University of Trento
Department of Industrial Engineering
Master's Degree in Autonomous Systems

~ · ~

Academic Year 2023–2024

Automatic Generation Of Formal Specifications Using Ontology For Verification Of CNN Based Object Detection Modules

Supervisor
Prof. Daniele Fontanelli, *UniTn*
Prof. Marco Roveri, *UniTn*
Vivek Poovalingam, *ThoughtWorks*

Graduate Student
Kalaiselvan Thangaraj
239405

Final examination date: July 17, 2024

Acknowledgments

Completing this master thesis has truly been a transformative journey for me, one that would not have been possible without the support and guidance of many individuals.

First and foremost, I would like to express my deepest gratitute to Mr. Vivek Poovalingam, principal consultant at ThoughtWorks and ThoughtWorks India for giving me a wonderful opportunity to work on an interesting problem in the automotive industry. I would like to express my deepest gratitude to my advisors, Prof. Daniele Fontanelli and Prof. Marco Roveri for their valuable guidance, support and feedback throughout the process. Last but not least, I would like to thank my family and friends for supporting me in all my endeavours.

Abstract

Autonomous Vehicles (AV) promise to revolutionize road transport by enhancing safety, reducing congestion and increasing mobility options. Achieving fully autonomous (level 5) vehicles is challenging because of the lack of formal verification tools for verifying neural network based object detection modules. The thesis addresses this concern by presenting an automation tool capable of generating formal specification and performing formal verification of AI-based object detection modules. The proposed tool uses ontologies to generate semantically meaningful formal specifications which reduces errors and increases consistency in elicitating specifications. The effectiveness of the tool is evaluated by verifying the zero-shot CLIP classifier on classes from the RIVAL10 and GTSRB datasets. The results shows that the tool can be effective in generating specifications without syntactic and semantic errors and is able to verify object classification by the zero-shot classifier. It also identifies the features or concepts the model has learned which had led to those classifications. This work advances the field of automation tools for formal verification of AI-based perception components and assists in the homologation process of autonomous vehicles.

Keywords: Software Defined Vehicles, Formal Verfication and Validation, Trustworthy AI, AI Verfication and Validation, VLM, CNN, Automation Tool, Automotive Safety Standards, SOTIF, .

Contents

Introduction

Autonomous Vehicles (AV) are vehicles designed to autonomously drive through the complex scenes of city streets and highways. AVs are said to revolutionize the road transport industry by enhancing the safety of its passengers, reducing traffic congestions by performing optimally and by providing alternative mobility options for people who can't drive conventional automobile. Autonomous vehicles have not yet become fully autonomous. Currently available vehicles are only designed to drive in specialized areas called as their Operational Design Domains (ODD). The autonomy these vehicles can have on their respective ODD's like highways, city streets etc., are expressed as SAE automation levels ranging from level 0 (no automation) to level 5 (complete automation). Level 5 vehicles are supposed to manage the Dynamic Driving Task (DDT) and take care of the fallback by itself without any human input. Current stage of AVs in production are form level 3 to level 4[34][1].

However, recent incidents in the autonomous driving industry indicate that it is far from achieving the goal of level 5 autonomy. Eventhough it is said that 90% of the Autonomous Driving Problem is solved, it is very hard to solve the remaining 10% of the problem before reaching level 5 autonomy. Automotive industry is well known for their regulations. Every automobile present in the road has to undergo rigorous safety standards check before they are allowed to drive on the roads. There are standards like *ISO26262[41]* to ensure functional safety in vehicles. However, with the recent involvement of AI in successful developement of AVs, assessing the AI performing the control or perception tasks has become very hard.

Neural networks, which are a class of deep learning methods is attributed to the rapid development of autonomous vehicles, for their contrbution in perception tasks. Neural networks are developed using a different approach than the conventional software development. Traditional softwares are developed by providing detailed set of instructions to perform a function or a task. Different algorithms or techniques can be used to achieve a similar functionality. In the development of neural networks, we let the network to learn the algorithm by itself. We only provide examples of input data and annotate them with their correct classes. This shift in the software development approach from traditional coding to development of Learning Enabled Components (LEC) have made the existing techniques for safety standard checks incompatible.

Standards like ISO 26262 ensure the safety of the hardware and software components by enforcing the industry best practices during every part of the development cycle. One important aspect of the standard is called as the Verification and Validation (V&V). Verification and Validation are performed on different stages of the hardware and software development cycle. Verification is the process of checking if the product is developed according to the defined specifications. In other words, it is the answer to the question, are we developing the product right? Validation is the process of checking if the defined specifications are sufficient to meet user's

needs or function satisfactorily . In other words, it is the answer to the question, are we building the right product?. Common hazards or risks encountered in the automotive development process and the best practices to avoid or mitigate them, are well documented in the form of safety standards. With the advent of the different approach to the vehicle software development, the standard verification and validation processes have become unsuitable to verify the AI modules present in autonomous vehicle.

Several works are going ongoing to standardize new technologies used in autonomous vehicles like the SOTIF standard [42], but they do not present techniques to implement verification and validation of AI modules. Eventhough research activities on verification are improving day by day, they are not enough to be implemented in the development process of autonomous vehicles. Hence, the AV industry has adopted simulation-based testing as a means to verify the systems. Simulation based techniques are an essential part of Testing, Verification and Validation (TV&V) of AV development, but they in themselves are not sufficient to ensure safety in all situations. Researchers agree that formal methods have the potential to develop trustworthy AI systems.

Three main problems have to be identified and must be solved for AI modules in AVs to be successfully verified. Research in the topic of AI verification and validation are growing steadily over the years and it has outputted several promising techniques for the verification of deep neural networks. Perception is usually the major bottleneck in AVs, as they influence the performance of remaining modules. Convolutional Neural Networks (CNN) are often employed in the perception modules of an AV for object detection. Most of the current researches are concentrated on verifying the robustness property of the CNNs against adversarial inputs.

Several properties of the neural network needs to be verified before the AI is considered trustworthy. Functional verification is a type of verification that concentrates on verifying the functionality of the neural network. For example, it investigates if the CNN performing object detection performs it's task in accordance with the specifications. It tries to verify if the CNN classifies the object correctly and it does so for the right reasons. This helps us to understand that whether the CNN model has learned the right features in an object and uses it to classify the object. Current testing of CNN only concentrate on the classification result and does not care about checking the features it has learned.

The second problem in verifying neural networks is that, the available techniques are unable to be scale to actual neural network because of their huge size and the number of parameters in them. Another one of the problems encountered in performing formal verification of AI modules other than the problems due to the inherent nature of AI is the lack of methods to formally specify the desired properties of the neural network.

The third and final problem is the lack of tools to facilitate the entire automation process from the generation of formal specifications to performing formal verification. Current testing strategies like simulation based testing have many tools, while formal methods do not have tools that are able to perform all the tasks in the verification workflow like, formal specification generation and formal verification.

This thesis aims to address these three problems. It will investigate formal specification languages suitable for verification of AI-based perception modules, techniques to formally verify CNNs that are scalable. The findings of these research objectives will be used to create an automation tool that generates formal specifications that is capable of capturing the desired properties of the object detection module and perform a formal verification of them. It uses

an ontology to model the driving environment of the AV and uses it to generate formal specifications that are semantically meaningful. Ontologies provides the much needed context that is often missing in traditional TV&V processes. The tool uses a novel method that leverages Vision Language Models (VLM) to verify the classification of objects and the detection of features or concepts present in an object. This method is highly scalable and can be used to verify CNNs of any size.

The thesis is structured as follows. Chapter 1 presents a detailed background on autonomous vehicles, their components. It also diffrentiates the notion of safety and security pertaining to AVs, the homologation process (certification process) of AVs and automotive safety standards. It then explains recent developments in AI and their use in the AV followed by an introduction to Verification and Validation in AVs. Chapter 2 presents a detailed literature review on the state-of-the-art research on topics like object detection, formal specifications, formal verification of AI and state-of-the-practice in AV TV&V.

Chapter 3 presents the research design, objectives of the study, research questions to be explored, hypothesis and the scope of the thesis. It concludes by presenting the tool design and development methodology, requirement analysis, the generic tool architecture, the process and data flow of the tool and testing and validation methods for the tool.
4 presents the design principles used in the development of the tool and explores our choice of formal specification language and formal verification method used in the tool. It then describes the tool architecture in detail with its modules. The chapter concludes by describing the implementation of the tool and the processes involved in the verification of the zero-shot CLIP model and the metrics for assessing the results.

Chapter 5 presents the results of the zero-shot classification of the CLIP model on *RIch Visual Attributes with Localization(RIVAL10)* and the *German Traffic Sign Recognition Benchmark(GSTRB)* dataset. It then presents the verification results and discusses them. The chapter concludes by presenting some of the limitations and the directions for future research.

Chapter 1

Background

This chapter provides the necessary background information for understanding the landscape of verfication and validation of AI modules in autonomous vehicles. This chapter starts by providing a brief introduction on autonomous vehicles, tasks performed by them. levels of automation in autonomous vehicles and the components that constitute an autonomous vehicle. It then presents the notion of safety, security and their differences in the context of autonomous vehicles. It also discusses the homologation process of the autonomous vehicles briefly. The chapter then concludes by discussing the AI methods used in autonomous vehicles and the process of verification and vaidation techniques.

1.1 Autonomous Vehicles

Autonomous vehicles, also known as self-driving cars, are intelligent machines capable of navigating and controlling themselves without human intervention. The recent advancement of technologies such as AI, machine learning, and IoT has enabled to perform driving tasks in real-time scenarios [40]. The concept of road vehicle automation involves replacing human labor in driving, with electronic and mechanical devices. The evolution of automation driving over time has encompassed features like lane control, speed management, and more. These vehicles have the potential to significantly impact urban mobility, reduce traffic congestion, lower greenhouse gas emissions, and enhance overall transportation efficiency [16]. The Society of Automotive Engineers (SAE) defines six levels of driving automation 1.1, ranging from Level 0 (no automation) to Level 5 (full automation), where the vehicle can handle all driving tasks independently [64]. One of the main aim of autonomous driving system is to reduce the number of accidents caused by human error. This is achieved by integrating advanced sensors and AI for more precise and timely decision making compared to human drivers [65]. Since they are equipped with advanced safety features and continuously monitor their surroundings to avoid potential hazards, road accidents can be significantly reduced [64].

To understand the capabilities of vehicle automation, the Society of Automated Engineers (SAE), have provided a structured framework for understanding their progression and capabilities. To better understand the SAE standards explained in the table reftab:saeLevels, we need to address the definition of some important terms presented by the SAE 2021 standard:

Figure 1.1: Autonomous vehicle(Waymo) [61]

- **Dynamic Driving Task(DDT)**: all of the real-time operational and tactical functions required to operate a vehicle in on-road traffic, excluding the strategic (high-level) functions such as trip scheduling and selection of destinations and waypoints. DDT can be divided in at least two main subtasks:

  - **Object and Event Detection and Response (OEDR)**: this task involves the monitoring of the driving environment and executing an appropriate response to events happening in an environment, such as vehicles changing lanes, traffic light status, stop signs, children zones, etc.
  - **Object and Event Detection and Response (OEDR)**: the control task of the vehicle involves the manipulation of the vehicle motion to preform regulated motion along the x- and y-axes in the real-time situation.

- **Operational Design Domain (ODD)**: operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics.

- **Driving Automation System**: the hardware and software that are capable of performing a part of or all of the DDT on a sustained basis; this term is used generically to describe any system capable of Level 1-5 driving automation (from partial to full automation).

- **Automatic Driving System (ADS)**: the hardware and software that are collectively capable of performing the entire DDT on a sustained basis, regardless of whether it is limited to a specific operational design domain (ODD); this term is used specifically to describe a Level 3, 4, or 5 driving automation system.

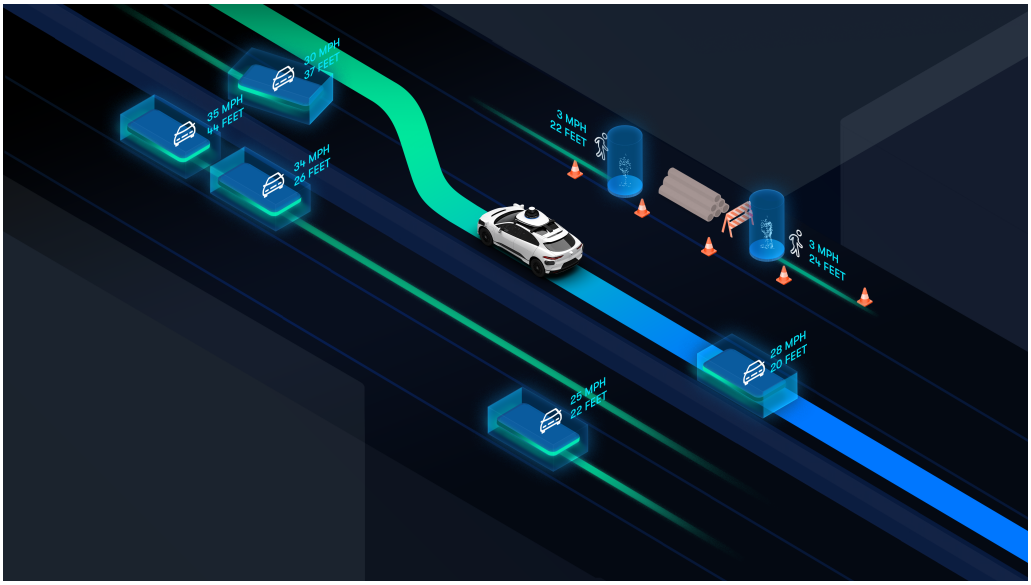

Figure 1.2: Autonomous Driving[61]

- **DDT Fallback**: the response by the user to either perform the DDT or achieve a minimal risk condition after occurrence of a DDT performance-relevant system failure(s) or upon operational design domain (ODD) exit, or the response by an ADS to achieve minimal risk condition, given the same circumstances.
- **Minimal Risk Condition**: a condition to which a user or an ADS may bring a vehicle after performing the DDT fallback in order to reduce the risk of a crash when a given trip cannot or should not be completed. Note that this definition does not establish what the "acceptable" reduced risk of a crash is. This is an open and important question as the success of the DDT fallback performed by the ADS will define its consideration as Level 4 or 3.
- **Request to Intervene**: a notification by an ADS to a user indicating that he/she should promptly perform the DDT fallback, which may entail resuming manual operation of the vehicle (i.e., becoming a driver again), or achieving a minimal risk condition if the vehicle is not drivable. This concept is also referred to as Take-Over Request (TOR)[17].

With the above definitions, we can now describe the six levels of vehicle automation of the SAE standard from SAE International, 2021
[54].

1.1.1 Components of Autonomous vehicles

Autonomous vehicle have several key components that help them to perceive the environment and perform driving tasks. They are integrated as various modules in the vehicle such as,

- **Perception Module**: This module takes input from various sensors to detect surrounding objects and maps a concrete feature space (e.g., pixels) to an output such as a classification, prediction, or state estimate. It is essential for understanding the environment and

Level	SAE Definition	Driving Automation	Human Driver Role	Fallback
0	No Driving Automation	No automation. Driver performs all driving tasks.	The driver performs the entire DDT, even when enhanced by active safety systems.	Driver
1	Driver Assistance	The sustained and (limited) ODD-specific execution by a driving automation system.	The driver performs the remainder of the DDT .	Driver
2	Partial Driving Automation	The system executes the part of the OEDR subtask that allows the execution of vehicle motion control	the driver completes the OEDR subtask and supervises the driving automation system.	Driver
3	Conditional Driving Automation	ADS performs sustained and (limited) ODD and entire DDT	User is receptive to ADS requests to intervene.	Driver
4	High Driving Automation	ADS performs sustained and (limited) ODD and entire DDT	Driver may not need to continuously monitor the environment, but must be prepared to resume control if the system malfunctions or requests it.	System
5	Full Driving Automation	ADS performs sustained and unconditional ODD and entire DDT	No human driver input is required. (Not yet achieved in production vehicles).	System

Table 1.1: SAE Levels [54]

making informed decisions. This module includes various components like sensors (cameras, RADAR, LiDAR) and algorithms that process sensor data to detect, classify, and track objects in the environment [12]. This can be normally found mounted on the top of an autonomous car 1.1, as they have a large module with multiple sensors. Some of these sensors are fused to increase the accuracy of the data.

- *Cameras*: They are used to provide a 360-degree view of the surroundings, essential for object detection and recognition. However, they can be inaccurate in low-light conditions and generate large amounts of data.
- *Infrared Cameras*: These cameras have a better performance in low visibility conditions. This compensates for the low performance of the cameras during low-light conditions.
- *LiDAR*: Light Detection and Ranging sensor uses laser beams to measure distances by calculating the time it takes for the light to reflect back to the receiver. This helps in creating a precise 3D map of the environment around the vehicle. The data can be annotated and segmented to identify different objects and their positions in the environment. This segmentation helps in understanding the scene and making

decisions based on the types of objects detected [22]. The data can be annotated and segmented to identify different objects and their positions in the environment. This segmentation helps in understanding the scene and making decisions based on the types of objects detected [40].

- *RADAR*: This sensor uses radio waves to calculate the distance between the vehicle and surrounding objects. This is achieved by emitting radio waves and measuring the time it takes for the reflected waves to return to the receiver. This system can determine the velocity of objects by analyzing the Doppler shift in the frequency of the reflected radio waves. This helps in understanding the speed of nearby vehicles and other moving objects. One of the significant advantages of RADAR is its ability to operate effectively in various weather conditions, including rain, fog, and snow. This makes it a reliable sensor for autonomous driving in diverse environments. Also, they're capable of detecting objects at long distances, which is essential for high-speed driving and early detection of potential hazards. This long-range capability complements other sensors like LiDAR and cameras [40].
- *Ultrasonic*: For shorter distance detection, ultrasonic sensors are used, as the above sensors will have a blind spot when encountered with obstacles in close range. For more accurate and robust perception of the environment, these sensors are fused together to compensate the limitations of other sensors [30].
- *GNNS*: Provides navigation, positioning, and traffic information using satellite signals. These signals can be weak and susceptible to interference.
- *GPS*: Provides navigation, positioning, and traffic information using satellite signals. These signals can be weak and susceptible to interference [40].

