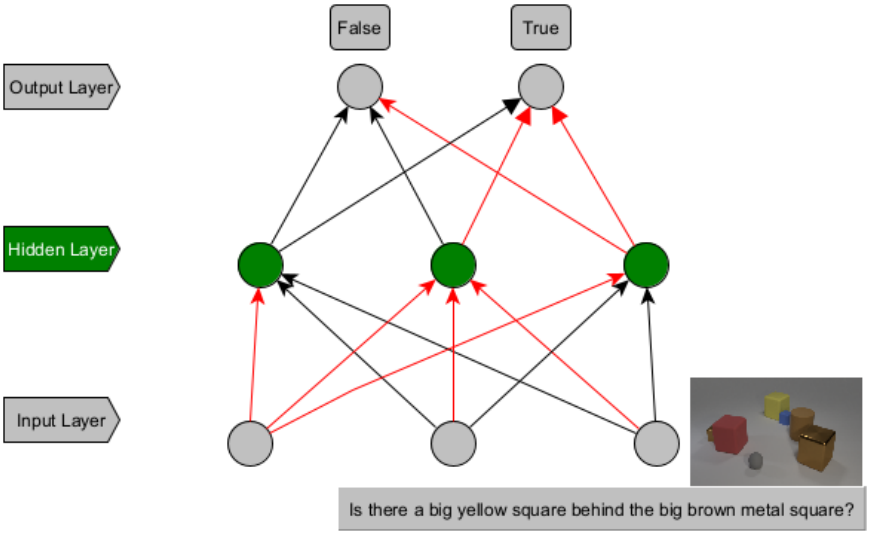

Figure 2.2: Example neural network with three inputs in its input layer, a hidden layer with three neurons, and two neurons in it output layer.

Deep Learning[23, 16] is a subset of neural networks. Deep learning approaches are widely used when dealing with a large scale input data with abundant training data available for the neural network to obtain a higher level of accuracy. This is the main reason it is used when the need to process images, videos, large texts or audio data arises.

In the field of computer vision an AIs main task is to analyse pictures, videos or other visual media in order to make decisions. AI models are constructed to process input from cameras and sensors, with the final goal of providing higher accuracy than humans [18]. From the basic ability of recognising objects or detecting movement to navigating abstract concepts relating to sight, many task are being delegated to AIs.

A *Convolutional Neural Network* is a deep learning algorithm, commonly used for image processing and other tasks involving pixel data. As the field of computer vision mainly relies on images, CNN is the most favoured algorithm used. It consists of the following layers:

- convolutional layer, that is the core of CNN. The kernel or filter inside this layer (or layers) moves through the fields of the images and decides if a feature is present in it.
- pooling layer, like the convolutional layer sweeps through the image, but with reduced parameters.
- fully connected layer where the classification of the image happens.

CNN-based solutions are frequently used in image processing components of autonomous vehicles [11]. The algorithm receives images made by a camera on the dashboard and outputs the steering angle directory. This algorithm is sufficient to control the steering wheel of an autonomous vehicle and allows for training in simulators .

2.1.2 Relational reasoning

Relational AIs are using machine learning in combination with graph-structured or relational data. AIs are generally associated with making rational choices, but in reality, they are susceptible to biases and exclusions. From this susceptibility derives the need for the relational AI, where not only the given answer matters, but the correct reasoning behind them as well. For this purpose CNN provides an interesting choice which can combine reasoning with complex data processing like object detection.

2.2 Datasets for vision-based relational models

Deep learning requires a large amount of training data, that would take a long time to produce manually, therefore it is recommended to use an automatic method to generate the input.

2.2.1 Clevr

Clevr is "a diagnostic dataset for studying the ability of Visual Question Answering systems to perform visual reasoning" [18]. This dataset contains:

- Images to be used as input data
- Questions related to the images.

The task of the AI is to answer these questions correctly. The benchmark provides tools for generation these images and the corresponding questions with a given distribution.

The images (as illustrated in Figure 2.3) contain three to ten objects each, chosen from the three available shapes (sphere, cube, and cylinder), in two different materials (metal and rubber). There are also two sizes (small and big) and eight colors to choose from. All objects on the images can be described with the use of these four parameters.

The questions present are capable of assessing aspects of visual reasoning such as attribute identification, logical operations, comparisons, counting and multiple attention. They are separated into 90 different families with each family containing at least 4 synonymous templates.

> **Example 1.** *Lets check an example image/question pair generated by Clevr:*
>
> - ***Image:*** *illustrated in Figure 2.3.*
> - ***Question:****"Are there any big yellow rubber cube things?"*
>
> *[17] provides an prototype model solution created by Facebook Research for visual question answering. In this example, their prototype solution can give the correct answer, which is* ***"true"*** *(as there is a big yellow rubber cube on the top right corner).*

2.2.2 Carla

Carla [9] is an open-source simulator with open digital assets, like buildings, maps and vehicles. It provides a wide range of utilities relating to the modification of the simulation

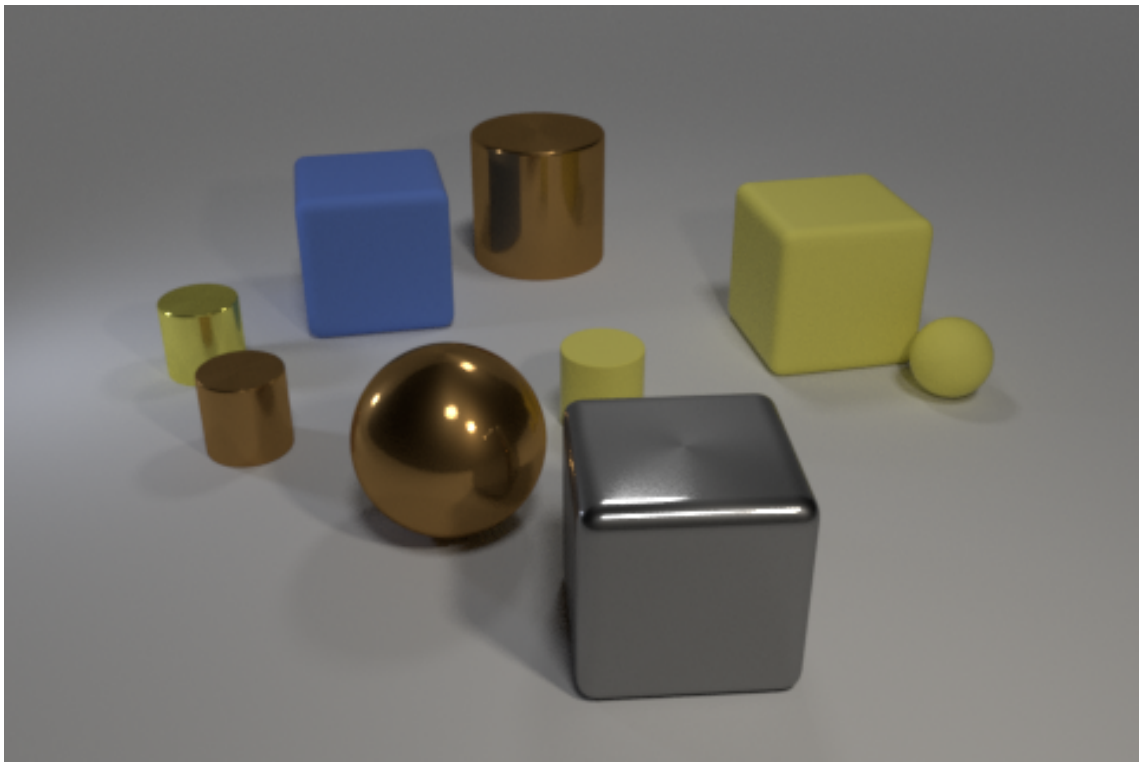

Figure 2.3: Generated image for Clevr

(ex.: generating scenarios, changing the weather, monitoring sensors). Moreover, Carla is closely linked with testing self driving cars, including open source self driving AI models like OpenPilon [2, 8, 26].