This module heavily uses deep neural networks1.2.1 for tasks like object detection and classification to effectively perceive the environment.

- **Prediction Module**: The prediction module is essential for forecasting the future trajectories of surrounding objects, such as other vehicles, pedestrians, and cyclists. This foresight allows the autonomous vehicle to make informed decisions and plan safe maneuvers. The prediction module takes input from the perception module, which processes sensor data like road images, point clouds, and GPS signals to detect and classify surrounding objects. Prediction modules heavily utilize deep neural networks (DNNs) to analyze patterns and predict future movements of detected objects. These DNNs are trained on large datasets to improve their accuracy and reliability [30].The prediction module operates in real-time, continuously updating its forecasts based on the latest sensor data. This real-time capability is crucial for adapting to dynamic changes in the environment, such as sudden stops or lane changes by other vehicles [51]. The output of the prediction module is fed into the planning module, which uses the predicted trajectories to decide the optimal path for the autonomous vehicle.
- **Planning Module**: The planning module is crucial for determining the optimal path and actions for an autonomous vehicle to navigate from one point to another safely and efficiently. Various path planning algorithms are employed to calculate the optimal path. These include discrete optimization approaches, cubic spline interpolation techniques, and Frenet coordinate systems. These methods help in generating smooth and efficient paths for the vehicle. The module continuously updates its decisions based on the latest sensor data and prediction data. This ensures that the vehicle can adapt to dynamic changes such

as moving obstacles and varying traffic conditions [40]. During this phase the vehicle will have a general understanding of the environment as depicted in the image1.2, the vehicle uses this knowledge to make a decision.

- **Control Module**: The control module receives the planned route from the planning module, which has already processed the perception and prediction data to determine the optimal path for the vehicle. It receives the planned route from the planning module, which has already processed the perception and prediction data to determine the optimal path for the vehicle. Typically, traditional logic-based programs and control algorithms are employed. These may include Proportional-Integral-Derivative (PID) controllers, Model Predictive Control (MPC), and other advanced control techniques to ensure precise vehicle maneuvering [30].

1.1.2 Safety and Security in Autonomous vehicles

Safety in autonomous vehicles refers to the system's ability to operate without causing harm to passengers, pedestrians, and other road users. It involves ensuring that the vehicle can navigate and respond to its environment accurately and reliably. Here we focus on the vehicle's operational performance, including accurate perception, reliable prediction, and effective control. It involves rigorous testing and validation to ensure that the vehicle can handle various driving scenarios safely. Safety failures can lead to accidents, injuries, or fatalities. For example, if the perception system fails to detect a pedestrian, it could result in a collision [51] [10]. Safety standards for autonomous vehicles are defined by various regulatory bodies, such as ISO and SAE. These standards outline the requirements for safe operation, including perception accuracy, system reliability, and emergency response capabilities.

Security in autonomous vehicles pertains to protecting the system from malicious attacks, unauthorized access, and data breaches. It ensures that the vehicle's software and hardware are safeguarded against cyber threats that could compromise its functionality. Here the main focus is to protect vehicle's communication networks, data integrity, and software systems from external threats. This includes implementing encryption, authentication protocols, and intrusion detection systems to prevent cyber-attacks. Security measures involve implementing cybersecurity protocols, such as encryption, secure communication channels, and regular software updates. These measures protect the vehicle from hacking, data breaches, and other cyber threat. Failures can lead to unauthorized control of the vehicle, data theft, or system malfunctions. For instance, a hacker could take control of the vehicle's steering or braking systems, posing significant risks to passengers and other road users. Security standards are also defined by regulatory bodies and focus on protecting the vehicle's systems from cyber threats. These standards include guidelines for secure software development, data protection, and incident response protocols [51].

1.1.3 Homologation Process

The homologation process refers to the approval process that vehicles must undergo to ensure they meet safety and regulatory standards before they can be allowed on public roads. In traditional automotive homologation processes, the main aim is to guarantee the safety of vehicles on public roads and ensure they comply with established regulations [66]. This procedure typically involves rigorous testing and examination to ensure the product complies with safety, environmental, and performance regulations [17]. The World Forum for Harmonization of Vehicle Regulations (WP.29) is an international body, established in 1958, was formed to address

vehicle regulatory standards globally [66].

The current homologation process involves certifying products meet regulatory standards for legality. The regulatory compliance testing traditionally is done by testing the vehicles physically on tracks or test benched, to assess the safety levels. A detailed documentation of product specifications and test results are also examined for compliance. While using test tracks and test benches may be satisfactory for most of the vehicles, it raises a question when the process involves an autonomous vehicle. Since these vehicles depend upon various software elements, for their performance in real-world driving, it requires a further level of testing. The traditional method requires the vehicles to be tested in real-world conditions, this testing is mainly to evaluate the performance of the software systems in the vehicle. [66] [17].The UNECE uses an independent third-party to preform the homologation process (type approval), while the OEM provide a self-certification for their products. Type approval process, conducted by OEM independent organizations like TÜV or DEKRA, certifies components/products to meet regulations before entering the market [66].

Safety standards ensure that automated vehicles (AVs) are designed, manufactured, and operated with a high level of safety in mind. Safety is the most crucial factor influencing user acceptance of AVs, making it a top priority for developers and policymakers. The assessment of safety in AVs involves complex challenges such as real-world testing limitations and innovative methods requirement [17]. Safety standards, such as ISO 26262, play a crucial role in ensuring the reliability and security of electronic systems used in vehicles. ISO 26262 establishes guidelines for software development in automotive electrical/electronic systems, focusing on functional safety operations. These standards define requirements related to development, design, implementation, integration, verification, and validation processes in the automotive industry. Safety standards aim to reduce hazards associated with software in safety functions to an acceptable level by providing feasible requirements and processes. The main elements covered by safety standards include requirements definition, design aspects, implementation framework, integration with tools, verification and validation, and configuration processes [66].

1.2 AI and ML

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans. Machine Learning (ML), a subset of AI, involves the use of algorithms and statistical models to enable computers to improve their performance on a task through experience. These technologies are increasingly used in various sectors such as transportation, healthcare, finance, etc. One of the notable sector where significant research is being conducted using AI is autonomous vehicles. AI, particularly deep learning models like Convolutional Neural Networks (CNNs), is widely used in developing autonomous driving systems. These systems can perform tasks such as recognition, prediction, and planning in diverse driving conditions, making them crucial for the advancement of self-driving cars [12]. For instance, AI has gained significant attention in the development of autonomous driving systems for railways and other transport sectors [43].

1.2.1 Deep Learning

Deep learning is a subset of machine learning that involves neural networks with many layers, known as deep neural networks, which are designed to mimic the neurons in a human brain, and it's ability to learn from large amounts of data. These networks consist of interconnected nodes (neurons) which are organized in layers. The layers consist of an input layer, one or more hidden layers, and an output layer. Each neuron in one layer is fully connected to every neuron in the next layer [65]. Deep learning is extensively used in the development of self-driving cars. It enables vehicles to learn meaningful features that constitute an driving environment such as a traffic light, stop signs, humans, etc. The key advantage of deep learning is it's ability to automatically learn features from raw data, this avoids the need for hand designed feature extractors, which require considerable engineering skill and domain expertise. Sensors provide the raw input data to the model, which is processed to make driving decisions through an end-to-end learning approach. Deep learning models are trained using various learning rules, including supervised learning, unsupervised learning, and reinforcement learning. The choice of learning rule depends on the specific task and data available [27]. Convolutional Neural Networks (CNNs) are particularly effective for this application, allowing vehicles to execute recognition, prediction, and planning tasks in diverse conditions [65]. The field of deep learning is rapidly evolving, with new learning algorithms and architectures being developed. These advancements are expected to accelerate progress and expand the range of applications for deep learning [27].

Neural Networks

Neural Networks are the fundamental building blocks. They are computer systems loosely inspired by the structure and function of the brain. They consist of interconnected nodes (artificial neurons) that process information and learn from data [24]. Each neuron computes a weighted sum of its inputs and applies an activation function, such as a sigmoid or ReLU, to produce an output. The connections between neurons have associated weights, which are adjusted during training. Biases are additional parameters that help the model make more accurate predictions [27]. There are different type of neural networks,

- **Feedforward Neural Networks(FNNs)**: These networks have a straightforward structure where the data flows in one direction from input to output. Multi-Layer Perceptrons (MLPs) are a common type of FNN [24].
- **Convolutional Neural Networks (CNNs)**: Designed for processing data with a grid-like topology, such as images. CNNs use convolutional layers to detect features and pooling layers to reduce dimensionality.
- **Recurrent Neural Networks (RNNs)**: Suitable for sequential data, RNNs maintain a state vector that contains information about previous inputs, making them effective for tasks like language modeling and time-series prediction [3].

Training neural networks can be computationally intensive and requires significant resources. Additionally, designing the optimal network topology and ensuring robustness against adversarial attacks are ongoing challenges [20].

1.2.2 Transformers

Transformers are a type of neural network architecture designed to handle sequential data, such as text, by using self-attention mechanisms to weigh the importance of different parts of the input sequence [27]. The Transformer model was introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017 [57], revolutionizing natural language processing tasks by enabling parallel processing of sequence data.The key components of transformers are,

- **Self-Attention Mechanism**: This mechanism allows the model to focus on different parts of the input sequence and weigh the words in a sentence when producing an output, making it highly effective for tasks like translation and text generation as the model will have a better understanding of the text.
- **Positional Encoding**: Since transformers do not process data sequentially, they use positional encodings to maintain the order of the input sequence, ensuring that the model understands the relative positions of tokens [27].

The original Transformer model consists of an encoder to process the input sequence and a decoder to generate the output sequence. Both the encoder and decoder are composed of multiple layers of self-attention and feedforward neural networks. The architecture involves a *Multi-Head attention* component allows the model to jointly attend to information from different representation subspaces at different positions, enhancing its ability to capture various aspects of the input data [27].

There several advantages which makes them a great choice for NLP, as Unlike RNNs, transformers can process all tokens in the input sequence simultaneously, significantly speeding up training and inference times, and they can be scaled up to handle very large datasets and complex tasks, thus making them suitable for state-of-the-art models like BERT and GPT [27].

1.2.3 Influence of AI in Autonomous vehicles

AI has been integral to the development of autonomous vehicles, starting from early automated range finders to today's self-driving cars. Neural Network (NN) algorithms, particularly Deep Neural Networks (DNNs), have been widely applied in both civilian and military domains, enabling vehicles to plan and execute complex operations with minimal human interaction. The trend in the industry is to use DNNs to implement vehicle control algorithms. This deep-learning-based approach allows vehicles to learn meaningful road features from raw input data and output driving actions as discussed in 1.2.1 , this facilitates end-to-end learning for complex real-world driving tasks [65].

By leveraging AI, autonomous driving systems have seen significant improvements. AI and Machine Learning (ML) algorithms have been crucial in developing ADAS, which must comply with numerous rules and laws. The complexity of these systems has been managed by leveraging data-driven AI models, which process vast amounts of data from sensors like cameras, radar, and LIDAR to ensure functional and safe systems [19]. Autonomous driving requires the vehicle to have a behaviour reflex which maps the sensory input directly to driving actions and mediated perception which involves using numerous sub-components to identify driving-related objects such as traffic lights, lanes, signs, and pedestrians, creating a comprehensive representation of

the vehicle's surroundings [64].

The recent data-driven improvements, such as the availability of datasets and increased computational power also played a role in their improvements. Since the data-driven models learn from a vast amount of traffic data, it directly constitutes to the increase in their accuracy and reliability [19]. To address the limitations of purely data-driven models, hybrid-AI approaches combine data-driven AI with knowledge-based systems. This enhances situational awareness, allowing the system to reason about its competence in real-time traffic situations and decide when to hand over control to the driver or safety system [39] [64].

To factor in the safety and compliance, models are being developed to adhere to traffic laws and road safety standards. This can be accomplished by formalizing the traffic regulations into ontologies, which helps the AI system to understand the rules and safety standards, thus enhancing the reliability and safety of autonomous driving [39]. The influence of AI in autonomous driving systems has enhanced major components of it, allowing further development in the field.

1.3 Verification and Validation

Verification and validation (V&V) are crucial processes in ensuring the effectiveness and safety of automated vehicles (AVs) [32]. The primary goal of verification and validation is to guarantee the reliability, safety, and trustworthiness of AI systems in their operation and outcomes, supporting their responsible deployment for societal well-being [17].

1.3.1 Verification

Verification is the process of evaluating a system or component to determine whether it meets specified requirements. It ensures that the product is built correctly according to design specifications and standards. Verification is crucial in system development, as it helps identify defects early in the development cycle, reducing the cost and time associated with fixing issues later. It ensures that the system performs as intended under specified conditions. Verification in AI involves confirming that the AI system operates as intended and meets specific requirements. Verification aims to ensure the reliability, safety, and effectiveness of AI systems by systematically assessing their performance against predetermined criteria [17]. Some of the verification methods are,

- **Formal Verification**: This method requires formulating requirements as range constraints on neuron outputs, which is challenging due to the black-box nature of DNNs and the enormous size of the state-space, often necessitating tools like solvers (e.g., Reluplex) that are limited by architecture types and speed [44].
- **Model Checking**: Involves using a specification language and analysis to verify system correctness relative to requirements , also incorporating techniques like linear temporal logic specifications for detailed property representation [32].
- **Scenario-based Testing**: Categorizes into fundamental and advanced approaches based on the interaction between road users in generated scenarios [32].

- **Fault Injection Testing**: A method for intentionally introducing faults to evaluate system robustness and its ability to handle errors.

1.3.2 Validation

Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. It ensures that the right product is built to meet the user's needs and expectations. It is essential for ensuring that the final product meets the user's needs and expectations, providing assurance that the safety requirements are adequate in terms of correctness and completeness. For AI systems, we assess if it meet's the user's needs and expectations, ensuring its effectiveness and safety. Some of the validation methods involve,

- **Field Testing**: Field testing is vital for validating AI systems in real-world environments, ensuring their performance matches expectations
- **User Acceptance Testing(UAT)**: UAT involves end-users testing the system to validate its usability, functionality, and whether it meets their requirements.
- **Cross-Validation**: The cross-validation technique helps in validating the AI model's performance by partitioning the data into subsets for training and testing.
- **A/B Testing**: A/B testing is a validation method where two versions of the AI system are compared to determine which performs better based on specified metrics [17].

Chapter 2

Literature Review

This chapter presents a literature review on the topics of object detection, formal specifications, formal verifications and on the state-of-the-pratice testing, verfication and validation methods in the autonomous vehicles developement process. The sections on verficaition and validation focus on presenting influential works in the field of software verification and vailidation. It also discuss the verification and validation techniques available for neural networks and other AI methods. Some of the related works regarding the verificaiton and validation of Convolutional Neural Networks(CNNs), used in the control and perception modules of the autonomous vechiles are also presented.

2.1 Object detection

Current state-of-the-art object detectors are highly advanced, involving many different algorithms, methodologies and architectures. The following sections discuss some of the advanced techniques in the field of object detection.

Single-stage detectors are one of the classes of deep learning models, which are known for their fast and efficient object detection. The name single-stage comes from their nature to perform both object localization and classification in a single forward pass of the network. This makes the single-stage detectors quicker and more efficient computationally compared to two-stage detectors. It is designed to be fast and efficient, making it suitable for real-time applications such as autonomous vehicles [27]. Some of the popular single-stage detectors are YOLO (You Only Look Once) [23] and SSD (Single Shot Detector) [45]. YOLO is a family of single/stage detectors that has been continuously improved over the past years. The YOLOv10 is one of the recent addition to the YOLO family of detectors, this is considered to be one of the most advanced detectors in the field and has shown remarkable results in various applications [59]. Recent advancements in single-stage detectors have addressed challenges like domain adaptation and incremental learning. Attention-based domain adaptation techniques are said to have improved the single-shot detector performance when applied to a new domain [58]. Techniques like mining memory neurons are utilized to allow single-stage detectors to learn new object classes without forgetting previously learned classes [29]. This has enabled SSD to be widely adapted in autonomous driving systems for tasks such as vehicle detection, pedestrian detection, and traffic sign recognition. Its real-time processing capability is crucial for the dynamic environments encountered by autonomous vehicles [27]

Two-stage detectors are another popular class of deep-learning models, that are used for object detection. They distinct themselves by performing object detection in two stages. In the first, stage the model generates a set of potential object regions or the so-called proposals within an image. In the second stage, the proposed regions are refined along with their bounding box coordinates. These models have a network or algorithm that is dedicated towards generating region proposals. Because of this two-stage approach, these models have higher accuracy than the single-stage detectors even for small objects or complex scenes. Some of the popular two-stage detectors are R-CNN (Region-based Convolutional Neural Network), Fast R-CNN, Faster R-CNN, Mask R-CNN, Cascade R-CNN etc [13]. R-CNN is the original two-stage detector that used selective search for region proposals followed by a CNN feature extraction and SVM classification [7]. Other models were developed by improving or modifying R-CNN. While, the two-stage detectors have higher accuracy, they are often slower than single-stage detectors. This makes them less suitable for applications, such as autonomous driving where speed of detection is crucial.

Transformers are a class of deep-learning models that found its application in object detection because of their success in natural language processing applications. The applications of a transformer in object detection include image feature extraction, transformer encoder and decoder. Some of the characteristics of the transformers like their ability to capture the global context by understanding the relationship between objects have made them a desirable choice in the field of object detection. One of the most popular examples of a transformer in object detection is the DETR (DEtection TRansformer). It was introduced by Meta AI and is said to have comparative performance with Faster R-CNN [47]. It uses a standard transformer architecture to perform an end-to-end object detection [9]. However, transformers need to overcome some of their shortcomings like computational complexity and their need for large amount of data for training before they can be deployed for safety-critical applications.

2.2 Formal Specification

The importance of verifying a system is well understood. However, a verification task is only meaningful if meaningful and high-quality formal specifications. Formal specifications are used to clearly define the properties that are expected to be exhibited by a system. Most of the literature discuss the problems in formulating formal specifications for AI-based systems. AI-based systems learn from data rather than following pre-defined rules which makes it difficult to specify its desired properties. Some of the notable works on formal specifications are presented below.