In this report, Carla will serve as the simulator of choice for generating the input dataset of an AI operating in an autonomous vehicle. Carlas model for describing scenes is illustrated on Figure 2.4

2.2.2.1 Scenic and traffic situation generation

Scenic [10] is a domain-specific probabilistic programming language for modeling the environments of cyber-physical systems like robots and autonomous cars.

Scenic allows for the creation of complex and rare traffic situations. The environment, the placement, attributes and behaviours of vehicles can be described in a manually written code. The placement of vehicles are chosen based on the required road or crossroad, therefore from one code, multiple vastly different situations can arise.

> **Example 2.** *Code fragment Figure 2.5 we can observe the description of a behaviour and a vehicle. The behaviour receives the speed of the vehicle. The vehicle is to follow its designated* `lane(FollowLaneBehaviour)` *until an other objects enters its safety distance, when it has to break with the intensity of 1.*
>
> *The description of the vehicle includes the lane (that is picked at random from the available lanes) and a spot (the centre of the lane). The vehicle is to spawn following the direction of the lane, within the specified range from the spot, have the above mentioned behaviour, with the specified model.*

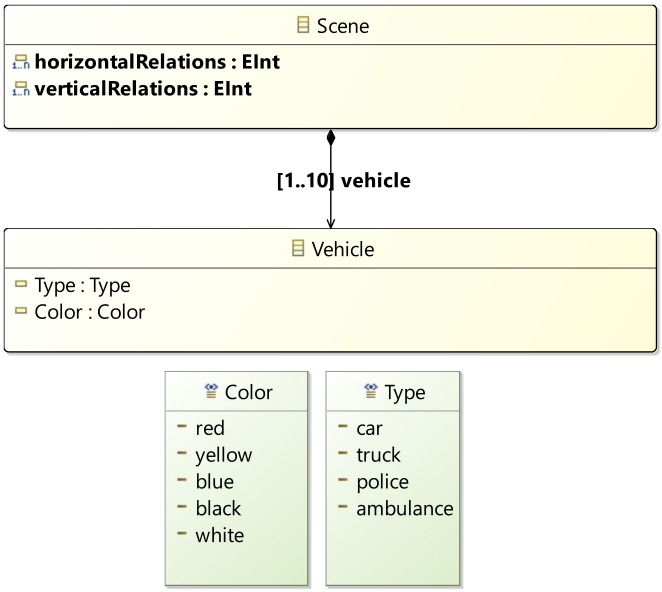

Figure 2.4: Model for describing scenes for Carla

> *We can observe that the lane and spawnpoint of the vehicle is chosen at random from a number of possible choices allowing for a large number of permutations.*

Figure 2.6 is made using scenic to describe the situation that we wanted the generate. If we compare it with the image we have received from the Clevr Dataset (illustrated in Figure 2.3), we can observe, that the relevant objects in the scenes are comparable to each other. An object in Clevr is a vehicle in Carla, with each having their own attributes to describe them.

2.3 Modeling environments

Finally, the report presents the modeling background for describing questions and scenes for various domains.

2.3.1 Eclipse Modeling Framework

Model Driven Development is a distinct way to approach software design and development, using models for a more concise system. The MDD approach separates the functional and technical aspects of development. It allows for models to exist independently from a platform, therefore creating a platform-free interpretation of the program. These models are a great way of defining systems requirements from the get-go and can be used to generate at least parts of the code.

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model [24].

EMF makes use of the MDA approach and combines it with old-school code writing. The models created with EMF can be used to generate parts of the code, but it is necessary to write the more complicated functions ourselves. In the EMF model one can find the data structure of the application; objects attributes, their relationships and the functions that are available to be executed on them.

```
SAFETY_DISTANCE = 30
BRAKE_INTENSITY = 1.0
EGO_MODEL = "vehicle.tesla.model3"
EGO_SPEED = 10

behavior EgoBehavior(speed=10):
    try:
        do FollowLaneBehavior(target_speed=speed)
    interrupt when withinDistanceToObjsInLane(self, SAFETY_DISTANCE):
        take SetBrakeAction(BRAKE_INTENSITY)

lane = Uniform(*network.lanes)
spot = OrientedPoint on lane.centerline

ego = Car following roadDirection from spot for Range(-30, -20),
    with blueprint EGO_MODEL,
    with behavior EgoBehavior(EGO_SPEED)
```

Figure 2.5: Carla's Generated Scenario

> **Example 3.** *On the picture we can observe the model of the images (illustrated on Figure 2.4) that can be generated by Carla. We see, that these images – represented by the Scene Class – contains the following items:*
>
> - *vehicles: the image contains one to ten vehicles, each having a color and a type.*
> - *verticalRelations: the vehicles ordered, starting with the closest to the camera*
> - *horizontalRelations: the vehicles ordered, starting from the left side of the image*
>
> *We can also take a look at the generated class of Vehicles(illustrated on* **??***) where we can see, that the generation automatically created a number of functions for the class, mostly getters and setters, but we can see a toString method as well.*

2.3.2 Xtext

Xtext is a framework for development of textual languages and domain-specific languages [6].

Xtext grants programmers the ability to create their own languages, with its own parser, linker, compiler and all the other tools required from a complete language infrastructure.

> **Example 4.** *Let's take a look at the following template in Clevr:* ***"Are there any <Z> <C> <M> <S> things?"*** *where <Z> is size, <C> is color, <M> is material and <S> is shape.*
>
> *If we bind the parameters 'big', 'yellow', 'rubber' and 'square' to <Z> <C> <M> <S> we can create the question* ***"Are there any big yellow rubber square things?"****.*

Figure 2.6: Carlas Generated Scenario

> *We can also bind nil to any of the parameters. If <C> is nil, then we get the more general question of* ***"Are there any big rubber square things?"****.*

2.4 Related Work

2.4.1 Visual Question Answering Benchmarks

Clevr proposed a visual question answering benchmark witch is the main motivation of this report [17]. This Clevr benchmark was replicated several times in the literature including solutions from Facebook Research [28], Google Deepmind [21], OpenAI [20]. Blocksworld [4] is a similar benchmark with vertical relations (up and down) instead of horizontal (front, behind).

The main downside of those approaches is that they generate questions by random sampling witch can produce highly biased AI models. For example, in [18] the AI model is presented with over 60% accuracy, witch does not even get the image as an input. This can be explained by the high amount of 0/false answers in the dataset, witch is the safest answer in all the cases.

2.4.2 Autonomous Vehicles

The testing of autonomous vehicles is more beneficial if it is done with the use of simulators, as it is cheaper and more safe. One of the simulators to be used for this purpose is CARLA [9]. In Carla we can generate the rare and critical scenarios (scenarios that could end in an accident), or even the combination of the two. For this purpose we use a tool, that easily describes a traffic situation [10].

The most common way to operate and train an autonomous vehicles is with the use of deep learning [11], more specifically Convolutional Neural Networks. They have already been used in LIDARS [12] and ADAS systems [22], or training end-to-end controllers [29]

There are a variety of more complex CNNs in literature.

2.4.3 Neural Networks

Neural networks are widely used in the filed of computer vision, a number of more advanced approaches have appeared in literature recently, especially at concerning the relational AIs. Take the Recurrent neural networks(RNN) [27] or Graph Neural Networks(GNN) [13] as an example for of a specialized neural network for assessing graph structured relational data.

Chapter 3

Overview of the Approach

The goal of this report is to extend the Clevr benchmark into a domain independent performance evaluation framework for vision-based relational artificial intelligence models. Therefore different AI models can be objectively compared, else the visual representation of the domain can be evaluated separately from the relational properties. Moreover existing benchmarks for Clevr can be adapted to autonomous driving domains.

For the running example I have chosen a question from the Clevr Dataset and its modified version from Carla.

Clevr version:

"What is the *material* of the *big yellow square* that is *behind* the *big gray metal square*?"

Carla version:

"What is the *color* of the *car* that is *behind* the *red truck*?"

3.1 Functional Overview

In the following, I represent the functional overview of my approach (illustrated in Figure 3.1). I have created an evaluation environment for artificial intelligence models.

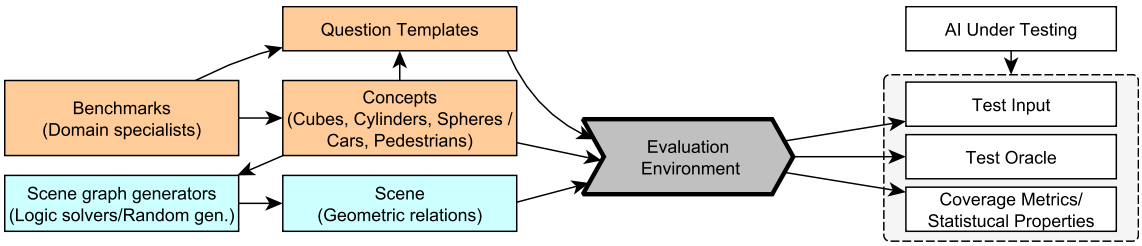

Figure 3.1: Overview of the approach.

First, test generation is initiated from a benchmark.

Benchmark

A benchmark environment for vision-based relational AI models defines image formats and (logic) reasoning task. Benchmark environments are constructed by domain experts to capture relevant aspects of real life problems.

Concepts

Concepts describe the possible types of actors, their spatial relations and attributes on the image. Moreover more complex derived concepts can be derived.

In this report we will use vision-based AI in traffic situations. In these instances the following concepts are crucial to be understood:

- **Actor**: Different types of vehicles and any other being that can actively participate in traffic. The paper uses vehicles as it's actors.
- **Attributes**: All of the attributes belonging to the actors. The paper uses color (e.g., black, white, red) and type (e.g., car, ambulance).
- **Relations**: These are the terms that are applicable to the relationship between actors; these are 'left', 'right', 'behind' and 'in front of'.

With the extension of the benchmark the concepts can be extended as well. The current concepts are illustrated on Figure 3.2 using UML class diagrams.

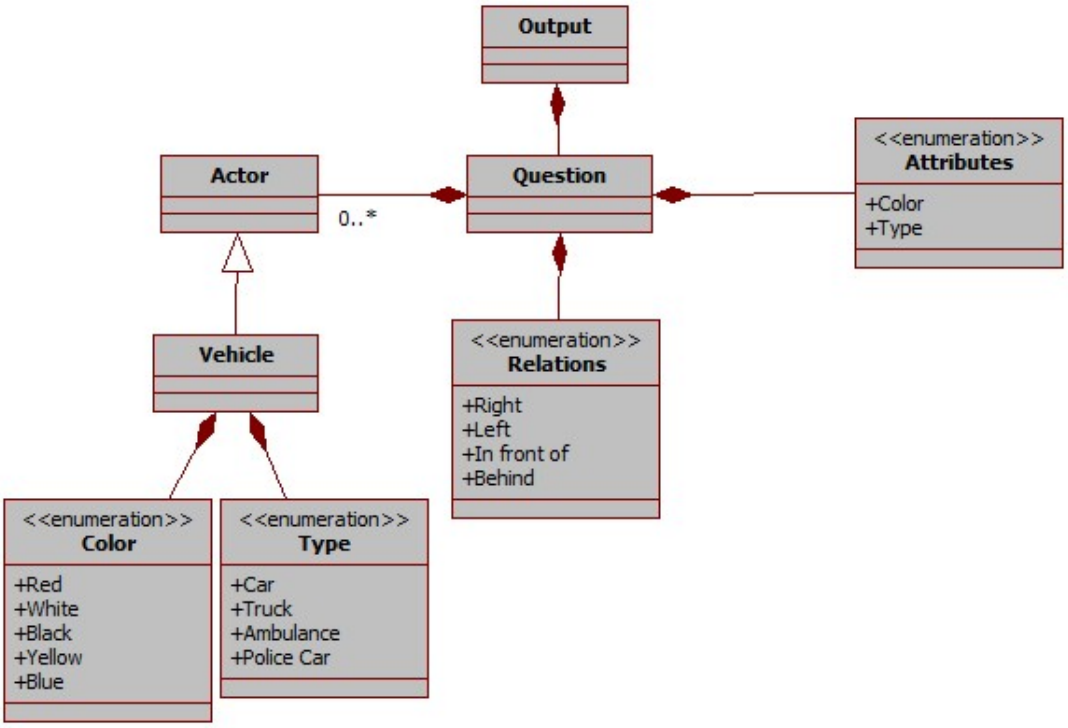

Figure 3.2: Class Diagram of the Concepts

Scene Graph Generators(SGG)

An SGG is a tool capable of producing graph structured data using the available concepts.

SGGs include statistical and probabilistic methods [10] or logic solvers like [5, 7].

Scene

A scene is the description of a picture containing all the necessary information for an AI to answer all the questions asked of it, and no more.

In our case a scene includes all the relevant attributes of objects and their positions on the image, described related to each other.

To every pictures belongs a single scene, yet a scene can be used to illustrate an infinite number of pictures. With the ability to describe every image relating to our case, it can be said that the scene object is sufficient for deducing the correct answer to a visual question answering problem.

A scene must contains the following items:

- Objects: this is a list of the present actors on the image. There are three to ten object in each scene.
- Attributes: to every object belongs a series of attributes.
- Relationships: this is the ordered list of the actors on the image. It contains a horizontal sublist, that has the actors ordered from left to right and the depth sublist, that has them ordered from the closest to the camera to the farthest. These sublist contain only the indexes of the actors, derived from the Objects.