Verification of deep neural networks also needs formal specifications to be provided. They can be classified into many types based on the property of the neural network that needs to be verified. They are system-level specifications, specifications on input-output robustness, specifications on input-output relations, specifications on semantic properties, specifications enforcing monotonicity and fairness, coverage and temporal specifications [46]. A framework is presented for generating synthetic training data for DNNs and defines formal specifications to ensure high quality data generation. This work is aimed to address the challenges in obtaining well-labeled training data to improve the efficiency of DNN training. The idea is to precisely define the characteristics of the objects that needs to be present in the data that is being generated. It uses a light version of the Structured Object-Oriental Formal Specifications (SOFL) to specify

the distinct features of traffic signs like its shape, color etc [63]. A case study is conducted to validate the framework. Although the application of the formal specifications is different, it can also be extended for the purposes of verification.

Another framework called CAMUS [18] is proposed to formally express safety properties related to perception of a neural network using simulators. The idea is to take advantage of the simulators which are often used to train machine learning models using statistical tests to formally express and verify safety properties. The properties are defined such that it covers all cases that could be generated by the simulator as opposed to only using simulators for statistical tests. The framework utilizes a translational tool called ONNX2SMT, which converts the neural networks to SMT based logical formulas which can be then used to verify different properties. Another work [53] introduces a new language called as Bounding Box Specification Language (BBSL), which is designed to precisely describe the objects or entities found in a driving environment as bounding boxes in a 2D space. BBSL uses mathematical notations to describe the positional relationship between two objects in an environment, thus making it suitable for OEDR task specification.

2.3 Software Verification And Validation

Verification and validation of software in safety-critical systems like autonomous vehicles are crucial to ensure safe operating of the system. Verification is the process of confirming that our system is built according to the defined specifications. Verification involves activities like static analysis, code reviews and inspections. Validation is the process of confirming if the defined specifications are enough to meet the user needs when used. There are several different methods available for verification and validation of the software. Some of the validation activities are user testing, dynamic testing etc. Some of the conventional verification and validation methods are automated testing, runtime monitoring, formal methods, cross-validation and visualization.

2.3.1 Formal Verification

Formal verification is the most preferred method among the existing verification techniques. They use rigorous mathematical techniques to prove the correctness of the algorithms or systems. This section presents some of the types of formal methods found in the literature. They are abstract interpretation, semantic static analysis, model checking, proof assistants, deductive verification, model-based testing [26]. Abstract interpretation is the process of simplifying the verification process by focusing on the high level abstractions of the system. Semantic static analysis analyses the source code without executing them. Model checking uses graphical models of the system to verify the specifications. Techniques involving Satisfiability Modulo Theories (SMT) solvers and Mixed Integer Linear Programming (MILP) are also often used to verify the robustness property of the neural networks.

Leucker[28] discusses the challenges encountered in verifying the neural networks in detail. Lack of formal specifications in machine learning applications and the need for formal specification techniques are highlighted in this work. The notion of trustworthy AI is discussed in the paper, also a novel approach that uses a surrogate model to verify the properties of a recurrent neural network (RNN) is presented. It provides insights into identifying violations and provides an error probability for the said violations. However, there are no methods presented to extend

this method for the verification of CNNs.

Verification of AI

It is absolutely necessary to verify the AI used in safety-critical applications like autonomous driving to ensure its reliability, safety and improve its trustworthiness. Ai systems are complex and are of black-box nature. Existing methods for the verification and validation of software systems are not readily translatable to AI systems. AI systems are also called as Learning Enabled Components (LEC) and they are a different paradigm of softwares which learn by examples rather than by providing explicit instructions. Some of the formal verification techniques available for the verification of AI found in the literature are presented below,

[26] explore the usage of formal methods to verify machine learning systems. The study points out that not enough attention is given to the initial stages of the model development like data collection and model training. It provides a useful summary of existing formal verification methods like abstract interpretation, SMT solvers and MILP techniques. Some of the formal verification methods and tools that have been used for verification of convolutional neural networks are presented below.

CNN-Cert, DeepZ, ReluVal are some of the tools that are capable of verifying the robustness properties of the CNN. CNN-Cert is designed to test the robustness of CNNs against adversarial attacks thereby ensuring that small changes in the input of the network does not lead to incorrect classifications [8]. DeepZ is a tool that uses abstract transformers to verify the robustness property, to ensure that the CNN has constant performance with adversarial perturbations [48]. ReluVal is a tool designed to iteratively refine the input scope of the CNN, thus ensuring that the CNN behaves correctly for a wide range of inputs [60].

This paper [25] introduces the concept of local transformational robustness, which measures how well a CNN model is able to handle the changes in an image such as shift in position or color, without misclassifying them. The local transformations are encoded as a MILP problem which allows for the formal verification of the CNN under transformations. The idea is tested by a model trained on the MNIST dataset with varying transformations.

2.4 Testing, Verification and Validation

The state of the practice in testing autonomous driving systems is investigated by [31] by conducting interview and survey with 110 developers from 10 different companies and it had identified seven common practices. It found that problems like identifying corner cases and unexpected driving scenarios is well researched. It points out how recent researchers have concentrated on testing strategies for end-to-end models while the industry is using modular systems. It stresses the importance and need for techniques that will reduce the need for testing like verification.

Chapter 3

Methodology

In this chapter, we discuss the research design, the hypotheses and our automated tool development process. The section on research design focuses on explaining the objective of our study and the rationale behind it. The following sections discuss the overall methodological approach to the tool development process

In the section explaining the tool development process, we start by explaining how information like driving environment, objects in the environment etc are represented using ontologies. Then the process of generating formal specifications from ontologies are discussed. We then explore how the specifications are used to formally verify our model, that is going to be deployed on the autonomous vehicle. Finally, we finish our discussion by describing the software and hardware requirements for the process, the dataset used in the process.

3.1 Research Design

This section presents the research design by analysing the objective of the study, established research questions, the hypotheses behind the solution implementation and the overall rationale behind the design and the methodological approach taken to implement them.

It is evident from 2, that current Testing, Verification and Validation (TV&v) methods are simply unable to provide us with enough safety guarantees to allow autonomous vehicles to operate freely on roads. Even though the robotaxi companies are now allowed to operate in multiple cities in U.S on a trial basis, the companies have to spend a lot of resources in performing simulation-based and real-world testing before they are allowed to do so. This has led to many backlashes both at the companies and the city administration for allowing it, when the vehicle ends up in an accident[61].

The problem of autonomous driving is already 90% solved. Present day robotaxis can complete their rides without any safety incident most of the times. But the problem comes when the robotaxis encounter a scenario they have not seen or in an unknown environment. Autonomous vehicles can't be allowed to fail even if their failure percentage is less, as one of the main points of using them is to save lives from road accidents. But can't we wait until these vehicles are completely safe? One of the answers provided for this question from the industry is that, even at this stage where the autonomous vehicles are not completely safe, they are performing way

better than a average human driver. Road [61]accidents have been noted to be considerably less when autonomous vehicles are involved/.

Hence, it is reasonable to assume that, simulation based testing is only a temporary solution taken up by the companies to achieve their goals while more sophisticated methods are developed to guarantee the safety of the vehicles. Considering available techniques today, the method that might most likely lead us to guaranteed safety is when the TV&V is combined with formal methods. Formal verification methods can provide robust guarantees for the safety of the vehicles or AI based methods like neural networks. These methods might not completely replace simulation and real-world testing but will reduce the amount of time, effort and resource spent on making the autonomous vehicles safer.

The objectives of this study are defined based on the above said rationale and also on the problems commonly found in verfication and validation . Accordingly, the main objective of this work is to explore techniques and methods to automate the process of formal verification, focusing on object detection modules using neural networks. In order to achieve this objective, we must answer the following research questions.

1. **RQ1**: What could be some of the effective verification methods that are available for formally verifying the functional properties of a neural network based object detection module?
2. **RQ2**: How can we design a tool that captures the desired behaviour of the object detection module accurately and integrates seamlessly with already existing tools in the industry?
3. **RQ3**: How can the results of the verification be used to further better the object detection model?
4. **RQ4**: How can the developed tool aid in the homologation process of the autonomous vehicle for guaranteeing of safety?

The importance of these research questions are explained below. Please note that the notion of perception in the following text is referring to a neural network based object detection module. Perception modules may also have additional functions like object tracking, localization etc but in the context of this document, it refers to object detection modules.

There exists a lot of formal verification techniques as seen in 2, that focus on formally verifying neural network based controllers. Verification of controllers is an important process in evaluating the readiness of the entire system but there is comparatively very less verification techniques that focus on verifying perception based systems. Hence our RQ1, focuses on exploring verification techniques for perception components particularly for functional verification. Similarly, most of the available works that do focus on the verification of perception modules attempt to verify their robustness property. In contrast, we try to do a formal verification of their functional properties, as this area is comparatively less explored.

When designing an automation tool, it is imperative that it uses a specification language that is able to accurately capture the desired behaviour of our object detection module. The goal of any object detection module is to accurately identify objects present in an image and

classify them correctly. This functionality of the module will not be changing irrespective of the technology that is used to achieve it. RQ2 focuses on exploring specification languages that are capable of expressing this desired functionality formally.

When developing a tool to automatically generate specifications, it is important to make sure that our tool is compatible with already existing tools, that are alredy used in the industry. RQ2 also focuses on exploring methods or technologies that makes our tool compatible with existing ones.

It is a great challenge for ML engineers and data scientists to develop a neural network that is capable of performing well for most of the classes in different environment conditions. The neural network development process is an iterative process and it is very unlikely that any model behaves completely in accordance with our specifications. Thus the goal of our neural network development process should be to iteratively improve the model until it achieves the expected performance. To that end, RQ3 focuses on techniques that makes use of the results of the verification process to further improve the performance of the model.

At the end of the autonomous vehicle development process, it is heavily tested in both simulation and real world under different scenarios to make sure that the vehicle is capable of handling any unforeseen situations. This process is usually done by testing or targeted validation. After these testing process, the vehicle must be approved by regulatory bodies before it can officially hit the roads. Thus our tool must also focus on aiding this homologation process by generating useful artifacts like verification reports in an effective manner. These artifacts can be later used to support the claim of the vehicle being roadworthy. RQ4 concentrates on identifying the things that will make our tool helpful in certifying the object detection module.

3.1.1 Hypothesis

After exploring the research landscape and analysing gaps from existing works, the following hypothesis is derived. Technologies like ontologies, Domain specific languages (DSL) have the potential to be used to automatically generate formal specifications. The derived specifications can be encoded based on available integer programming techniques and solved with some off-the-shelf solver, thus completing the automation of formal verification.

3.1.2 Why use ontologies?

Ontologies are formal, structured representations of knowledge within a specific domain that define key concepts, their properties, and the relationships between them [62]. They are a semantic framework for organizing and representing information in a machine readable format. They also help in establishing a shared vocabulary within a domain, which may be useful in setting the context for the different activities within a domain. Hence the idea is to create a vision ontology to represent diverse environmental scenarios and object features, facilitating the generation of thorough and relevant specifications.

Some of the advantages of using ontologies as a component in our automation framework are as follows. Ontologies provide a structured way to represent domain knowledge, making it easier for automated systems to understand and process information [49]. Ontologies helps maintain

consistency in access to knowledge for different parts of the system, facilitating communication and data exchange. Most important reason to use ontologies is their formal structure and its automated reasoning and inference capabilities. These capabilities help us to make sure that the generated specifications are semantically meaningful.

Ontologies are already being used in the Autonomous Driving (AD) testing and Advanced Driver Assistance Systems (ADAS) testing. They are used in creation of different scenarios in comprehensive scenario based testing and for deriving test cases according to the V-model [15]. By using ontologies, our tool is able to easily integrate with other similar tools that uses ontology.

3.1.3 What is the need for automation?

Automating the process of Testing, Verification & Validation is a common part of a workflow in the software development process. Once the requirements of the software to be developed is set, automating the process of TV&V saves a lot of time, resource and effort. Principles like Continuous Integration and Continuous Deployment/Delivery (CI/CD) heavily relies on the automated testing and verification process, to ensure no downtime in the traditional software development and deployment.

When AI is part of the software system, it adds complexity to the already existing principles of CI/CD. With the involvement of AI models, we now have to make sure that newly obtained data is used to update and retrain the models frequently. This can be done by establishing Continuous Training (CT) pipelines to make sure the best possible model is ready to be deployed. When the software system is to be used as a part of safety-critical system like autonomous vehicles, we need to make sure that the model developed is not violating any safety specification. This leads us to the notion of Continuous Verification (CV). On top of that, before the software takes its place in the vehicle we have to make sure that it is built to satisfy industry specific safety standards, leading us to the notion of Continuous compliance (CC).

Making sure everything goes well at each and every stage of the process can be very labour intensive and inefficient. Automating this process of training, integration, verification, compliance and deployment by developing rigorous tools for design and analysis [14] may have immense benefits like fast product iteration, accuracy and efficiency. These tools may even become compulsory when it is mandated by authorities.

3.1.4 Scope of the thesis

Testing, Verification and Validation (TV&V) is a very large and complex process. It is especially true in the context of safety-critical systems development like autonomous vehicles. Autonomous vehicles development process is made modular and individual teams are usually responsible for development of those sub-systems. They usually have their counterparts in testing, who test the developed sub-system. It usually requires multi-disciplinary people coming together to perform the TV&V process. It is not possible to develop one size fits all tool, that solves all the problems in the industry. Thus the goal of this research has a limited scope and focuses only on certain parts of the complete TV&V life cycle.

The main goal of the TV&V process is to identify hazards and effectively handle them. There

are many kinds of hazards like safety hazards, security hazards, operational hazards, environmental hazards, ethical hazards etc. But we limit our focus to safety hazards, functional safety hazards in particular. It is important to understand the difference between safety and security as they are easily confused with each other. Safety deals with the hazards that have roots within the system like malfunctioning, poor design etc. Security deals with hazards caused by external factors targeting to cause damage to our system. Hacking, car jacking are some of the common security concerns.

We focus only on the safety of software components, specifically AI and the hazards caused by them. There are already a plethora of research done on different AI methods. We specifically concentrate on neural networks known for their notoriously opaque nature. Among the components of the autonomous vehicle as presented in 1.1.1, we limit our scope to considering only perception module that uses neural networks like the object detection module. Other parts of the perception system like localization, object tracking etc are considered out of the scope.

Our focus is solely on the functional safety i.e hazards that are caused by the malfunctioning behaviour of the vehicle or its components [41]. As discussed in section [standards], even this definition falls short with the involvement of AI. New definitions like intended functionality have been introduced to refer to the hazards cause by AI systems because of their inherent nature and their interaction with the system. This research focuses on the TV&V based on the notion of intended functionality as defined in the SOTIF standard [42]. Other scopes pertaining to the SOTIF standard also applies to this research.

3.2 Tool Design & Development Methodology

This section details the tool development process. It starts by introducing the tool and its goal followed by the requirement analysis of the tool, its design and implementation and testing. To solve our problem in verification and validation, we propose a tool that is capable of taking care of the entire formal verification process from generating accurate specifications to formally verifying those specifications and generating results or reports. Most of the tools used in the industry today are based on simulation based testing. As the trend of formal verification for AI systems grows, there is going to be a high demand for tools like these to aid in their development process.

3.2.1 Requirement analysis

Before we start our tool development process, it is necessary for us to be clear on the requirements of the tool. The following requirements are set based on our defined problem, research questions and theoretical research.

1. **Modular architecture**: The architecture of our tool needs to be modular to be adaptive of the evolving needs of the industry.
2. **Workflow integration**: The components that make up the tool must be able to easily fit into the already existing workflow of the industry.
3. **Ontology Processing**: The tool must be capable of parsing ontologies. It should support commonly used ontology formats like OWL, RDF etc.

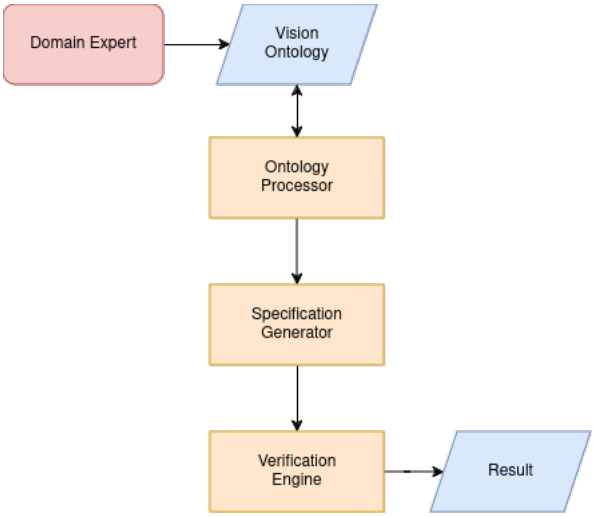

Figure 3.1: General Architecture

4. **Syntax Checking**: The tool should be capable of making syntax checks on the generated specifications to ensure correctness.

5. **Semantic checking**: The tool should be capable of making semantic checking to ensure that the specifications are meaningful.

6. **Traceability**: The tool should be able to maintain traceability between the specification and the verification.

We only concentrate on the functional requirements of the tool for now. Non-functional requirements like performance, scalability except for compatibility are assumed to be outside of the scope of this research.

3.2.2 Tool Architecture

Based on requirements for the tool, the architecture of the tool is designed to be modular. The modules that make up the architecture are an ontology processor, a specification generator, a verification engine. An illustration of the tool architecture in a generic sense is presented in figure 3.1.

Ontology Processor: This module will consist of an ontology processor that is capable of reading an ontology file in formats like OWL,RDF etc. It is responsible for extracting data from the vision ontology like the objects present in the driving environment, their features and preprocess it if needed. The ontology processor enhances the ontology data by inferring implicit relationships and properties based on the ontology's axioms by using reasoning engines like Pellet, Hermit etc. It should be able to retrieve specific data from the ontology using query languages. It should support some popular query languages like DL, SWRL, SQWRL, SPARQL etc.

Specification Generator: The specification generator should convert the data extracted from the ontology into formal specifications using a defined formal grammar of the specification

language and combinatorial algorithms. It should be able to parse formal grammar rule which are usually written in BNF (Backus-Naur Form) or EBNF (Extended Backus-Naur Form). It should use any templating methods to standardize the generation of formal specifications as reusable templates.