The scene

> **Example 5.** *As an example of a scene, let's look at the following images in Figure 3.3. As we take a look at the scene object on the bottom figure, we can tell the name of the image is "Test_1.png". It has three objects; a* ***red car****, a* ***black car*** *and a* ***black police car****. The relations shows that the* ***red car*** *is the first from the left and the* ***black police car*** *is the last. It also tells, that the* ***black car*** *is the closest to the camera and the* ***red car*** *is the farthest from it.*
>
> *Let's say, that we have moved the black car to the right by two pixels. These two pixels create such a such a small difference, that the overall order of the objects remain unchanged. This way we can understand, that the scene shown above can belong to a multitude of different images.*

Question Templates

Question templates are text templates with variables. These variables are bound to different objects or attributes relating to the scene.

The generation of questions is based on preexisting question templates. These templates are heavily inspired by Clevrs interpretation, but modified to fit autonomous vehicles.

Each question consists of terminal and non-terminal parts. The terminal parts are basis for separating questions into categories for easy mix-and-match. The non-terminal parts can be considered as blank spaces, that can only be filled by the correct type of variable, chosen from the concepts section. The question templates are illustrated on Figure 3.4.

> **Example 6.** *Let's take the following questions:*
>
> *Is the number of* ***big yellow matte square****s the same as the number of* ***big blue matte square****s?*

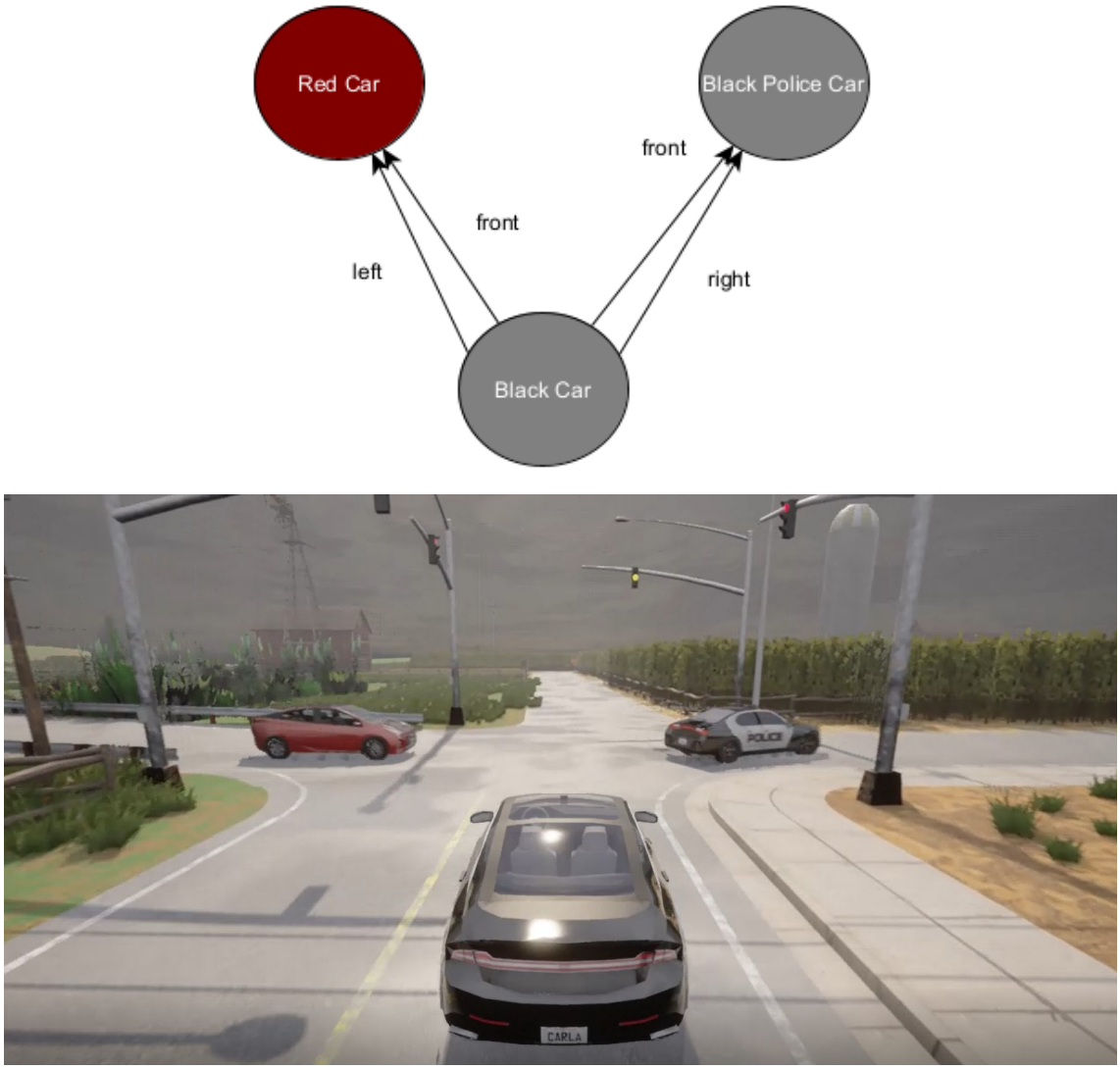

Figure 3.3: A graph created from the scene and it's real picture

> *Is the number of **black car**s the same as the number of **red car**s?*
>
> *The first version of the question is created with Clevrs template, the second is with Carlas. In both templates the terminal and non-terminal parts are easy to differentiate as the parameters are distinct from the rest of the text and it is clear on both versions what kind of variable should be bound to the parameters.*

3.2 Objects Generated by the Environment

Test Inputs
A collection of questions generated for a set of images.

One of the generators main purpose is to create test inputs. In Carlas case, this happens using the available question templates, scenes and concepts. Every test input contains a set of one hundred questions by default that are verified to be relevant to the picture.

```
WhatIs:
        'What' 'is' 'the' adjective=AttributeName 'of' 'the' subject=Attributes 'object' ('?' |continue=WhatIsExtended)
;
WhatIsExtended:
        'that' 'is' relation=Relation 'of' 'the' object=Object ('?' | continue=(SecondRemoved |TwoPointsOfReference))
;
SecondRemoved:
        'that' 'is' relation=Relation 'the' object=Object ('?' | continue= ThirdRemoved)
;
ThirdRemoved
        'that' 'is' relation=Relation 'the' object=Object '?'
;
TwoPointsOfReference
        'that' 'is' 'both'  relation=Relation 'the' object=Object 'and' relation2=Relation 'the' object2=Object '?'
;
```


```
[{"text": ["Are there an equal number of <Z> <C> <M> <S>s and <Z2> <C2> <M2> <S2>s?", "Are there
the same number of <Z> <C> <M> <S>s and <Z2> <C2> <M2> <S2>s?", "Is the number of <Z> <C> <M>
<S>s the same as the number of <Z2> <C2> <M2> <S2>s?"], "nodes": [{"inputs": [], "type": "scene"},
{"side_inputs": ["<Z>", "<C>", "<M>", "<S>"], "inputs": [0], "type": "filter_count"}, {"inputs": [],
"type": "scene"}, {"side_inputs": ["<Z2>", "<C2>", "<M2>", "<S2>"], "inputs": [2], "type":
"filter_count"}, {"inputs": [1, 3], "type": "equal_integer"}], "params": [{"type": "Size", "name": "<Z>"},
{"type": "Color", "name": "<C>"}, {"type": "Material", "name": "<M>"}, {"type": "Shape", "name":
"<S>"}, {"type": "Size", "name": "<Z2>"}, {"type": "Color", "name": "<C2>"}, {"type": "Material",
"name": "<M2>"}, {"type": "Shape", "name": "<S2>"}], "constraints": [{"params": [1, 3], "type":
"OUT_NEQ"}]}
```


Figure 3.4: Clevr's question Template(bottom) and my question template(top)

The number and sampling of these questions can be changed depending on the desired outcome.

> **Example 7.** *Clevrs generates the questions to multiple images at the same time into the same file, as it is indicated by the indexing of both pictures("image_index") and the questions("question_index").*
>
> *Carlas version generates large amount of questions to one image at the time, without indexing, but separated by a new line.*

Test Oracle

A collections of correct answers to the asked questions.

In Carlas case this set contains the correct answer in the form of a number or text. These answers are produced alongside the questions, written into separate files per image.

Using the test oracle one can calculate the accuracy of the AI model under test as the number of correctly answered questions are divided by the number of questions. However accuracy is not a sufficient metric in safety critical domains [15, 14], therefore it is critical to formulate a different coverage metrics to ensure the testing if critical and dangerous situations.

```
Is there another cyan cube of the same size as the big cube ?
Is there a small gray rubber sphere that is behind the small yellow rubber sphere that is behind the gray cube ?
Is there a yellow cylinder ?
Is there another small brown cylinder of the same shape as the yellow metal sphere ?
Is there another brown rubber cylinder of the same color as the small blue metal cylinder ?
Is there a blue cylinder ?
Is there a sphere ?
Is there a big green rubber cube that is right the small blue metal cube that is left the big purple cube ?
```

```
How many blue cars are front the cyan ambulance that is behind the purple car ?
Is there a purple ambulance ?
Are there fewer blue ambulance that are behind the blue car than gray ambulance that are behind the purple car ?
Is there a green ambulance that is right the blue car that is front the gray ambulance that is front the purple ambulance ?
What is the type of the car object ? The cyan car that is both behind the green ambulance and behind the green car is what type ?
What number of gray car are both right the brown ambulance and front the cyan car ?
What is the type of the brown object ?
Is the number of purple ambulance that are behind the red ambulance greater than the number of green car ?
```

Figure 3.5: Generated Clevr and Carla questions

Coverage Metrics

Coverage metrics give data on the software during testing as they measure the percentage of potential questions and scenes.

In Carlas case the coverage metrics are able to show the touched upon questions. It can be used later to measure the type and complexity of the questions that were answered correctly and those that were not, therefore giving an insight into the workings of the AI. With this knowledge further question generation can be adjusted based on the performance of the AI.

3.3 Verification and Benchmark Tasks

The final goal of the AI is to be able to answer all questions that it is trained to answer correctly. If we want to achieve such a goal, then we have to also make sure that the environment it is trained in only gives correct answers to all questions. That is the point where verification of both questions and answers are necessary.

3.3.1 Verification of the Questions

One of the goals of question verification is to give only situationally relevant, complete questions.

Situationally Relevant Questions

We call a question situationally relevant if the question only features actors, attributes and relations that exist on the picture.

Example 8. *Let's take the image in Figure 2.6 and its scene object. On this image we have a* ***red car*** *left of a* ***black car*** *that is left of a* ***black police car****.*

In the context of these objects the question ***Are there any red cars left of the black car?*** *is a relevant question as there is both a red and a black car present on the picture. Note that the question in this instance is relevant even if we use right instead of left, as relevance does not mean that the answer has to be positive.*

On the other hand a question like ***Are there any red cars left of the black car that is right of the red firetruck?*** *is not relevant, because the is no red firetruck on the image. If we substitute the red firetruck with* ***black police car*** *the question still remains irrelevant, because there is no black car right of the black police car.*

Complete Questions

We call a question complete if all of their simple variables are accounted for and at least one variable is present from each of their complex variables.

Example 9. *Let's take a look at the following question:* ***What is the <adjective> of the <actor> that is <relation> the <actor>?****. This question has to meet the following requirements to be considered complete:*

- *The adjective variable is simple, therefore it has to have a value and cannot be nil.*
- *The relation variable is simple, therefore it has to have a value and cannot be nil.*
- *The actor variables are complex, made up from a <color> and <type> part. At least one of these variables have to have a value.*

The other goal is to generate numerous questions, without repeating one.

3.3.2 Benchmark Tasks

Using the proposed evaluation environment several benchmark tasks can be adapted to self-driving (Celvr) domains from Carla. First training set for self driving cars are frequently enhanced by simulated scenarios. However large amount of simulated training data can hinder the accuracy of the AI model because it seems it overadapts to the images. Performance benchmarks are frequently used to evaluate the accuracy of the AI model, where a large number of synthetic image-question pairs are required. Simulated environments are widely used in testing, as it can detect weaknesses of the model. The main difference between performance benchmarks and testing is that performance bench-

marks require a realistic distribution of test inputs while testing typically is motivated by corner cases.

	Clevr	Carla
Training	+	+/-
Testing	+	+
Performance benchmark	+	-

Table 3.1: Clevr and Carla tasks

One of the main benefit of my approach is that it can be parameterized by different stochastic distributions to follow the real life distributions of questions. Moreover given a coverage metric questions can be generated to increase the coverage of a test suite.

Realistic Generation

Given a statistical metric, a generated set of questions is realistic if it follows the distribution of questions in the training set.

Diverse Generation

Given a coverage metric, a generated set of questions is diverse if each question increases the coverage metric of the test suite (ideally 100%).

Chapter 4

Elaboration

This chapter discusses the details of the process responsible for generating the questions.

4.1 Question Templates in Xtext

In this section we discuss the implementation of the Question Templates.

4.1.1 Concepts

The generation of questions relating to traffic situations required the creation of a new grammar, specifically to describe traffic situations with ease. The grammar has been name carlaQuestions, that comes with its own extension cq.

The grammar contains all of the concepts mentioned in Chapter 3(illustrated in **??**).

- Actors are represented by the only actor that is present at this stage of the development: the Object class, that describes a vehicle.
- There are two attributes present in the program: Colortype and Cartype, both of which are part of the Object class.
- Spatial Relations are implemented by the Relations class. As by the original notion it is a complete class, as it has all the elements discussed in Chapter 3(Figure 3.2).

The interpretation also contains the AttributeName class for avoiding unnecessary template duplication, whenever the inquired upon attribute changes. This class collects the 'names' of all attributes that can exist related to any of the actors.

4.1.2 Question Template

The template collection has a number of question families of various sizes. All questions featured are only allowed to have terminal parts written in free-text and non-terminal parts chosen from the above mentioned concepts.

The questions are categorized based on their terminal parts, as they repeat often thus it is necessary to reuse as much of it as possible to reduce the size of the templates. As it

```
WhatIs:
        'What' 'is' 'the' adjective=AttributeName 'of' 'the' subject=Attributes 'object' ('?' |continue=WhatIsExtended)
;
WhatIsExtended:
        'that' 'is' relation=Relation 'of' 'the' object=Object ('?' | continue=(SecondRemoved |TwoPointsOfReference))
;
SecondRemoved:
        'that' 'is' relation=Relation 'the' object=Object ('?' | continue= ThirdRemoved)
;
ThirdRemoved
        'that' 'is' relation=Relation 'the' object=Object '?'
;
TwoPointsOfReference
        'that' 'is' 'both'  relation=Relation 'the' object=Object 'and' relation2=Relation 'the' object2=Object '?'
;
```

Figure 4.1: The abstraction of the WhatIs class

stands there are twelve families in the collection, one of which we are going to take a closer look at.