Verification Engine: The final module in our tool architecture will be a verification engine which encodes the generated formal specifications. The specifications are usually encoded as optimization problems or logical formulas and use some solver to solve them for verification. It should have an integrated solver like Z3, SCIP or some SMT solvers. It should have some functionality of logging and reporting to store or report the verification results.

UI: At least a basic UI is expected to be present in the tool, for communicating with the user. The configurations affecting the verification process are expected to be exposed to the user via the UI. It may also be used for accessing or viewing the verification results.

3.2.3 Process And Data Flow

The process of automatic verification starts when an ontology file is provided to the automation tool. The ontology is expected to be a vision ontology i.e an ontology containing objects present in a driving environment, their features, possible environmental conditions and other necessary information that helps us generate combinations of factors that we can use for specification generation. These ontologies are expected to be provided by domain experts who makes informed decisions from factual evidence or derive from their experience in the domain. The use of ontologies as an important piece in the autonomous vehicle development is being increasingly recognized and effort is being made to standardize the use of ontologies like the ASAM OpenX-Ontology [4].

It is assumed that correct configurations like file paths and parameters are set before beginning the verification process by using the UI of the tool. The process flow starts from the ontology processor, which extracts all the data necessary from the provided vision ontology file. The data is then used by the specification generator to create the specifications. At this stage we get an specification document outputted from the tool. The specifications are then parsed by the verification engine and encoded into a verification problem in proper formats. Data flow also begins in a similar fashion of that of the process flow.

3.2.4 Testing And Validation

The developed tool should be tested and validated, to make sure it meets the requirements as expected. Since the tool is going to be used for the purpose of automating the verification process of object detection modules, it is imperative that it is free of errors. Unlike the problem we are trying to solve, there is no AI present in the tool and it should be relatively simple to verify and validate that our tool behaves as intended. So the conventional methods used in conventional software engineering can be applied here.

It is recommended to set an objective and come up with a testing strategy for the tool. In this case, our objective is to ensure the functional correctness of the tool developed and also to test some of its features like assessing the correctness of the generated specifications. For this

purpose the usual testing strategy of unit testing, integration testing and system testing will suffice.

This chapter has discussed the research design in detail and presented the rationale behind the objectives, research questions and scope of this research. The hypothesis behind coming up with this automatic verification tool for our defined problem is discussed along with the rationale behind them. The later part of this chapter focuses on presenting the generic tool design and development process. This section was written with a focus to help anyone who wants to recreate this tool or expand it to other applications. Requirements of the tool and the system architecture along with their components are presented. The chapter also discusses about the process and data flow and the testing process of the tool at the end.

Chapter 4

Design And Implementation

In the previous chapter, we discussed the methodology of the tool development process. In this chapter we present detailed information on the design and implementation of the tool. We start by discussing the design principles that guide the development, the tool architecture and their sub-component design. We present the rationale for our choices in design and also present background information and analysis on different techniques or technologies considered. The potential benefits of our choices are also discussed. We finally discuss integration of those modules and how they are tested.

4.1 Design principles

Design principle of software development are a set of well defined and accepted ideas that ensure the quality of software systems being created. By adhering to well defined software principles during development, we can create quality software which will meet the defined requirements. The design of our tool is also guided by a few design principles of software development to ensure the effectiveness of the tool in automating the verification process. Some of the design principles taken into consideration are modularity, extensibility and usability. We will discuss about these design principles and how they contribute to the overall goal of the tool.

4.1.1 Modularity

A module is a software component that is created by dividing up software. A module may work independently but they must also fit seamlessly when integrated together with other modules. The process of creating modules is called as modularity in software engineering. The principle of modularity is that *Systems should be built from cohesive, loosely coupled components (modules)* [21]. Modularity may be decided upon by different factors like composability, decomposability, continuity etc.

Based on this principle, it is important for our tool architecture to be modular. This is taken as an important requirement for the tool. The landscape of tools and technologies available for data representation, parsers, verification methods and solvers is ever changing. New and improved tools are almost out more frequently than ever and it would be beneficial for us to design our tool to incorporate improved modules to better reach the goal of automatic verification.

According to the design principles of modularity, we built our tool architecture to be modular. As discussed in the 3, we have the following modules in our tool

1. Ontology Processor
2. Specification generator
3. Verification engine

We have divided our tool into these modules based on functionality. Each module has a specific job to do, which is then passed on to the next module upon completion. These modules are designed to be interacted in a linear fashion starting from the ontology processor up to the verification engine. So it is imperative that we make sure that there are no problems in the interaction, as they might make the tool unusable. We can do this by emphasizing the integration testing more.

Modularization of our code makes the development and debugging process simple. It also improves parallel development, code reusability and simplifies testing. More importantly, modularity benefits our tool by making our tool more adaptable. So, when we need to update any part of our tool, for example the ontology processor, it is possible for us to replace it with a better one. This is especially useful for the specification generator and verification engine module, as the goal and scope of the verification process changes. For example when the goal of the verification process changes from object detection to object tracking, we have to use the ontology data to generate specifications in a language that is capable of accurately describing the object tracking module behaviour. Not every verification problem can be encoded in the same way and thus we may want to use a different verification engine with a different solver that is capable of solving the verification problem.

Considering the benefits of the modularity in our goal to automate the verification process and for the tool to be relevant for different verification tasks, we have made modularity as a requirement for the tool as discussed in Section 3.2.1.

4.1.2 Extensibility

Extensibility is the ability to extend or adapt the software to meet the future demands. Extensibility is built upon the principle of modularity and thus a modular tool architecture is a pre-requisite for extensibility. In order for our tool to be extensible, we have to make sure that the modules or components of our architecture meet some expectations. The most important of them is that the modules need to encapsulate a specific functionality. It makes sure that the modules are independent.

Modules need to be interchangeable i.e, adding, removing or modifying the modules should not have system wide impact. To achieve interchangeability, there must be decoupling between the modules. Decoupling is the minimizing of dependencies between different software components which facilitates easier extensibility.

Another important characteristics of extensibility is flexible configuration. It involves designing software to modify its behaviour through configuration files or settings, rather than hard coding them. Modules need to expose important configurations to the user, through an UI. This

makes sure that the configurations of the verification process are adaptable according to the user needs.

4.1.3 Usability

Principles of usability make sure that our tool is effective, simple and easy to use. Usability is the idea of designing software from the perspective of its user. The targeted user of our tool will be a developer/tester of neural network based object detection modules. Two main aspects of high usability are effectiveness, efficiency. Our tool will be considered effective if it can provide usable feedback to improve the neural network model that is being tested.

For our tool to be efficient, it must have features to make sure that resources like effort, time etc are consumed less. Some of the feature that makes our tool efficient are error handling, debugging, logging and feedback. The tool must be capable of handling errors effectively and expose them to the user to enable debugging. It should also have other features like logging errors, warning and results. The tool must be capable of providing a feedback in the form of report generation, visualization etc.

4.2 Formal Verification Method

In this section we present our choice of the formal specification language and the formal verification method that are implemented in the tool. Our goal is to do a formal verification of the neural network based object detection modules that is going to be deployed within an autonomous vehicle. We would like to verify two main properties of the module. First, we would like to verify if the object detection module has classified an object correctly. Second, when an object is correctly classified, we would like to make sure that the object was classified for the right reasons i.e When an object is classified as a car, then it only makes sense if the module has learned what are the features present in a car (eg: a wheel) and then classify only when relevant features are detected. If these two properties are verified it makes the module more trustable and even when some objects are not properly we would know the reason behind it. We can use this knowledge to update the training strategy of the classifier and make it better.

The first property is the one that is usually verified when developing a neural network. A verification set containing images that the classifier has not seen during training is used to check how well the classifier has generalized the data. Verification of the second property is very hard but it is necessary if we are going to trust the vehicle on the roads. It is challenging because of the fact that these classifiers operates on low level pixel-based inputs and their internal computational structure is uninterpretable for the most part. The features learned by the CNN can be highly entangled i.e, get mixed together in complex ways, making it hard to isolate and understand individual features.

Convolutional neural networks are a type of deep learning models, specifically designed to process images. They consist of multiple specialized layers called as convolutional layers which act as feature extractors to extract features like edges, textures etc. The convolutional layers are connected to fully-connected layers which classify the image based on the extracted features. Vision Language Models (VLM) are a type of multimodal models that are capable of processing multiple types of data simultaneously. According to their name, VLMs are capable of processing

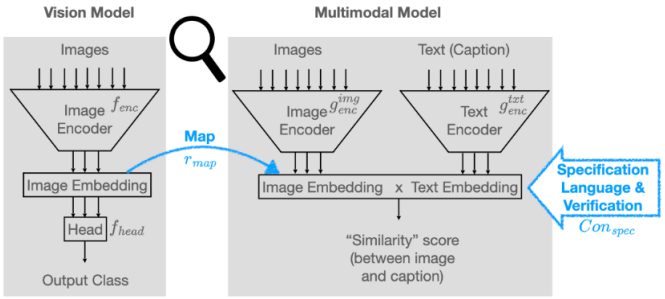

Figure 4.1: Overall verification approach[33]

vision and textual data. The VLMs consists of an image encoder and a text encoder. The image encoder extracts the features from an image and converts them to a set of image embeddings. Similarly, a text encoder processes the text to produce a set of text embeddings. The model is trained on a huge dataset consisting of pairs of images and captions. The model is trained to align the image embeddings with their corresponding text embeddings. At inference, the aligned embeddings allow the model to perform tasks such as zero-shot classification, where the model can classify images into categories based on textual descriptions without actual training. They also have other applications like image captioning and visual question answering. The output of a VLM is a similarity score between the image and the text.

The convolutional layers of the CNN can be considered as an encoder, responsible for translating pixel based image input into high level representations (features). The main idea behind the verification is to build an affine map between the image embedding space of the CNN and the corresponding image embedding space of the VLM [36]. As a consequence, the image representation space of a VLM can serve as a proxy for the representational space of the CNN. It is possible to use this setup to do a formal analysis of a CNN when we express the desired properties in text based specification language, which is then encoded via the text encoder. Thus, checking if an image satisfies certain properties reduces to checking similarity score between the image representations of the CNN, proxied by the VLM and the textual representation of the VLM.

Since the verification is performed in the common representation of image and text, this technique is scalable unlike other formal verification techniques as discussed in 2.

4.3 Formal Specification

Con_{spec} is a first order specification language that can be used to express concept-based specifications about neural network based object detection modules. Concepts are an abstraction of a combination of features or attributes of an object such as wheels for cars and ears for cats etc. In the context of VLMs, concepts are the direction of the embeddings that are learned during training. When the specifications are expressed using Con_{spec}, we can have it checked in an automated fashion. The syntax and semantics of this languages are well defined as shown in Figure 4.2 and 4.3. The syntax of Con_{spec} consists of a set of variables (images), a set of concepts and a set of classes. The set of concepts are defined in a task specific manner, based on the verification goal. The language defines two main types of predicates called the strength predicate and the

$$\begin{array}{rl} \text{(variables)} & x \in Vars \\ \text{(concept names)} & con_1, con_2 \in Concepts \\ \text{(classes)} & c \in C \end{array}$$

$$E ::= >(x, con_1, con_2) \mid predict(x, c) \mid \neg E \mid E \wedge E \mid E \vee E$$

Figure 4.2: Syntax of Con_{spec}

$$\begin{array}{rl} (\texttt{Con}_{\texttt{spec}} \text{ expressions}) & e \in E \\ \text{(classifiers)} & f \in F := \mathbb{R}^d \to \mathbb{R}^{|C|} \\ \text{(inputs)} & v \in X := \mathbb{R}^d \\ \text{(concept representation maps)} & rep \in Rep := Concepts \to (\mathbb{R}^d \to \mathbb{R}) \\ \text{(semantics)} & [\![e]\!] \in F \times X \times Rep \to \{\texttt{True}, \texttt{False}\} \end{array}$$

$$\begin{array}{c} [\![>(x, con_1, con_2)]\!](f, v, rep) := rep(con_1)(v) > rep(con_2)(v) \\ [\![predict(x, c)]\!](f, v, rep) := (argmax(f(v)) = \{c\}) \\ [\![\neg e]\!](f, v, rep) := \neg [\![e]\!](f, v, rep) \\ [\![e_1 \wedge e_2]\!](f, v, rep) := [\![e_1]\!](f, v, rep) \wedge [\![e_2]\!](f, v, rep) \\ [\![e_1 \vee e_2]\!](f, v, rep) := [\![e_1]\!](f, v, rep) \vee [\![e_2]\!](f, v, rep) \end{array}$$

Figure 4.3: Semantics of Con_{spec}

predict predicate. The strength predicate is used to express the strength of concepts in an image or a class. It is defined as $> (x, con1, con2)$, where $con1$ and $con2$ are constants from the set of concepts. The predict predicate is defined to constraint the output of the neural network based classifier. It also has a secondary predicate to check if a class or an image has a specific concept. It is defined as follows

$$\text{hasCon}(x, \text{con}) := \bigwedge_{\substack{\text{con}_i \in \text{Concepts} \\ \text{con}_i \neq \text{con}}} \succ (x, \text{con}, \text{con}_i).$$

Every Con_{spec} specification is evaluated over a triple, namely the classifier under verification, a input image, and a concept representation (VLM).

Given the predicate $> (x, con1, con2)$, it evaluates to True, if as per the VLM, strength of $con1$ is greater than $con2$. We say that a classifier satisfies a Con_{spec} specification e, if e evaluates to True for all inputs defined in an input scope B.

4.4 System Architecture

The automatic verification tool is designed to be of modular architecture as depicted in 4.4. The tool consists of three main modules namely the ontology processor, the specification generator and the verification engine. Each module has a specific role and are designed to interact seamlessly with other modules to perform the verification task. For the purposes of demonstrating the feasibility and effectiveness of this tool in automating the verification process, we have chosen the Python programming language developement of the tool because of its simplicity and support. This choice of programming language has influenced our choice of packages or tools used within each modules of the system.

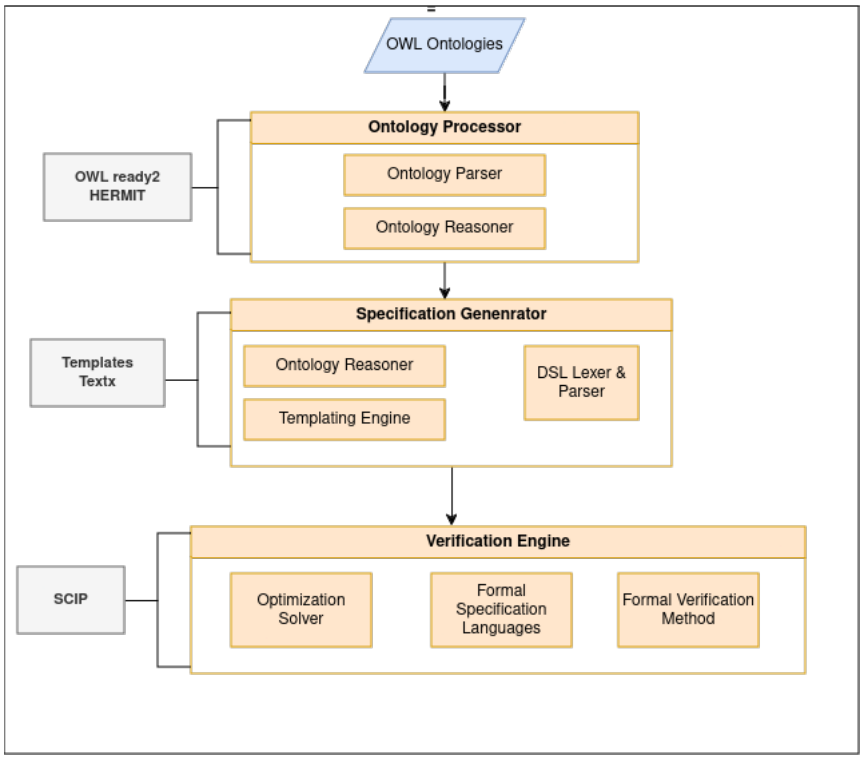

Figure 4.4: Detailed Architecture

The tool has some pre-requisites before it can start the process of automatic verification. It expects a vision ontology, ideally prepared by a domain expert to be given as an input. For the verification process, it would need a dataset containing images of all the classes it is expected to verify. The paths to these data are expected to be inputted into the configuration of the tool before it is used.

Creating, editing and maintaining ontologies are essential in fields like knowledge management, semantic web, artificial intelligence and data integration tasks. There are several tools available for these purposes like Protégé [37], TopBraid Composer [56], OntoStudio [52] and Apollo [11] etc. Among these software Protégé and Apollo are opensource and the rest are commercial.

Protégé is a popular choice for creating ontologies with a rich interface. It was created and made open source by the Stanford center for Biomedical informatics. It supports ontology file formats like Web Ontology Language (OWL) and Resource Description Framework (RDF). It also offers a wide range of plugins for additional functionalities. Protégé has integrator reasoning engines like Hermit for consistency checking, reasoning and inference. Apollo is an lightweight, easy to use ontology editor which is browser based. It provides the basic functionality for creating and managing ontologies and is best suited for simple projects.

4.4.1 Ontology Processor

The first module of our tool architecture is the ontology processor. The ontology processor has functions like loading the ontology file, parsing and extracting ontology data and managing those ontology data. It leverages inference and reasoning capabilities to extract semantically meaningful insights from the ontology data. The module queries the given ontology using any of query languages, which is used for constructing specifications and perform semantic checking on them.

Ontologies can be stored and exchanged in several different file formats, each with its own syntax and features. The most common ones are RDF and OWL, both of which are standardized formats from the World Wide Web Consortium (W3C). Hence, our parser should be able to read ontologies in these languages. There are several Python packages available for ontology manipulation. RDFlib and OWLready2 are some of the popular ones among them. RDFlib deals with the generic RDF format, while OWLready2 can handle both RDF and OWL files, as OWL is built on top of RDF.

The implementation of the tool uses OWLready2 [38], as it offers integrated reasoning capabilities. They can load OWL ontologies into Python objects, modify them and save them. Reasoning is performed by the inbuilt reasoner HermiT. OWLready2 allows for a transparent access to the OWL ontologies unlike other common JAVA-based API.

4.4.2 Specification Generator

Extracted and inferred data from the ontology processor module is passed on to the specification generator module. The role of the specification generator is to convert these data into the format of the formal specifications. The choice of our formal specification language depends upon the goal of the verification task. In this implementation of the tool, our verification goal is to verify

if the object detection module detects the objects correctly and if so, has it detected based on the right features?. For these purposes, we use the Con_{spec} language as defined in [33].

4.4.3 Con_{spec} Specification Language

As discussed above in 4.3, Con_{spec} is our formal specification language. It is capable of expressing our property of right classification using the predict predicate and the classification for the right concept using the strength predicate. This makes Con_{spec} a good choice for our verification problem.

There are also other choices of formal specification languages for our purpose of specifying the behaviour of object detection module. A light-weight implementation of the Structured Object Oriented Formal Language SOFL, used by [63] is shown to express the specifications of a traffic sign but for the purposes of synthetic data generation. Although, the same can be adapted for our purposes of formal verification. Another choice of formal specification language is the Spatio-Temporal Perception Logic (STPL) as defined in [22]. It is a slightly more sophisticated language that is capable of expressing both spatial and temporal properties of objects in a scene. Hence, it is deemed to be a better choice when attempting the formal verification of perception modules involving spatial and temporal properties. Example: In the verification of an object tracking module, one important property that needs to be verified is the ability of the module to assign a label to an object and maintain it throughout its lifetime without changing it, until it goes out of scope (out of the frame). This property can be easily expressed with STPL, leveraging its unique ability to combine both spatial and temporal properties. But for our purpose of verifying the classification and the correct features, we have chosen the con_{spec} language for its simplicity and effectiveness.

Once the formal specification language is decided, we need to create templates in the syntax of the chosen language. This is often done by the process of templating, where files or strings containing placeholders or variables are created. These placeholders can be dynamically replaced with the actual data during the rendering process. Templating is commonly used in web development process to generate HTML pages but it can also used for many other purposes. There are many templating packages in Python like Jinja2, Genshi etc but we use the straightforward Templating sub-module from the Python strings module. We then use the data collected from the ontologies to fill up these templates to create our specification language automatically.

To check if the generated specification language is correct, we have to check them for syntax and semantic errors. For that, we have to express the syntax and semantics of con_{spec} using a formal grammar description. A formal grammar is a precise mathematical description of a language's syntax and structure. It consists of a set of rules that define how valid expressions or statements in a language can be constructed [5]. Formal grammars are divided into two types based on their expressiveness. They are context-free grammars and context-specific grammars. Context-free grammars only cares about preserving the production rules and does not care for the context in which it is written. On the other hand, context-specific grammars also take the context with which they are written. Context-free grammars are comparatively simple and have less expressiveness than the other. Context-specific grammars are more complex and have more expressiveness. Context-free grammars are favoured more for their simplicity and practicality. For our purposes of verifying the syntax and semantics of the generated language, context-free grammars are sufficient.

Formal grammars can be used to specify the syntax of programming languages, domain specific languages, specification languages and other structured notations. Developing formal grammar for our specification language allows for the automatic validation of our language, ensuring that it conforms to the languages rules and constraints. To do that we have to construct a lexer and parser, to analyze and interpret our Con_{spec} language. A lexer is also known as a lexical analyzer or tokenizer. It is the first step in a analyzing formal specification language. It breaks down the input into a sequence of tokens. It identifies and categorizes individual elements like keywords, identifiers etc. It then formats the tokens into a stream for the parser to use. A parser takes in the stream of tokens from the lexer and analyzes its structure according to the grammar rules of the specification language. It does it by constructing a intermediate representation of the input like the abstract syntax tree (AST) and uses it to compare it to the formal grammar. It detects and reports any syntax errors it finds.

There are several tools and frameworks available for building domain specific languages. Some of the popular ones are Another Tool For Language Recognition (ANTLR), Xtext, JetBrains Meta Programming System (MPS), TextX etc. Our module was built using TextX. TextX is meta-language for domain specific languages (DSL) specification in Python [2]. TextX can build a DSL from a single grammar description file. It generates a meta-model in the form of Python classes and generates lexers and parsers for it automatically. TextX is based on Xtext, which is based on Java. We chose this language because of its supoort in Python language and its simplicity. The generated specifications are checked by the parsers generated from TextX. To check the semantics of Con_{spec}, we can use the Python objects created by TextX to define rules or conditions it has to pass. We can leverage ontological querying for this purpose to cross-check the generated specifications.

4.5 Verification Engine

The final and the most important part of our automatic verification tool is the verification engine. The role of the verification engine is to take the generated specification engine and encode it into a suitable verification problem and solve it using a solver. By using a VLM as a tool to analyze the CNN, we can reduce the verification problem to solving a ser of linear constraints. It relies on the assumption that the head of the CNN model or in other words the fully connected layer of the CNN is a linear layer. This assumption is true for most of the models but some may have non-linear layers. In that case, we may want to choose a powerful solver to address the non-linear constraints[33].

Here we choose the famous Contrastive Language-Image Pretraining (CLIP) model as our VLM. The key idea behind CLIP is the usage of contrastive learning to align the visual and textual data. Contrastive learning is carried out by defining a contrastive loss function, with the goal of bringing image and text embedding pairs together in the embedding space while pushing the embeddings of unrelated pairs. Typically a VLMs image encoder is a CNN or a vision transformer (ViT). Models like BERT or GPT are used for the text encoder. CLIP is used for applications like zero-shot learning, image and text retrieval, multimodal understanding etc. Zero-shot classification is the process of classifying images into categories that it has never seen during training, using only the textual description for classification. For example, given an image and five text labels like cat, dog, car etc, the CLIP model retrieves the image embeddings of the

given image and compares it to the text embeddings of the given text labels. It then assigns a class to the image based on the similarity score. This works as a zero-shot classification because the image and text embeddings are trained to be close in the embedding space if they are related.

The verification process is as follows. We start by defining an input scope or focus region within the embedding space of CLIP. A training set of images for the corresponding classes to be verified is taken and their image embeddings are found. The mean of these embeddings are considered to be the mean of the embedding space within the CLIP model of that particular class. We set up an upper and lower bound to every dimension of the embedding vector based on this mean embedding vector. Given a Con_{spec} specification like $predict(c) = con1 > con2$, we attempt to to solve $(predict(c) = con1 > con2)$ is equivalent to $predict(c) \bigwedge (con1 > con2)$ by encoding it via constraint integer programming. We then use any off-the-shelf solver like SCIP[6] to solve this problem. If no solution is found, it means that the property holds and when a solution exists, it indicates a counterexample.

4.6 Implementation

There are two objectives for the experiment. They are to analyze the zero shot classification performance of the CLIP model and to analyze the performance of the tool in performing an automatic verification of the zero shot CLIP classifier.

4.6.1 Ontology and Dataset Preparation

Before performing the automatic verification of the CLIP model, we first have to prepare the vision ontology and the dataset that is to be used in the verification process. The ontology serves as the foundational knowledge base, encompassing various aspects of he driving environment of the autonomous vehicle. It includes the definition of objects, their features or concepts and their relationship, which are necessary for generating formal specifications for the verification process.

Based on the scope and goal of the verification, we define the key components of the ontology. For example, we can have different classes representing the objects, climatic conditions and different driving scenarios. For the purpose of the experiment, we define only the objects and their features of both the RIVAL10 dataset and GTSRB dataset. For RIVAL10, We start the ontology development by defining the primary classes like objects, concepts and their relationship. The class objects consists of all objects present in the dataset. The set concepts consists of all the features or concepts of all things in the object class. There is also an intermediate class connecting the objects and their features called as the object concepts. For example, It has entities like cat tail, dog ears etc. The classes of the ontologies for RIVAL10 can be seen in Figures 4.5, 4.6, 4.7.

Similarly, for GTSRB we can create the ontology by defining the primary classes like colors, shapes, content, border and traffic signs. The colors class is a set of all colors, the class shapes is the set of all shapes found in the traffic signs. The class border is a set of all borders. The border class has properties of a shape (border shape) and color (border color). The class content is a set of contents found at the center of a traffic sign. It has the sub-classes text, number and graphic which are common contents of a traffic sign. The class traffic_signs is the set of all traffic signs that we wish to verify. The classes of the GTSRB can be seen in Figures 4.8, 4.9.

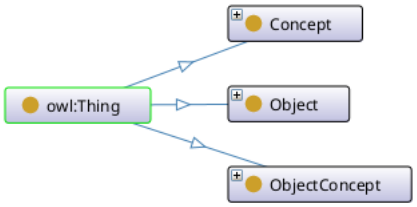

Figure 4.5: Primary classes of the ontology

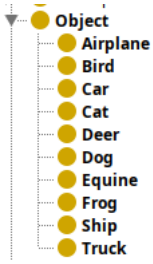

Figure 4.6: The object class

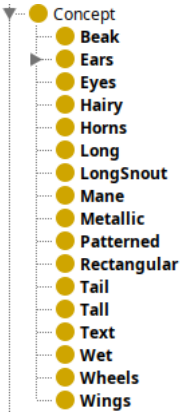

Figure 4.7: The concept class

To perform the zero-shot classification, we need both the RIVAL10 [35] dataset for generic classes and the GTSRB [50] dataset for traffic signs. We will be using the test subset of both the datasets for our zero-shot classification task. The verification procedure requires datasets only to identify the focus region in the CLIP representation space. More on focus regions is discussed in 4.5. For this purpose we may use the train subset of both the datasets. To make these datasets usable, we have to make sure that the images of a class are put in a folder with name as their class. Most of the datasets are already provided in this format. But for the RIVAL10 dataset, it had to be done manually. The folder names of the GTSRB classes had to be changed from their class ids to their class names. Sample images from both the datasets can be seen in Figures 4.10, 4.11.

4.6.2 Procedures

The procedures for the zero-shot classification task and the CLIP verification tasks are discussed in detail below.

Zero-Shot Classification

Zero-shot classification is the ability of a model to correctly classify data that it has never encountered during training. Models with good zero-shot performance are able to generalize the

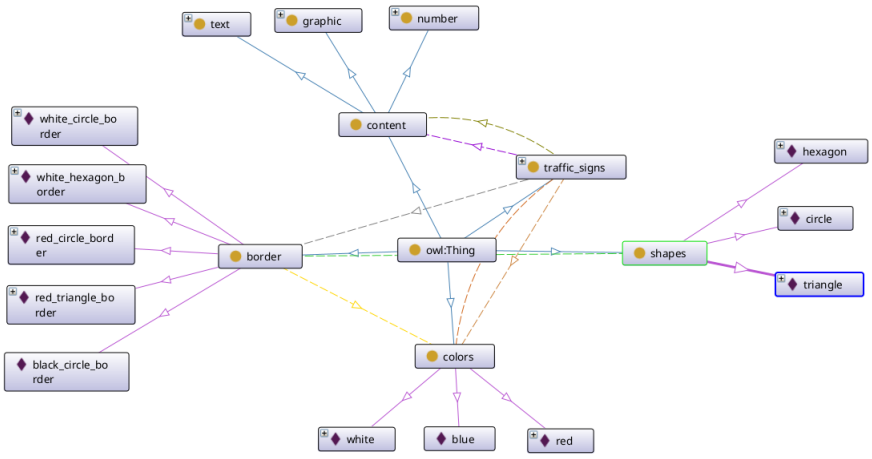

Figure 4.8: Primary classes of traffic sign ontology

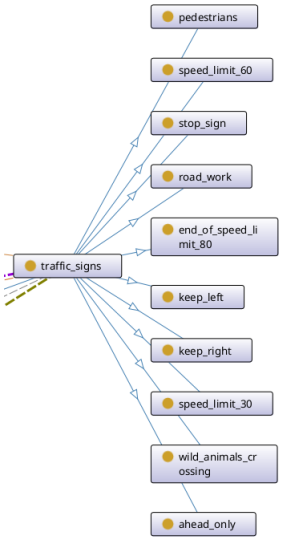

Figure 4.9: The traffic signs class

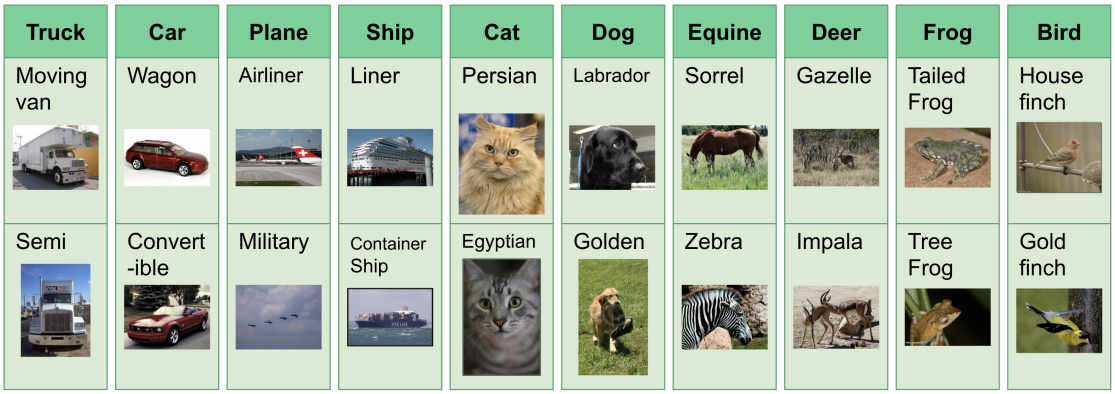

Figure 4.10: Sample images from RIVAL10 dataset [35]

Figure 4.11: Sample images of GTSRB dataset [50]

unseen classes well from their seen classes based on some form of semantic understanding. Contrastive Language-Image Pre-training (CLIP) is a Vision Language Model (VLM) developed by OpenAI and it is capable of performing zero-shot classification. CLIP model is trained using approximately 400 million image-caption pairs, which makes the model generalize well among classes.

During training, the CLIP model learns to maps the images and text in a shared embedding space. The training is designed in a way to bring together related classes and increase the distance between unrelated classes within its embedding space. Before beginning the zero-shot classification, we need to prepare the set of labels or captions that the model is going to choose from. Ideally, these labels are the name of the classes that we need to classify. As an alternative, we can create a set of captions that describes different types of images of a particular class. Examples of both techniques are shown below.

$$class\_labels = [\text{"}cat\text{"}, \text{"}car\text{"}, \text{"}dog\text{"}, \text{"}truck\text{"}, \text{"}airplane\text{"}]$$

$captions = [\text{"}a\ photo\ of\ a\ cat\text{"}, \text{"}a\ photo\ of\ a\ car\text{"}, \text{"}a\ photo\ of\ a\ dog\text{"}, \text{"}a\ photo\ of\ a\ truck\text{"}, \text{"}a\ photo\ of\ an\ airplane\text{"}]$

Test images for the classes to be classified also needs to be prepared. We use the dedicated test datasets of RIVAL10 and GTSRB. As discussed in 4.5, the CLIP architecture has two encoders, one for each modality (images and text). The collection of labels or captions are passed on to the text encoder of CLIP and their respective embeddings are collected. The image that needs to be classified is then passed to the image encoder of CLIP and its embedding is collected. The class of the image is identified by performing a similarity calculation called as cosine similarity. Cosine similarity is a measure of similarity between two non zero vectors in an inner product space. It is defined as the cosine of the angle between the two vectors, ranging from -1 (exactly opposite) to 1 (exactly same), with 0 indicating orthogonality (no similarity) [33]. Mathematically it is calculated using the dot product of the vectors divided by the product of their magnitude.

The cosine similarity is used to compare the representation of the image with the embeddings of texts of different class labels. The class whose text embeddings has the highest cosine similarity with the image is chosen as the image class. Thus CLIP model is able to do zero-shot classifications as it generalizes to new model classes by comparing their similarity with their corresponding text embeddings.

Zero-shot CLIP verification

After the preparation of the vision ontology and the datasets, we can use the tool to generate the the Con_{spec} specifications automatically. These specification are generated according to the grammar as described in Appendix A. To do a verification of the CLIP zero-shot classifier, we first have to identify interesting regions in the embedding space of the CLIP model. These interesting regions are called the focus regions. The focus region is defined as $\overline{B} := [[l_1, u_1]...[l_p, u_p]]$, where l and u are the lower and upper limit in the embedding space for all dimensions in the embedding space. They are defined for all the classes that need to be verified and are used to check if the defined specifications hold in those regions.

To define the focus region, we pass a dataset of the class we need to verify to the clip model and collect their embeddings. A mean embedding is calculated for all the embeddings and the lower and upper limits are calculated as shown below

$$l_i = \mu_i - (\gamma * \sigma_i)$$
$$u_i = \mu_i + (\gamma * \sigma_i)$$

Gamma is an variable that decides the size of the focus region.

Encoding of the verification problem

To verify our model, we need to first encode the Con_{spec} specifications into a verification problem. These specifications are encoded using constraint integer programming, and SCIP [6], an off the shelf solver is used to solve it. For example, if we have the specification $predict(c) \Rightarrow con1 > con2$, then we attempt to solve $predict(c) \wedge (con1 > con2)$ or $predict(C) \wedge \neg(con1 > con2)$. If no solution is found, it means that the property holds, while a solution indicates a violation as there exists a counterexample that violates that specification within than focus region.

Thus the overall verification problem can be expressed by the following inequalities.Equation 4.1 expreses that the verfication must hold within the foucs region. Equation 4.2 represents that within the focus region the model should classify the correct class without any misclassification with the other classes.Equation 4.3 expresses that the relevant concept should be strongly represented within the focus region when compared with irrelevant concepts

$$l_i \leq z_i \leq u_i, l_i = \mu_i - (\gamma * \sigma_i), u_i = \mu_i + (\gamma * \sigma_i), \quad \forall z_i, i \in 1...p, \tag{4.1}$$

$$\sum_i \frac{z_i}{\|z\|} \frac{q_i^c}{\|q^c\|} > \sum_i \frac{z_i}{\|z\|} \frac{q_i^{c_j}}{\|q^{c_j}\|}, \quad \forall c_j \neq c \tag{4.2}$$

$$\sum_i \frac{z_i}{\|z\|} \frac{q_i^{con_2}}{\|q^{con_2}\|} > \sum_i \frac{z_i}{\|z\|} \frac{q_i^{con_1}}{\|q^{con_1}\|} \tag{4.3}$$

4.7 Metrics

Before beginning our experiment, it is important to clearly define the metrics used to evaluate the performance of the CLIP model in both zero-shot classification and the efficiency of the tool in automatically verifying the clip model. Metrics provide a quantifiable means to assess the effectiveness and reliability of the model and our tool. This section will outline the metrics for our two verification goals.