We can see, that the terminal parts of the question are highlighted in green and the non-terminal parts in orange: these are the variables that are the substituted. There are only one type of variable that is able to fill one spot. If the need arises for one blank to be able to be filled by two different variable, we can simple create a class, that is capable of achieving that goal.

The family illustrated on Figure 4.1 is the WhatIs family. By its name we can already imagine, that it is a collection of question templates starting with *"What is"* and ending in the following ways:

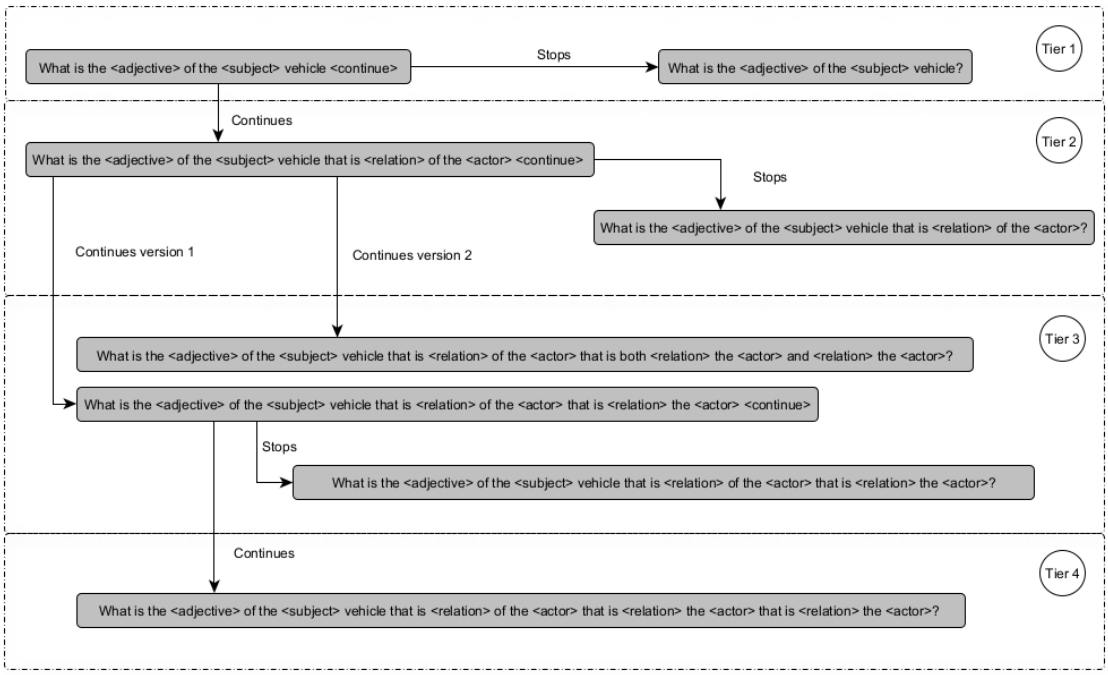

Figure 4.2: WhatIs family tree

On Figure 4.1 we can see, that the possible endings of the question are separated by a | symbol. This allows for two possible variables to be inserted; one is the **?** that ends a question, the other one is the continue variable, that takes moves the question a tier lower. This behavior forms the above illustrated tree.

On the first tier we can see the shortest question(s) templates that the family is able to produce. These templates are the easiest to answer as they have to least amount of variables to take into account and do not require the understanding of **AND** operators as some of the following templates. The deeper we go in the tree, the higher the tiers numbering becomes, we encounter more challenging questions, with more logical operators and more variables.

> **Example 10.** *Let's take a look at possible tiers of questions the WhatIs class can produce. The easiest questions come from tier 1, where it must have an adjective and a subject:* ***What is the color of the car?****. On this tier the only thing that the AI has to accomplish is attribute identification.*
>
> *If we move down a tier, then we must extend the question with another actor from the scene and a relation:****What is the color of the car that is left of the red truck?****. On this tier the AI has to be able to identify attributes and understand the spatial relation between objects.*
>
> *Another tier down one of the possible questions challenge the AI to understand the* ***AND*** *operator while the other one focuses on the use of multiple* ***that is*** *keywords. The former presents the* ***What is the color of the car that is left of the red truck that is both left of the yellow truck and right of the blue car?*** *question and requires multiple attribute recognition and spatial recognition. The latter produces the* ***What is the color of the car that is left of the red truck that is in front of the yellow bus?*** *question and requires the AI to shift the subject of the question to another actor.*
>
> *Further down we can find more uses of the* ***that is*** *keyword.*

4.2 Test Generator in Java

In this section we discuss the three available methods of question generation, starting from the simplest.

4.2.1 Random Question Generation

The simplest way of all generation is to create completely random items.

Illustrated in Figure 4.3 is the short description of the function generating the questions. It relies on the generateRandomQuestion function to give complete questions, collects them in a list, then saves to a file. Illustrated in Figure 4.4 the generateRandomQuestion function. This functions purpose is to choose a template family to work with. This can be selected entirely at random with the use of random number generation, but can be defined manually as well.

> **Example 11.** *The following example shows the generation of the question:* ***What is the color of the car that is behind the red truck?****.*
>
> *This question belongs to the* ***WhatIs*** *family. The example assumes that the family has been chosen for generation.*

```
START
questions: contains the list of generated questions
questionNumber: contains the number of questions to generate

For from 0 to questionNumber
        Call generateRandomQuestion -> q
        Add q to questions
For Ends
Save questions to file
END
```

Figure 4.3: Pseudocode of the generation of random questions

```
START
questionFamily: one of the family(class) of questions

Choose questionFamily
Call the familys generation function -> question
Return question
END
```

Figure 4.4: Pseudocode of the generateRandomQuestion Function

> *As illustrated on Figure 4.2, the family tree shows that we have to generate the following items to achieve a complete question:*
>
> - *On Tier 1: the adjective(attribute) and subject of the question. For the continue variable we must choose Continue. This is done in the function responsible for Tier 1 question generation of the family(illustrated on Figure 4.5).*
> - *On Tier 2: the relationship and another actor of the scene. For the continue variable we must choose Stops. This is done in the function responsible for Tier 2 question generation of the family(illustrated on Figure 4.6).*

4.2.2 Generation of Unique Questions

A more advanced method for question generation is to allow the creation of a set of different questions.

Illustrated in Figure 4.7 is the short description of the function responsible for generating different questions. Like its predecessor it also relies on the generateRandomQuestion function to give complete questions, however in each of its iterations it will also check if the question has already been generated, in witch case the function will throw it away.

Due to the nature of the questions, it is impractical to reach the text itself, therefore another solution was required to assess the uniqueness of the questions. This method led to the creation of the Set class. The Set class contains the question itself and the unique number combination identifying each question. The number combination is created by collecting all the random numbers during the generation of the question.