4.7.1 Zero-shot classification metrics

Zero-shot classification evaluates the model's ability to correctly classify images into the defined classes, without having encountered any training images for those classes. The following metrics can be used to assess the zero-shot classification performance of the CLIP model.

Accuracy

Accuracy for a classification task is defined as the ratio of correctly predicted images to the total images in the dataset. Accuracy provides an overall measure of the model's performance across all classes.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Precision

Precision is defined as the ratio of correctly predicted positive instances of the class to the total predicted instances of the class. Precision is crucial for understanding the accuracy of correct predictions, especially in cases where false positives can critical damage.

$$Precision = \frac{True\ positives}{Total\ predictions\ of\ a\ class}$$

4.7.2 Automation Tool And Verification Metrics

The verification of CLIP involves checking if the Zero-shot CLIP classifier can adhere to the defined specifications within the defined embedding space. The following metrics can be used to assess the tool and the verification process.

Specification Compliance Rate

Specification compliance rate is defined as the ratio of number of specifications held to the total number of specifications. This metric can provide an overall measure of how well a model meets the defined specifications.

$$Specification\ ComplianceRate = \frac{Number\ of\ specifications\ held}{Total\ number\ of\ defined\ specifications}$$

Specification Violation Measure

Not all specifications are verified by the model, there are and can be some violations in the defined specifications. They often reveal useful insights that can be used to retrain the model. The specification violation measure epsilon (ϵ) quantifies the degree to which a model's behaviour deviates from the specified properties. It is used to assess how closely the model adheres to the specifications within the defined focus region.

A lower epsilon indicates that the model's behaviour is closer to the specifications, while a higher epsilon suggests a significant deviation. The epsilon also helps in identifying useful counter-examples for the model. The violation measure is only valid within the defined focus

region B.

Apart from the verification task, it is important to evaluate the effectiveness of the automatic specification generation and the verification tool. Metrics like the correctness and completeness of the specifications generated may be assessed qualitatively.

Complete implementation of the tool can be found here in GitHub [55].

Chapter 5

Results and Discussion

In this chapter, we use the developed automatic verification tool to perform verification of the zero-shot classification of the CLIP model. There are two goals for our experiment. First, we test the zero-shot performance of the CLIP model in identifying the classes present in the Rich Visual Attributes with Localization (RIVAL10) [35] dataset and the German Traffic Sign Recognition Benchmark (GTSRB) [50] dataset. RIVAL10 dataset was selected to assess the zero shot performance of the CLIP model in generic classes like cats, dogs etc., and the GTSRB dataset was chosen to assess the zero shot performance of CLIP in industry specific application.

Our second goal is to perform a verification of the CLIP model, by using our automatic verification tool. We will use the tool to automatically generate the specifications and verify them for the CLIP model. Five classes from RIVAL10 dataset and 8 classes from GTSRB dataset are chosen for our experiments. They are cat, car, dog, truck and airplane from the RIVAL10 dataset and stop, yield, speed limit 60kmph, end of speed limit 80kmph, children crossing, pedestrians and wild animal crossing. The first set of classes are particularly chosen to see how well the CLIP model is capable of differentiating animals from different modes of transport. The second set of classes are chosen to see how the CLIP models perform on different types of content present in the traffic signs. The stop sign has text, speed limit signs have numbers, children crossing has a graphic and the yield sign has no content.

This chapter presents the results and outcomes of the experiment, implications and contribution of the work, directions for future research.

5.1 Zero-shot Classification results

Zero-shot classification of the CLIP model was tested using the RIVAL10 and GTSRB datasets. The classes of cat, car, dog, truck and airplane were tested from the RIVAL10 dataset and the classes stop, yield, children crossing, speed limit 30kmph, speed limit 60kmph, end of speed limit 80kmph, pedestrians and wild animals crossing were tested from the GTSRB dataset. Accuracy and precision were measured for each class and is used to assign a performance rating for the zero-shot CLIP model. The results of the classification can be found in table 5.1 and 5.2.

As we can see from the results, the zero-shot CLIP classifier is able to perform much better in the RIVAL10 dataset than the GTSRB dataset. The CLIP model was trained of approximately

400 million pairs of images and captions that are scraped from the internet. This difference in the classification may be because of the class imbalance in its training dataset. There is a notable pattern in its recognition of the traffic signs. The CLIP model is able to perform better on signs with text on them (eg: stop), followed by signs with numbers on them (Eg: the speed limit signs). It performs rather poor on traffic signs with a graphic on them except for few signs like the children crossing.

Class	Total no of images	Accuracy	Precision
Cat	2122	97%	99%
Dog	2119	98%	97%
Car	2126	91%	96%
Truck	2011	97%	90%
Airplane	2110	88%	99%

Table 5.1: Classification results of RIVAL10

Class	Total no of images	Accuracy	Precision
Stop	780	89%	98%
Speed limit 30 kmph	2220	64%	90%
Speed limit 60 kmph	1410	52%	97%
End of speed limit 80 kmph	420	0%	0%
Children crossing	540	88%	36%
Yield	2160	26%	77%
Pedestrians	240	0%	0%
Wild animals crossing	780	41%	94%

Table 5.2: Classification results of GTSRB

5.2 Verification Results

The verification of the CLIP model will also be done for the generic classes from the RIVAL10 dataset and the traffic sign classes from the GTSRB dataset. The gamma factor defines the area of the focus region in which the verification will be performed. The verification of the signs from RIVAL10 dataset will be done with epsilon values $\epsilon \in [0.25, 0.40, 0.50]$. We will be verifying if the specification of type $predict(c) \Rightarrow (con1 > con2)$ hold within the focus regions of different gamma value. If a specification is violated, then we report on the violation measure (ϵ), for the violated specifications. Complete specifications that was automatically generated by the tool can be found in Appendix B

We redefine the specification $predict(c) \Rightarrow (con1 > con2)$ as $predict(c) \wedge (con1 > con2)$. We can divide the verification task into two, verifying the specification $predict(c)$ first and then the specification $(con1 > con2)$. The results of the verification for the classes cat, dog, car, truck and airplane of the RIVAL10 dataset for focus regions with ϵ 0.25, 0.30, 0.40 can be found in Tables 5.3, 5.4 and 5.5. The results of the verification of $predict(c)$ is either a pass or fail and it is indicated with a ✓for when the specification holds or with ✗when the specification is violated. We refer to the class that is currently being verified as the *Class Under Verification (CUV)* and

Class Under Verification (CUV)	Other Classes	Verification Result
Cat	Dog	✓
	Car	✓
	Truck	✓
	Airplane	✓
Dog	Cat	✓
	Car	✓
	Truck	✓
	Airplane	✓
Car	Cat	✓
	Dog	✓
	Truck	✗
	Airplane	✓
Truck	Cat	✓
	Dog	✓
	Car	✓
	Airplane	✓
Airplane	Cat	✓
	Dog	✓
	Car	✓
	Truck	✓

Table 5.3: Verification results for $\epsilon = 0.25$ for RIVAL10

other remaining classes as *Other Classes*. For example, when the verification result is ✓when comparing cat (CUV) with some other class like dog (Other Classes). for $\epsilon = 0.25$, it means when an image falls under the focus region, we can guarantee that the model will not misclassify a cat for a dog. When the result is ✗, it means that our solver has found at least one counter example, where the class cat is misclassified as a dog.

Now, we can verify the specification ($con1 > con2$). This type of specification is called as a strength predicate as it compares the strength of the concepts inside different focus regions. The results of this verification is a violation measure ϵ. It is called as a slack variable and it is formulated as part of the optimization problem when the specifications are encoded using constraint integer programming. The objective of the SCIP solver is to maximize ϵ, while verifying the constraints of the specifications. It is used as a metric to measure the violation strength of a strength predicate. A lower ϵ value indicates that the model is more likely to make the classification because of *Con1* rather than *Con2*. We have selected two relevant concepts or features for each class and they will be compared to the relevant concepts of other classes. For example, the relevant concepts of cat are eyes and ears. They will be compared will all other relevant features of other classes like wheels, wings etc as they are irrelevant for the class cat. There are cases where some concepts are relevant for more than one class like wheels being in common for cars and trucks. In that case, we try to apply the most relevant feature of that class. The results of the second part of the verification are provided in Figures 5.1, 5.5, 5.3, 5.4 and 5.2. Violation measures ϵ is provided in the y-axis and the strength predicates comparing different concepts are present in the x-axis.

Similarly, verification of the traffic signs from GTSRB dataset will be done with epsilon values $\epsilon \in [0.20, 0.30, 0.40]$. When a specification is violated, we report on the violation measure (ϵ), of

Class Under Verification (CUV)	Other Classes	Verification Result
Cat	Dog	✓
	Car	✓
	Truck	✓
	Airplane	✓
Dog	Cat	✓
	Car	✓
	Truck	✓
	Airplane	✓
Car	Cat	✓
	Dog	✓
	Truck	✗
	Airplane	✓
Truck	Cat	✓
	Dog	✓
	Car	✓
	Airplane	✓
Airplane	Cat	✓
	Dog	✓
	Car	✗
	Truck	✗

Table 5.4: Verification results for $\epsilon = 0.30$ for RIVAL10

Class Under Verification (CUV)	Other Classes	Verification Result
Cat	Dog	✗
	Car	✓
	Truck	✓
	Airplane	✓
Dog	Cat	✓
	Car	✓
	Truck	✓
	Airplane	✓
Car	Cat	✗
	Dog	✗
	Truck	✗
	Airplane	✗
Truck	Cat	✗
	Dog	✗
	Car	✗
	Airplane	✗
Airplane	Cat	✗
	Dog	✗
	Car	✗
	Truck	✗

Table 5.5: Verification results for $\epsilon = 0.40$ for RIVAL10

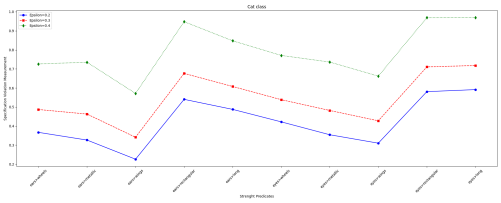

Figure 5.1: Strength predicate results for cat

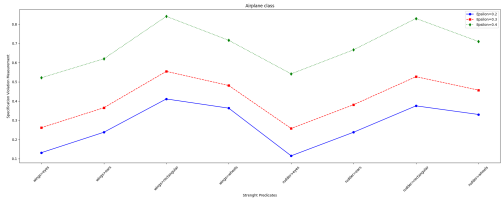

Figure 5.2: Strength predicate results for airplane

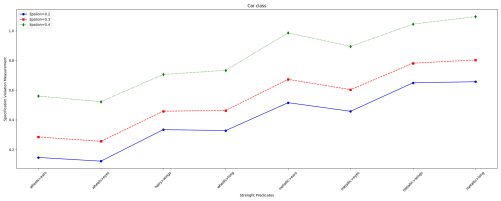

Figure 5.3: Strength predicate results for car

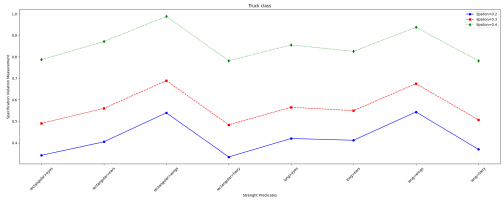

Figure 5.4: Strength predicate results for truck

the respective specifications. The results of the verification for the classes stop, speed limit 30 kmph, speed limit 60 kmph, end of speed limit 80 kmph, children crossing, pedestrians and wild animals crossing of the GTSRB dataset for focus regions with ϵ 0.20, 0.30, 0.40 can be found in Tables 5.6, 5.7 and 5.8. The results of the strength predicate for the traffic sign classes can be found in Figures 5.12, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11.

5.3 Discussion

In this section, we discuss the results of the zero-shot classification and the verification process. It can be seen that, the CLIP model has a good performance in the RIVAL10 classes compared to the GTSRB. It may be because of under-representation of traffic sign in the training of the CLIP model. But it is not sure how the 400 million images used to train the CLIP model is distributed. In the GTSRB dataset, the model can be seen have high zero-shot performance with traffic signs having text or number in them. This may be because of their ability to recognize texts from their other training images. The classes of end of speed limit 80 kmph and pedestrian crossing are seen to have 0% accuracy. But later in the verification results, we will see that the class pedestrians performs as equally or more than the class children crossing. The very low accuracy in zero-shot classification may be attributed to the low number of samples used to test the model.

5.3.1 Verification of RIVAL10 classes

The verification was performed for the classes cat, dog, car, truck and airplane from the RIVAL10 dataset for the focus regions 0.25, 0.30 and 0.4. In the focus region 0.25, all the classes of RIVAL10 were fully verified excepts for the car class. It has failed against the class truck. It

Class Under Verification (CUV)	Other Classes	Verification Result
Stop	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Children crossing	✓
	Pedestrians	✓
	Wild animals crossing	✓
Speed limit 30 kmph	Stop	✓
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✓
	Pedestrians	✓
	Wild animals crossing	✓
Speed limit 60 kmph	Stop	✓
	Speed limit 30 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✓
	Pedestrians	✓
	Wild animals crossing	✓
End of speed limit 80 kmph	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	Children crossing	✗
	Pedestrians	✗
	Wild animals crossing	✗
Children crossing	Stop	✓
	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Pedestrians	✗
	Wild animals crossing	✓
Pedestrians	Stop	✓
	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Children crossing	✗
	Wild animals crossing	✓
Wild animals crossing	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✗
	Pedestrians	✗

Table 5.6: Verification results for $\epsilon = 0.20$ of GTSRB

Class Under Verification (CUV)	Other Classes	Verification Result
Stop	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Children crossing	✓
	Pedestrians	✓
	Wild animals crossing	✓
Speed limit 30 kmph	Stop	✓
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✓
	Pedestrians	✗
	Wild animals crossing	✓
Speed limit 60 kmph	Stop	✗
	Speed limit 30 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✓
	Pedestrians	✗
	Wild animals crossing	✓
End of speed limit 80 kmph	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	Children crossing	✗
	Pedestrians	✗
	Wild animals crossing	✗
Children crossing	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Pedestrians	✗
	Wild animals crossing	✗
Pedestrians	Stop	✓
	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Children crossing	✗
	Wild animals crossing	✗
Wild animals crossing	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✗
	Pedestrians	✗

Table 5.7: Verification results for $\epsilon = 0.30$ of GTSRB

Class Under Verification (CUV)	Other Classes	Verification Result
Stop	Speed limit 30 kmph	✓
	Speed limit 60 kmph	✓
	End of speed limit 80 kmph	✓
	Children crossing	✓
	Pedestrians	✗
	Wild animals crossing	✓
Speed limit 30 kmph	Stop	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✗
	Pedestrians	✗
	Wild animals crossing	✓
Speed limit 60 kmph	Stop	✗
	Speed limit 30 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✓
	Pedestrians	✗
	Wild animals crossing	✓
End of speed limit 80 kmph	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	Children crossing	✗
	Pedestrians	✗
	Wild animals crossing	✗
Children crossing	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Pedestrians	✗
	Wild animals crossing	✗
Pedestrians	Stop	✓
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✗
	Wild animals crossing	✗
Wild animals crossing	Stop	✗
	Speed limit 30 kmph	✗
	Speed limit 60 kmph	✗
	End of speed limit 80 kmph	✗
	Children crossing	✗
	Pedestrians	✗

Table 5.8: Verification results for $\epsilon = 0.40$ of GTSRB

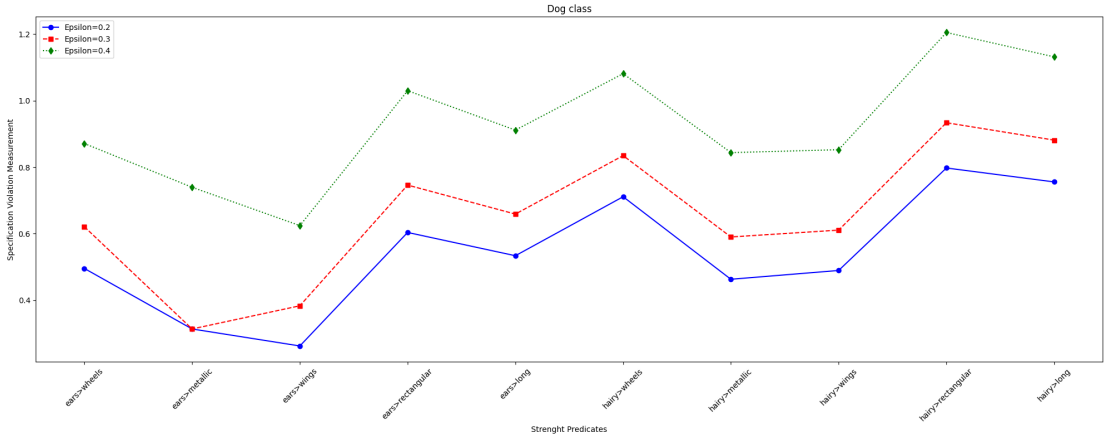

Figure 5.5: Strength predicate results for dog

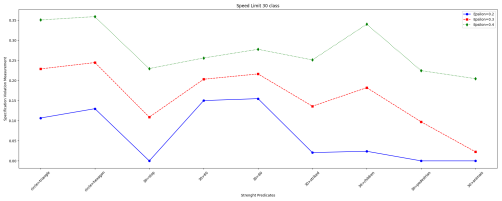

Figure 5.6: Strength predicate results for speed limit 30kmph

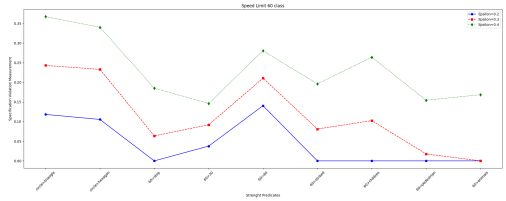

Figure 5.7: Strength predicate results for speed limit 60kmph

essentially means that when the model encounters an image which falls under the focus region of 0.25, then there are possibilities of misclassifying a car as a truck. While, the other classes are guaranteed to be classified correctly, if the input image falls into the focus region. If the embedding space of the CLIP model is assumed to be a Gaussian distribution then the focus region 0.25 accounts for about 19.74% of the embedding space of a class. When a verification holds in focus region 0.25, then it means that nearly 20 percent of the total embedding space of that particular class has been verified.

In the strength predicate verification, the strength predicates enforce the constraint that $Con1$ is greater than $Con2$ and the violation measure ϵ indicates how likely that constraint can be verified. The results of strength predicate verification should be considered along with the results of the $predict(c)$ specification. The results of the strength predicate of the class car indicates that, the concept wheel is learned strongly than the class metallic. This and the other insights indicate the models internal behaviour in deciding the classes of the objects.

Similarly, for the focus region 0.3, we can see that the class airplane has started to get confused with the classes car and truck. Strength predicate results indicate that the concepts like rectangular, which are seen to be performing well with the class truck is having high violating rates for the class airplane. Likewise, the concept wheels which have performed well for the class

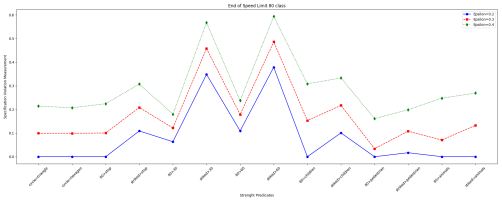

Figure 5.8: Strength predicate results for end of speed limit 80kmph

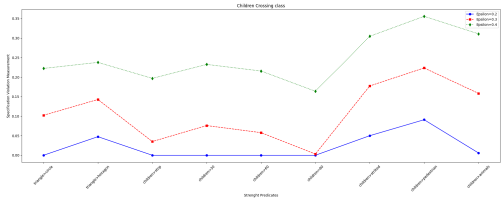

Figure 5.9: Strength predicate results for children crossing

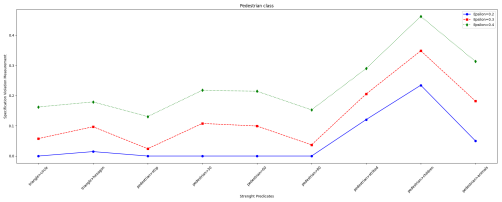

Figure 5.10: Strength predicate results for children pedestrians

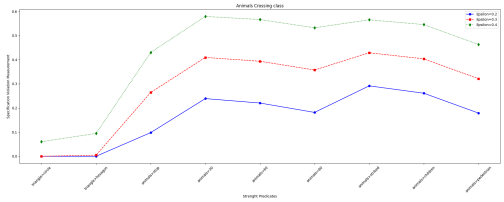

Figure 5.11: Strength predicate results for wild animals crossing

car also has high violations for class airplane. This might be the reason for its confusion with classes car and truck. In focus region 0.4, the classes car, truck and airplane have completely failed while the class cat has failed against only dog. The dog seems to be only class that is verified for all three focus regions.

5.3.2 Verification results of GTSRB classes

The verification was performed for the signs stop, speed limit 30 kmph, speed limit 60 kmph, end of speed limit 80 kmph, children crossing, pedestrians and animals crossing from the GTSRB dataset for the focus regions 0.2, 0.3, 0.4. It is worth noting that the CLIP model has a very poor performance on the GTSRB dataset compared to the RIVAL10 dataset and hence we try to keep the first focus region to 0.2. In focus region 0.2, stop is the only sign that was verified to be correctly classified against all other classes. The results indicate other interesting patterns like the sign with a number in them (speed limit signs) have failed the verification against other speed limit signs. This shows that the CLIP model has not yet learned to associate the numbers to their respective signs. Numbers are the most important feature present in a speed limit traffic sign, followed by shapes and border. The children crossing sign and the pedestrians signs are being confused with each other. Again, this shows that the graphic present in the signs are not associated well with the signs. Finally the signs, wild animals crossing and end of speed limit 80 kmph have failed entirely. This shows that the signs are not at all learned compared to the other ones.

The strength predicate results of the stop class shows that, the shape circle is more likely to be confused with the shape hexagon of the stop sign. While the possibility of other concepts like numbers and graphics misclassifying for stop is 0. Meaning that we can guarantee that the

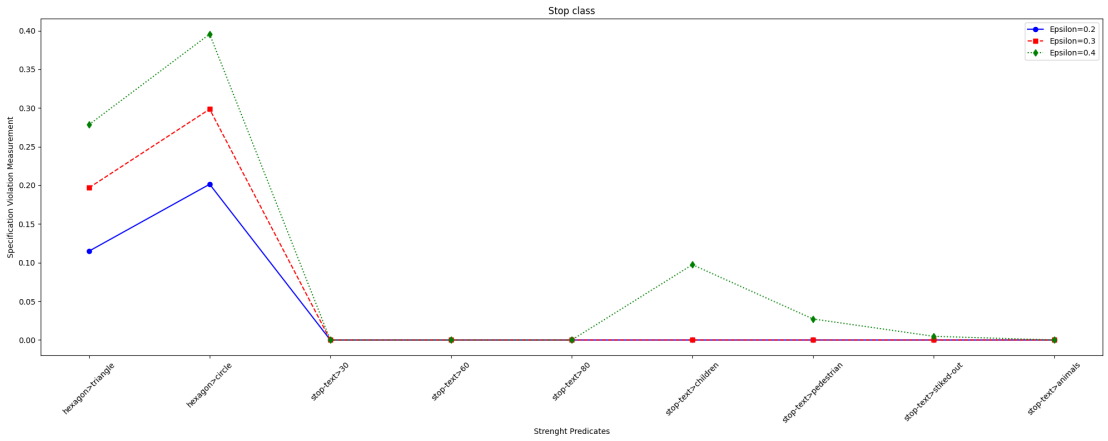

Figure 5.12: Strength predicate results for stop

stop sign is being classified because of the text STOP present in them compared to the other concepts within the focus region 0.25. The results of signs speed limit 30 kmph shows that there is high possibility that the shape hexagon to be confused with its circle shape. It is the other way around for Speed limit 60 kmph, where the shape triangle is more likely to violate the shape circle. Another interesting point to note is how the speed limit 30 signs can be confused for both the numbers 60 and 80 instead of 30 while the sign speed limit 60 is more likely to be confused by 80 than 60. This shows that the concept number is more strongly learned than its shape circle for sign speed limit 60 kmph and vice versa for the sign speed limit 30 kmph. As expected, the sign children crossing is expected to get confused with the pedestrian graphic and the pedestrians graphic getting confused for children graphic, resulting in their mutual misclassification. The classes animals crossing and end of speed limit 80 kmph seems to have learned the concepts circle well while having very high violations with their main content. This is a clear example that there is a priority among the relevant concepts. The main content of the traffic signs need to be given more importance than their shapes.

In focus regions 0.3, the speed limit signs starts to gets confused with other text-based and graphic-based signs. Children crossing has completely failed here and the class pedestrians is performing more consistently with only violations with wild animals crossing sign. There are no changes with the stop sign. In focus region 0.4, the stop sign begins to gets confused with the pedestrians sign. All other classes have either completely failed or has managed to pass only against one class. The pedestrian class has consistently performed well against the stop sign in all three focus regions and the speed limit signs have consistently performed well against wild animals crossing sign. This may also be because of the fact that the wild animals crossing class is underrepresented in the CLIP model.

5.4 Limitations and Future Directions

This section discusses some of the limitations and the directions for future research.

The verification engine of the automation tool is based on the idea of using a Vision Language Model (VLM) as a lens through which we can assess Convolutional Neural Networks. Thus the quality of the verification process depends upon the performance of the VLM being used. In our case, we use the CLIP model from OpenAI as our VLM. It is approximately trained with 400 million images-caption pairs scraped from the internet. Since we don't have access to the training dataset used to train the CLIP model in detail, we can't be sure to understand how many of the images that are relevant to the autonomous driving scenario is present in them.

Our choice of captions used to refer to the concepts of a class are very important in the verification process. The CLIP model may have to be retrained or tuned to have high correlation with the concepts that we use in the verification process for it to be effective. This can be done by carefully constructing a dataset of image-caption pairs to retune the CLIP model, but it requires a large amount of data to for that.

The vision ontology that we use to automatically generate the formal specifications are based on the vision ontology. It is expected that a domain expert creates and maintains that ontology. Since the tool is intended to be used as an aid in the AV homologation process, we expect that the entities and their relationship in the ontologies are verified in some manner or other before it is being used by the tool.

The verification engine based on the idea of concept based analysis of neural network from [33]. For demonstrating the working of the tool chain, we performed an verification on the zero-shot classifier of the CLIP model itself. When applied to practical situations, the zero-shot CLIP classifer needs to be replaced by the CNN model that is used in the vehicle. The paper presents techniques to perform verification for this scenario as well. The output of the convolutional layer of our CNN will be mapped to the representational space of the CLIP model. This can be done by learning an affine mapping to align the representation space of the CNN with the representational space of the CLIP model. This map will be trained using gradient descent to minimize the difference between the two spaces. Future research in this direction could be to verify a CNN module deployed in a vehicle using our automation tool.

Ontologies have started to be widely used in the autonomous vehicle development process. They are preferred for their ability for complex data to be modelled and their inference and reasoning abilities reduce the need for explicit data input. Ontologies are already widely used in several of the tasks like scenario generation for simulation-based testing. Recent works like [4] have concentrated on standardizing and formalization of the domain knowledge related to the automotive industry. The vision ontology that is used in this tool can be combined with the existing frameworks like the ASAM OpenXOntology for expanding the scope of the tool from functional verification of object detection modules to other areas.

Conclusion

The thesis focused on the addressing three main problems present in the widespread adaptation of formal verification of AI modules in autonomous vehicles. The three problems are lack of scalable formal verification methods for object detection, lack of formal specification methods for CNNs and the lack of tool chain for complete verification process, from generation of formal specifications to their formal verification.

The thesis presents an automated tool chain that utilizes ontologies to generate semantically meaningful formal specifications and formally verified them by using Vision Language Models (VLM). Ontologies are used to represent the objects present in the driving environment of the vehicle and the features that defines them. The tool uses ontology parser and reasoners to read and query meaningful data from ontologies to generate specifications in a language called Con_{spec}, specifically designed for expressing properties of neural network based object detection modules. The tool uses CLIP model to evaluate if the zero-shot classification is able to satisfy the expected properties expressed in the form of human understandable concepts.

The specifications are then encoded into a verification problem using constraint integer programming and uses the SCIP solver to solve them. The functionality and effectiveness of the tool is evaluated by verifying the zero-shot CLIP classifer for generic classes like cats, dogs, cars etc., and for traffic sign classes like stop, children crossing, speed limits etc. The tool was able to identify the classes that are most likely to be violated for each class. It also reveals insights into the learning of the CLIP model to classify the object based on their features or concepts.

The thesis has made significant contributions by presenting a first of its kind tool for complete verification of neural network based object detection modules. The idea of using ontologies for the purpose of formal specifications is unique and supports the trend of using ontologies to model domain specific knowledge. The tool has developed formal grammar for the formal specification language Con_{spec} in TextX and uses it for the syntax and semantic checks of the generated specifications. The tool is validated by verifying the zero-shot CLIP model for the traffic sign classes of the GTSRB dataset, thereby identifying the effectiveness of the model in learning the classes.

The tool is developed to be modular and is easily extendable to formal verification of different AI models. The tool presented here represents a significant step in the direction of automated formal verification. Future research direction of this work can be to extend the capabilities of the tool to verify CNN modules instead Zero-shot VLM models.

Appendix A

Con_{spec} Grammar

Grammar defining the Con_{spec} specification language in TextX

```
ConspecModel:
    (elements+=Elements)*
;

Elements:
    Type | Module
;

Module:
    'Module' name=ID triple=Triple '{'
    (specifications+=Specification)*
    '}'
;

Specification:
    'E' name=ID '::>'
    (expression_lhs+=Expression)*
    '=>'
    (expression_rhs+=Expression)* |
    'E' name=ID '::>'
    (expression_lhs+=Expression)*
;

Expression:
    ORExpression
;

ORExpression:
    left=ANDExpression ('v' right=ANDExpression)*
;

ANDExpression:
    left=NotExpression ('^' right+=NotExpression)*
;

NotExpression:
    '~' not_exp=NotExpression | exp=PrimaryExpression
;

PrimaryExpression:
```

```
    '(' Expression ')' | StrengthPredicate | Predict | Membership
;

StrengthPredicate:
    '>' '('
    cls=[Class]
    ','
    concept_prior=[Concept]
    ','
    concept_after=[Concept]
    ')'
;

Predict:
    'predict' '('
    cls=[Class]
    ')'
;

Membership:
    'hasCon' '('
    concept+=[Concept]
    ')'
;

Type:
    Image | Concept | Class | Network | ConRep | Triple | ImgCon | Comment
;

Image:
    'img' name=ID
;

Concept:
    'con' name=ID | INT
;

Class:
    'class' name=ID
;

Network:
    'network' name=ID
;

ConRep:
    'rep' name=ID
;

Triple:
    '(' net=[Network] ',' inp=[Class] ',' rep=[ConRep]')'
;

ImgCon:
    'imageCon' '(' img=[Image] ')' '=' '{' concept+=[Concept][','] '}'
;

Comment:
    /#.*$/
;
```