START
attribute: the asked upon attribute of the actor
actor: the subject of the question
extended: the extension of the question
question: a WhatIs class variable

Initialize *question*
Call generateAttribute -> *attribute*
Call generateActor -> *actor*
Call generateExtension -> *extension*
Bind *attribute*, *actor* and *extension* to *question*
Return *question*
END

Figure 4.5: Pseudocode of the generateRandomQuestion Function

START
relation: the relationship between the subject and the other actors
actor2: an actor of the scene
extension: a WhatIsExtended class variable

Call generateActor -> *actor2*
Call generateRelation -> *relation*
Bind *actor* and *relation* to *extension*
Return *extension*
END

Figure 4.6: Pseudocode of the generateRandomQuestion Function

4.2.3 Generation of Unique Questions With Answers

The most advanced method for generating questions is to create situationally relevant and complete questions with the anticipated answers attached.

Illustrated in Figure 4.8 is the short description of the function responsible for generating both the questions and the answers. This relies on the generateRelevantQuestions function.

This also required an answer to be added to each question. The easiest solution to the problem was to extend the Set class with an answer variable.

```
START
questions: contains the list of generated questions
questionNumber: contains the number of questions to generate
existing: contains the unique identifier of the already generated questions
set: contains a question and its identifier
generated: tells if an identifier already exists

While questionNumber is not reached
        Call generateRandomQuestion -> set
        Check if the identifier of set is already in exisiting -> generated
        If generated is false
                Add question of set to questions
                Add identifier of set to existing
        If Ends
While Ends
Save questions to file
END
```

Figure 4.7: Pseudocode of the generation of different questions

```
START
questions: contains the list of generated questions
questionNumber: contains the number of questions to generate
existing: contains the unique identifier of the already generated questions
set: contains a question, its identifier and the answer to the question
generated: tells if an identifier already exists
answers: contains the generated answers to the questions

While questionNumber is not reached
        Call generateRandomQuestion -> set
        Check if the identifier of set is already in exisiting -> generated
        If generated is false
                Add question of set to questions
                Add answer of set to answers
                Add identifier of set to existing
        If Ends
While Ends
Save questions to file
Save answers to file
END
```

Figure 4.8: Pseudocode of the generation of different questions with answers

Chapter 5

Evaluation

This chapter evaluates the environment discussed in chapter 4.

RQ1 What is the performance of the proposed approach?

RQ2 Is there a performance difference between different domains?

RQ3 What is the performance cost of generating diverse questions?

5.1 Selected domains

For the measurement includes the domains discussed throughout the report: Carla and Clevr. Both environments have the same definition of relation and attributes, but differ in the use of actors. Carlas actors consist of two different attributes, while Clevrs consists of four.

5.2 Measurement setup

The evaluation was conducted using a computer with the following specifications:

- **CPU**: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
- **Memory**: 8 GB
- **Operating System**: 64 bit Windows 11

The measurements were executed in java environment with the following specifications:

- **JRE**: JavaSE-11
- **VM Memory**: 4 GB

Each measurement were repeated 30 times and we collected the median runtime of the generation. We measured the generation time and the serialization of the questions separately. To take the warmup effect into account, the I run the measurement 10 more times.

On both domains we conducted similar measurement, with an increasing amount of questions to be generated. The following graphs show the results of Carla and Clevr, for the generation of 100, 1 000, 10 000 and 100 000 questions in seconds, with different samplings techniques.

- Random sampling: the generation of questions without discarding duplicates
- Focused sampling: the generation of questions without discarding duplicates, but manually choosing the family to generate from
- Different sampling: the generation of question with discarding the duplicates

5.3 Measurement results

Illustrated on Figure 5.1 and Figure 5.2 is the measurement results for the Focused sampling of Carla and Clevr respectively. On the x axis we can see the number of questions generated, while on the y axis the time that elapsed during the generation. The darker part of columns are the time spent with generation, while the lighter part is the time it took to write the questions to file.

The measurement results of the Focused sampling is illustrated on Figure 5.3 and Figure 5.4 respectively, while the results for the Different sampling are portrayed on Figure 5.5 and Figure 5.6. Both the x and y axis and the columns represent the same type of measurement as they do in the Random sampling.

Note that that number of questions on the x axis increment exponentially.

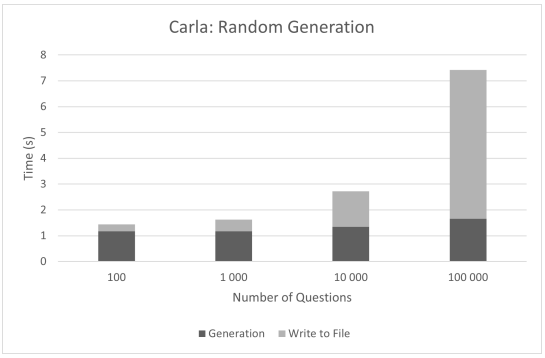

Figure 5.1: Carla

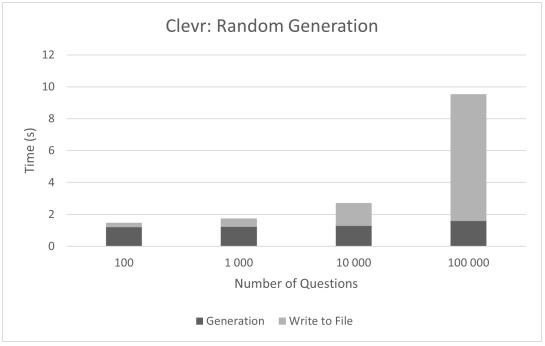

Figure 5.2: Clevr

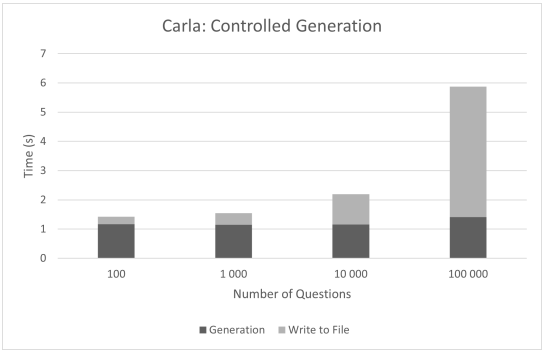

Figure 5.3: Carla

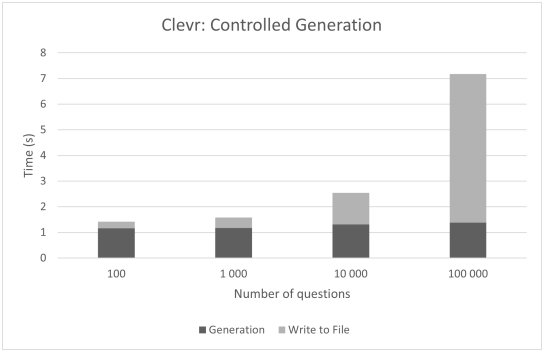

Figure 5.4: Clevr

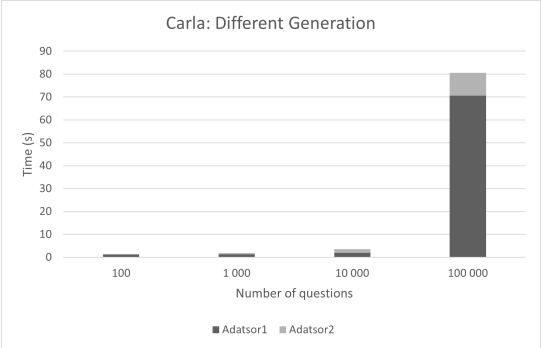

Figure 5.5: Carla

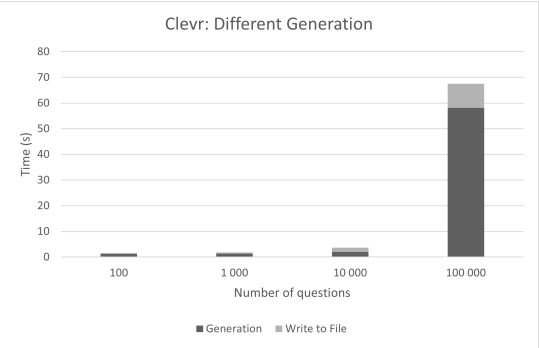

Figure 5.6: Clevr

The following diagrams present the comparison between Carla and Clevr using different samplings. On Figure 5.7 we can see the comparison using Random sampling, on Figure 5.8 using focused sampling and on Figure 5.9 using Different sampling.

On all diagrams the x axis represents the number of questions generated, the y axis is the time spent. With a yellow line we can see time it took Clevr to generate the questions, while the green line represents Carla. A black line shows how long it took for Clevr to write the questions to file, while a blue line shows Carla.

Note that that number of questions on the x axis increment exponentially.

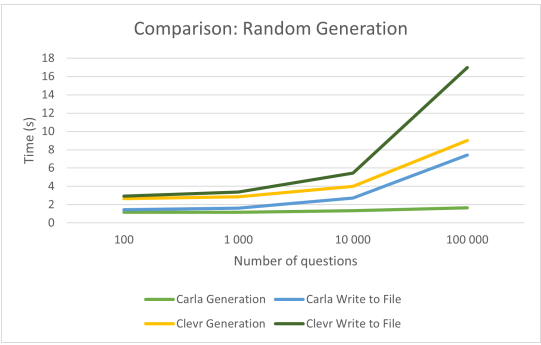

Figure 5.7: Random Gen.

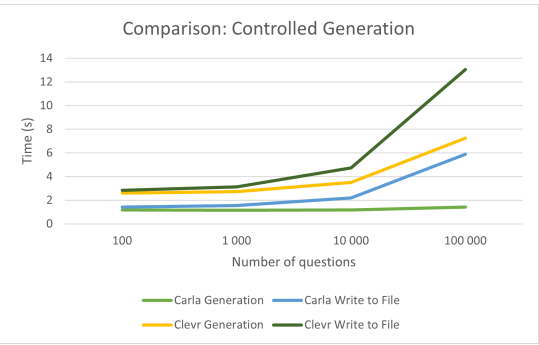

Figure 5.8: Controlled Gen.

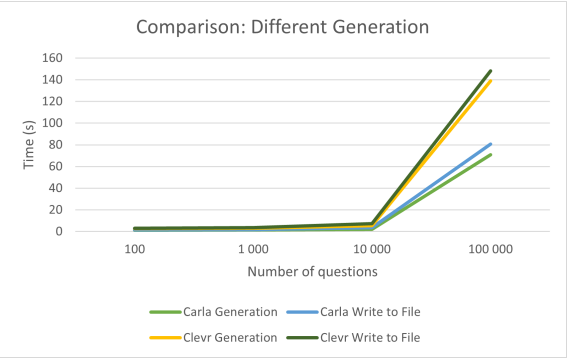

Figure 5.9: Different Question Generation

5.4 Analysis of the results

What is the performance of the proposed approach?

The runtime of question generation is mostly independent to the nunmber of questions. The time to write to file increases linearly, which dominates the runtime for large number of questions. However, the generation of 10.000 questions (which is the typical numer of questions in Clevr) is still relatively low (less then 2 minutes).

Is there a performance difference between different domains?

The generation of questions are largely the same between the two domains, the small time increase of the Clevr domain can be attributed to the number of attributes per actor is double.

What is the performance cost of generating diverse questions?

The time increase resulting from the generation of different questions are significant compared to the random generation, as the algorithm starts to run out of possible questions to generate (and it needs to look for more unique questions). However the time spent is still within acceptable margins.

Chapter 6

Conclusions and Future Works

In this report I made the following architectural contributions:

- I adapted the generation environment for the synthetic domain of Clevr [17] and autonomous vehicle domain Carla [9, 10].

I have made the following implementations:

- I have developed a cross-domain environment that aims to measure the capabilities of an AI, while avoiding forming biases.
- I provided performance benchmarks to assess the feasibility of my question generation approach.

Bibliography

[1] Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

[2] Faisal S Alsubaei. Reliability and security analysis of artificial intelligence-based self-driving technologies in saudi arabia: a case study of openpilot. *Journal of advanced transportation*, 2022, 2022.

[3] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[4] Masataro Asai. Photo-realistic blocksworld dataset. *arXiv preprint arXiv:1812.01818*, 2018.

[5] Aren A. Babikian, Oszkár Semeráth, Anqi Li, Kristóf Marussy, and Dániel Varró. Automated generation of consistent models using qualitative abstractions and exploration strategies. *Softw. Syst. Model.*, 21(5):1763–1787, 2022.

[6] Lorenzo Bettini. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[7] Boqi Chen, Dylan Havelock, Connor Plante, Michael Sukkarieh, Oszkár Semeráth, and Dániel Varró. Automated video game world map synthesis by model-based techniques. In *MoDELS (Companion)*, pages 4:1–4:5. ACM, 2020.

[8] Li Chen, Tutian Tang, Zhitian Cai, Yang Li, Penghao Wu, Hongyang Li, Jianping Shi, Junchi Yan, and Yu Qiao. Level 2 autonomous driving on a single device: Diving into the devils of openpilot. *arXiv preprint arXiv:2206.08176*, 2022.

[9] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[10] Daniel J Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: A language for scenario specification and data generation. *Machine Learning*, pages 1–45, 2022.

[11] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS research*, 43(4):244–252, 2019.

[12] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231, 2018.

[13] Martin Grohe. The logic of graph neural networks. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–17. IEEE, 2021.

[14] Florian Hauer, Alexander Pretschner, and Bernd Holzmüller. Fitness functions for testing automated and autonomous driving systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 69–84. Springer, 2019.

[15] Florian Hauer, Tabea Schmidt, Bernd Holzmüller, and Alexander Pretschner. Did we test all scenarios for automated and autonomous driving systems? In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2950–2955. IEEE, 2019.

[16] Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic Markets*, 31(3):685–695, 2021.

[17] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, abs/1612.06890, 2016.

[18] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[19] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[21] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

[22] Yusuf Satılmış, Furkan Tufan, Muhammed Şara, Münir Karslı, Süleyman Eken, and Ahmet Sayar. Cnn based traffic sign recognition for mini autonomous vehicles. In *International Conference on Information Systems Architecture and Technology*, pages 85–94. Springer, 2018.

[23] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[24] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[25] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011.

[26] Meriel von Stein and Sebastian Elbaum. Finding property violations through network falsification: Challenges, adaptations and lessons learned from openpilot. 2022.

[27] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5410–5419, 2017.

[28] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 670–685, 2018.

[29] Shun Yang, Wenshuo Wang, Chang Liu, and Weiwen Deng. Scene understanding in deep learning-based end-to-end controllers for autonomous vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):53–63, 2018.