Appendix B

Con_{spec} Specifications

Complete Con_{spec} specifications generated for the verification classes from RIVAL10 dataset.

```
class airplane
class car
class cat
class dog
class truck

con ears
con floppyears
con eyes
con hairy
con long
con longsnout
con metallic
con rectangular
con tail
con tall
con text
con wheels
con wings

network yolo
rep clip

Module module_0 (yolo,airplane,clip) {
E e1 ::> predict(airplane)

E e2 ::> hasCon(long)
E e3 ::> hasCon(metallic)
E e4 ::> hasCon(tall)
E e5 ::> hasCon(wheels)
E e6 ::> hasCon(wings)

E e9 ::> >(airplane,long,ears)
E e10 ::> >(airplane,long,floppyears)
E e11 ::> >(airplane,long,eyes)
E e12 ::> >(airplane,long,hairy)
E e13 ::> >(airplane,long,longsnout)
E e14 ::> >(airplane,long,rectangular)
E e15 ::> >(airplane,long,tail)
E e16 ::> >(airplane,long,text)
```

```
E e17 ::> >(airplane,metallic,ears)
E e18 ::> >(airplane,metallic,floppyears)
E e19 ::> >(airplane,metallic,eyes)
E e20 ::> >(airplane,metallic,hairy)
E e21 ::> >(airplane,metallic,longsnout)
E e22 ::> >(airplane,metallic,rectangular)
E e23 ::> >(airplane,metallic,tail)
E e24 ::> >(airplane,metallic,text)
E e25 ::> >(airplane,tall,ears)
E e26 ::> >(airplane,tall,floppyears)
E e27 ::> >(airplane,tall,eyes)
E e28 ::> >(airplane,tall,hairy)
E e29 ::> >(airplane,tall,longsnout)
E e30 ::> >(airplane,tall,rectangular)
E e31 ::> >(airplane,tall,tail)
E e32 ::> >(airplane,tall,text)
E e33 ::> >(airplane,wheels,ears)
E e34 ::> >(airplane,wheels,floppyears)
E e35 ::> >(airplane,wheels,eyes)
E e36 ::> >(airplane,wheels,hairy)
E e37 ::> >(airplane,wheels,longsnout)
E e38 ::> >(airplane,wheels,rectangular)
E e39 ::> >(airplane,wheels,tail)
E e40 ::> >(airplane,wheels,text)
E e41 ::> >(airplane,wings,ears)
E e42 ::> >(airplane,wings,floppyears)
E e43 ::> >(airplane,wings,eyes)
E e44 ::> >(airplane,wings,hairy)
E e45 ::> >(airplane,wings,longsnout)
E e46 ::> >(airplane,wings,rectangular)
E e47 ::> >(airplane,wings,tail)
E e48 ::> >(airplane,wings,text)
}

Module module_1 (yolo,car,clip) {
E e1 ::> predict(car)

E e2 ::> hasCon(metallic)
E e3 ::> hasCon(rectangular)
E e4 ::> hasCon(wheels)

E e5 ::> >(car,metallic,ears)
E e6 ::> >(car,metallic,floppyears)
E e7 ::> >(car,metallic,eyes)
E e8 ::> >(car,metallic,hairy)
E e9 ::> >(car,metallic,long)
E e10 ::> >(car,metallic,longsnout)
E e11 ::> >(car,metallic,tail)
E e17 ::> >(car,metallic,tall)
E e18 ::> >(car,metallic,text)
E e19 ::> >(car,metallic,wings)
E e20 ::> >(car,rectangular,ears)
E e21 ::> >(car,rectangular,floppyears)
E e22 ::> >(car,rectangular,eyes)
E e23 ::> >(car,rectangular,hairy)
E e24 ::> >(car,rectangular,long)
E e25 ::> >(car,rectangular,longsnout)
E e26 ::> >(car,rectangular,tail)
E e27 ::> >(car,rectangular,tall)
E e28 ::> >(car,rectangular,text)
E e29 ::> >(car,rectangular,wings)
E e30 ::> >(car,wheels,ears)
```

```
E e31 ::> >(car,wheels,floppyears)
E e32 ::> >(car,wheels,eyes)
E e33 ::> >(car,wheels,hairy)
E e34 ::> >(car,wheels,long)
E e35 ::> >(car,wheels,longsnout)
E e36 ::> >(car,wheels,patterned)
E e37 ::> >(car,wheels,tail)
E e38 ::> >(car,wheels,tall)
E e39 ::> >(car,wheels,text)
E e40 ::> >(car,wheels,wings)
}

Module module_2 (yolo,cat,clip) {
E e1 ::> predict(cat)

E e2 ::> hasCon(ears)
E e3 ::> hasCon(eyes)
E e4 ::> hasCon(hairy)
E e5 ::> >(cat,ears,floppyears)
E e6 ::> >(cat,ears,horns)
E e7 ::> >(cat,ears,long)
E e8 ::> >(cat,ears,longsnout)
E e9 ::> >(cat,ears,metallic)
E e10 ::> >(cat,ears,rectangular)
E e11 ::> >(cat,ears,tail)
E e12 ::> >(cat,ears,tall)
E e13 ::> >(cat,ears,text)
E e14 ::> >(cat,ears,wheels)
E e15 ::> >(cat,ears,wings)
E e16 ::> >(cat,eyes,floppyears)
E e17 ::> >(cat,eyes,long)
E e18 ::> >(cat,eyes,longsnout)
E e19 ::> >(cat,eyes,metallic)
E e20 ::> >(cat,eyes,rectangular)
E e21 ::> >(cat,eyes,tail)
E e22 ::> >(cat,eyes,tall)
E e23 ::> >(cat,eyes,text)
E e24 ::> >(cat,eyes,wheels)
E e25 ::> >(cat,eyes,wings)
E e26 ::> >(cat,hairy,floppyears)
E e27 ::> >(cat,hairy,long)
E e28 ::> >(cat,hairy,longsnout)
E e29 ::> >(cat,hairy,metallic)
E e30 ::> >(cat,hairy,rectangular)
E e31 ::> >(cat,hairy,tail)
E e32 ::> >(cat,hairy,tall)
E e33 ::> >(cat,hairy,text)
E e34 ::> >(cat,hairy,wheels)
E e35 ::> >(cat,hairy,wings)
}

Module module_3 (yolo,dog,clip) {
E e1 ::> predict(dog)

E e2 ::> hasCon(ears)
E e3 ::> hasCon(floppyears)
E e4 ::> hasCon(hairy)
E e5 ::> hasCon(longsnout)

E e6 ::> >(dog,ears,floppyears)
E e7 ::> >(dog,ears,eyes)
E e8 ::> >(dog,ears,long)
```

```
E e9 ::> >(dog,ears,longsnout)
E e10 ::> >(dog,ears,metallic)
E e11 ::> >(dog,ears,rectangular)
E e12 ::> >(dog,ears,tail)
E e13 ::> >(dog,ears,tall)
E e14 ::> >(dog,ears,text)
E e15 ::> >(dog,ears,wheels)
E e16 ::> >(dog,ears,wings)
E e17 ::> >(dog,floppyears,floppyears)
E e18 ::> >(dog,floppyears,eyes)
E e19 ::> >(dog,floppyears,long)
E e20 ::> >(dog,floppyears,longsnout)
E e21 ::> >(dog,floppyears,metallic)
E e22 ::> >(dog,floppyears,rectangular)
E e23 ::> >(dog,floppyears,tail)
E e24 ::> >(dog,floppyears,tall)
E e25 ::> >(dog,floppyears,text)
E e26 ::> >(dog,floppyears,wheels)
E e27 ::> >(dog,floppyears,wings)
E e28 ::> >(dog,hairy,floppyears)
E e29 ::> >(dog,hairy,eyes)
E e30 ::> >(dog,hairy,long)
E e31 ::> >(dog,hairy,longsnout)
E e32 ::> >(dog,hairy,metallic)
E e33 ::> >(dog,hairy,rectangular)
E e34 ::> >(dog,hairy,tail)
E e35 ::> >(dog,hairy,tall)
E e36 ::> >(dog,hairy,text)
E e37 ::> >(dog,hairy,wheels)
E e38 ::> >(dog,hairy,wings)
E e39 ::> >(dog,longsnout,floppyears)
E e40 ::> >(dog,longsnout,eyes)
E e41 ::> >(dog,longsnout,long)
E e42 ::> >(dog,longsnout,longsnout)
E e43 ::> >(dog,longsnout,metallic)
E e44 ::> >(dog,longsnout,rectangular)
E e45 ::> >(dog,longsnout,tail)
E e46 ::> >(dog,longsnout,tall)
E e47 ::> >(dog,longsnout,text)
E e48 ::> >(dog,longsnout,wheels)
E e49 ::> >(dog,longsnout,wings)
}

Module module_4 (yolo,truck,clip) {
E e1 ::> predict(truck)

E e2 ::> hasCon(long)
E e3 ::> hasCon(metallic)
E e4 ::> hasCon(rectangular)
E e5 ::> hasCon(tall)
E e6 ::> hasCon(text)
E e7 ::> hasCon(wheels)

E e8 ::> >(truck,long,ears)
E e9 ::> >(truck,long,floppyears)
E e10 ::> >(truck,long,eyes)
E e11 ::> >(truck,long,hairy)
E e12 ::> >(truck,long,longsnout)
E e13 ::> >(truck,long,patterned)
E e14 ::> >(truck,long,tail)
E e15 ::> >(truck,long,wet)
E e16 ::> >(truck,long,wings)
```

```
E e17 ::> >(truck,metallic,ears)
E e18 ::> >(truck,metallic,floppyears)
E e19 ::> >(truck,metallic,eyes)
E e20 ::> >(truck,metallic,hairy)
E e21 ::> >(truck,metallic,longsnout)
E e22 ::> >(truck,metallic,patterned)
E e23 ::> >(truck,metallic,tail)
E e24 ::> >(truck,metallic,wings)
E e25 ::> >(truck,rectangular,ears)
E e26 ::> >(truck,rectangular,floppyears)
E e27 ::> >(truck,rectangular,eyes)
E e28 ::> >(truck,rectangular,hairy)
E e29 ::> >(truck,rectangular,longsnout)
E e30 ::> >(truck,rectangular,tail)
E e31 ::> >(truck,rectangular,wings)
E e32 ::> >(truck,tall,ears)
E e33 ::> >(truck,tall,floppyears)
E e34 ::> >(truck,tall,eyes)
E e35 ::> >(truck,tall,hairy)
E e36 ::> >(truck,tall,longsnout)
E e37 ::> >(truck,tall,tail)
E e38 ::> >(truck,tall,wings)
E e39 ::> >(truck,text,ears)
E e40 ::> >(truck,text,floppyears)
E e41 ::> >(truck,text,eyes)
E e42 ::> >(truck,text,hairy)
E e43 ::> >(truck,text,longsnout)
E e44 ::> >(truck,text,tail)
E e45 ::> >(truck,text,wings)
E e46 ::> >(truck,wheels,ears)
E e47 ::> >(truck,wheels,floppyears)
E e48 ::> >(truck,wheels,eyes)
E e49 ::> >(truck,wheels,hairy)
E e50 ::> >(truck,wheels,longsnout)
E e51 ::> >(truck,wheels,tail)
E e52 ::> >(truck,wheels,wings)
}
```

Bibliography

[1] en. URL: https://www.tesla.com/support/autopilot.

[2] en. URL: https://textx.github.io/textX/.

[3] Oludare Isaac Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018).

[4] *ASAM OpenXOntology - V1.0.0*. Ingolstadt, Germany, 2023. URL: https://www.asam.net/standards/detail/openxontology/.

[5] Viktor Beloozerov and Alexander Shapkin. "Indices formal grammar of the Universal Decimal Classification". In: *Bibliosphere* (2018). URL: https://api.semanticscholar.org/CorpusID:239099263.

[6] Ksenia Bestuzheva et al. "Enabling Research through the SCIP Optimization Suite 8.0". In: *ACM Trans. Math. Softw.* 49.2 (2023). ISSN: 0098-3500. DOI: 10.1145/3585516. URL: https://doi.org/10.1145/3585516.

[7] Puja Bharati and Ankita Pramanik. "Deep learning techniques—R-CNN to mask R-CNN: a survey". In: *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2019* (2020), pp. 657–668.

[8] Akhilan Boopathy et al. *CNN-Cert: An Efficient Framework for Certifying Robustness of Convolutional Neural Networks*. 2018. arXiv: 1811.12395 [stat.ML]. URL: https://arxiv.org/abs/1811.12395.

[9] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. arXiv: 2005.12872 [cs.CV]. URL: https://arxiv.org/abs/2005.12872.

[10] Chih-Hong Cheng, Tobias Schuster, and Simon Burton. "Logically sound arguments for the effectiveness of ml safety measures". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2022, pp. 343–350.

[11] Apollo Developers. *Apollo*. Accessed: 2024-06-23. 2023. URL: https://github.com/ApolloDevelopers/apollo.

[12] Tommaso Dreossi et al. *VERIFAI: A Toolkit for the Design and Analysis of Artificial Intelligence-Based Systems*. 2019. arXiv: 1902.04245 [cs.AI]. URL: https://arxiv.org/abs/1902.04245.

[13] Lixuan Du, Rongyu Zhang, and Xiaotian Wang. "Overview of two-stage object detection algorithms". In: *Journal of Physics: Conference Series* 1544 (May 2020), p. 012033. DOI: 10.1088/1742-6596/1544/1/012033.

[14] Yrvann Emzivat et al. "A Formal Approach for the Design of a Dependable Perception System for Autonomous Vehicles". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2452–2459. DOI: 10.1109/ITSC.2018.8569903.

[15] Jannis Erz et al. “Towards an Ontology That Reconciles the Operational Design Domain, Scenario-based Testing, and Automated Vehicle Architectures”. In: *2022 IEEE International Systems Conference (SysCon)*. 2022, pp. 1–8. DOI: 10.1109/SysCon53536.2022.9773840.

[16] Asif Faisal et al. “Understanding autonomous vehicles”. In: *Journal of transport and land use* 12.1 (2019), pp. 45–72.

[17] David Fernández Llorca and Emilia Gómez. “Trustworthy autonomous vehicles”. In: *Publications Office of the European Union, Luxembourg,, EUR* 30942 (2021).

[18] Julien Girard-Satabin et al. *CAMUS: A Framework to Build Formal Specifications for Deep Perception Systems Using Simulators*. 2019. arXiv: 1911.10735 [cs.LG]. URL: https://arxiv.org/abs/1911.10735.

[19] Maximilian Grabowski and Ya Wang. “A CONCEPT TO SUPPORT AI MODELS BY USING ONTOLOGIES-PRESENTED ON THE BASIS OF GER-MAN TECHNICAL SPECIFICATIONS FOR LANE MARKINGS”. In: *27th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration*. 23-0265. 2023.

[20] Xingwu Guo et al. *OccRob: Efficient SMT-Based Occlusion Robustness Verification of Deep Neural Networks*. 2023. arXiv: 2301.11912 [cs.LG]. URL: https://arxiv.org/abs/2301.11912.

[21] Madhuri Hammad. *Modularity and its properties*. 2023. URL: https://www.geeksforgeeks.org/modularity-and-its-properties/.

[22] Mohammad Hekmatnejad et al. *Formalizing and Evaluating Requirements of Perception Systems for Automated Vehicles using Spatio-Temporal Perception Logic*. 2023. arXiv: 2206.14372 [cs.RO]. URL: https://arxiv.org/abs/2206.14372.

[23] Jeremy Jordan. *An overview of object detection: one-stage methods*. en. Mar. 2023. URL: https://www.jeremyjordan.me/object-detection-one-stage/.

[24] ZuWhan Kim and Ramakant Nevatia. “Uncertain reasoning and learning for feature grouping”. In: *Computer Vision and Image Understanding* 76.3 (1999), pp. 278–288.

[25] Panagiotis Kouvaros and Alessio Lomuscio. *Formal Verification of CNN-based Perception Systems*. 2018. arXiv: 1811.11373 [cs.LG]. URL: https://arxiv.org/abs/1811.11373.

[26] Moez Krichen et al. “Are Formal Methods Applicable To Machine Learning And Artificial Intelligence?” In: *2022 2nd International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*. 2022, pp. 48–53. DOI: 10.1109/SMARTTECH54121.2022.00025.

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[28] Martin Leucker. “Formal Verification of Neural Networks?” In: *Formal Methods: Foundations and Applications*. Ed. by Gustavo Carvalho and Volker Stolz. Cham: Springer International Publishing, 2020, pp. 3–7.

[29] Wei Li et al. “Incremental Learning of Single-stage Detectors with Mining Memory Neurons”. In: *2018 IEEE 4th International Conference on Computer and Communications (ICCC)* (2018), pp. 1981–1985. URL: https://api.semanticscholar.org/CorpusID:199440787.

[30] Guannan Lou et al. *Testing of Autonomous Driving Systems: Where Are We and Where Should We Go?* 2022. arXiv: 2106.12233 [cs.SE]. URL: https://arxiv.org/abs/2106.12233.

[31] Guannan Lou et al. *Testing of Autonomous Driving Systems: Where Are We and Where Should We Go?* 2022. arXiv: 2106.12233 [cs.SE]. URL: https://arxiv.org/abs/2106.12233.

[32] Yining Ma et al. "Verification and validation methods for decision-making and planning of automated vehicles: A review". In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 480–498.

[33] Ravi Mangal et al. *Concept-based Analysis of Neural Networks via Vision-Language Models.* 2024. arXiv: 2403.19837 [cs.LG]. URL: https://arxiv.org/abs/2403.19837.

[34] Mercedes. *Mercedes Drive Pilot.* en. 2024. URL: https://www.mercedes-benz.de/passengercars/technology/drive-pilot.html.

[35] Mazda Moayeri et al. "A Comprehensive Study of Image Classification Model Sensitivity to Foregrounds, Backgrounds, and Visual Attributes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2022.

[36] Mazda Moayeri et al. *Text-To-Concept (and Back) via Cross-Model Alignment.* 2023. arXiv: 2305.06386 [cs.CV]. URL: https://arxiv.org/abs/2305.06386.

[37] Mark A. Musen. "The protégé project: a look back and a look forward". In: *AI Matters* 1.4 (2015), 4–12. DOI: 10.1145/2757001.2757003. URL: https://doi.org/10.1145/2757001.2757003.

[38] OWLready2. *Owlready2's documentation — Owlready2 0.46 documentation.* en. URL: https://owlready2.readthedocs.io/en/latest/.

[39] Jan-Pieter Paardekooper et al. "A Hybrid-AI Approach for Competence Assessment of Automated Driving functions." In: *SafeAI@ AAAI.* 2021.

[40] Darsh Parekh et al. "A review on autonomous vehicles: Progress, methods and challenges". In: *Electronics* 11.14 (2022), p. 2162.

[41] *Road vehicles – Functional safety – Part 1: Vocabulary.* Geneva, Switzerland, 2018. URL: https://www.iso.org/standard/68383.html.

[42] *Road vehicles – Safety of the intended functionality.* Geneva, Switzerland, 2022. URL: https://www.iso.org/standard/70939.html.

[43] Jan Roßbach and Michael Leuschel. "Certified Control for Train Sign Classification". In: *Electronic Proceedings in Theoretical Computer Science* 395 (Nov. 2023), 69–76. ISSN: 2075-2180. DOI: 10.4204/eptcs.395.5. URL: http://dx.doi.org/10.4204/EPTCS.395.5.

[44] Gesina Schwalbe et al. "Structuring the safety argumentation for deep neural network based perception in automotive applications". In: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops: DECSoS 2020, DepDevOps 2020, USDAI 2020, and WAISE 2020, Lisbon, Portugal, September 15, 2020, Proceedings 39.* Springer. 2020, pp. 383–394.

[45] Baeldung on Computer Science and Baeldung on Computer Science. *Single Shot Detectors (SSDs) — Baeldung on Computer Science.* en-US. 2023. URL: https://www.baeldung.com/cs/ssd.

[46] Sanjit A. Seshia et al. "Formal Specification for Deep Neural Networks". In: *Automated Technology for Verification and Analysis.* Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2018, pp. 20–34.

[47] Tahira Shehzadi et al. *Object Detection with Transformers: A Review.* 2023. arXiv: 2306.04670 [cs.CV]. URL: https://arxiv.org/abs/2306.04670.

[48] Gagandeep Singh et al. "Fast and Effective Robustness Certification". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf.

[49] Amina Souag. "Towards a new generation of security requirements definition methodology using ontologies". In: *International Conference on Advanced Information Systems Engineering*. 2012. URL: https://api.semanticscholar.org/CorpusID:5422670.

[50] Johannes Stallkamp et al. "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". In: *IEEE International Joint Conference on Neural Networks*. 2011, pp. 1453–1460.

[51] Chen Sun et al. "Toward ensuring safety for autonomous driving perception: standardization progress, research advances, and perspectives". In: *IEEE Transactions on Intelligent Transportation Systems* (2023).

[52] Semafora Systems. *OntoStudio*. Accessed: 2024-06-23. 2023. URL: https://www.semafora-systems.com/en/products/ontostudio/.

[53] Kento Tanaka et al. "A Formal Specification Language Based on Positional Relationship Between Objects in Automated Driving Systems". In: *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2022, pp. 950–955. DOI: 10.1109/COMPSAC54236.2022.00147.

[54] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. SAE Standard J3016_202104. 2021. DOI: 10.4271/J3016_202104. URL: https://doi.org/10.4271/J3016_202104.

[55] Kalaiselvan Thangaraj. *GitHub - kalaiselvan-t/automatic-verification*. en. July 2024. URL: https://github.com/kalaiselvan-t/automatic-verification.

[56] Inc. TopQuadrant. *TopBraid Composer*. Accessed: 2024-06-23. 2023. URL: https://www.topquadrant.com/products/topbraid-composer/.

[57] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[58] Vidit Vidit and Mathieu Salzmann. *Attention-based Domain Adaptation for Single Stage Detectors*. 2021. arXiv: 2106.07283 [cs.CV]. URL: https://arxiv.org/abs/2106.07283.

[59] Ao Wang et al. "YOLOv10: Real-Time End-to-End Object Detection". In: *arXiv preprint arXiv:2405.14458* (2024).

[60] Shiqi Wang et al. *Formal Security Analysis of Neural Networks using Symbolic Intervals*. 2018. arXiv: 1804.10829 [cs.AI]. URL: https://arxiv.org/abs/1804.10829.

[61] waymo. *waymo*. en. 2024. URL: https://waymo.com/blog/.

[62] Katrin Weller. "2.1 Ontologies: Definition and Background". In: 2010. URL: https://api.semanticscholar.org/CorpusID:63911178.

[63] Yanzhao Xia and Shaoying Liu. "A Framework of Formal Specification-Based Data Generation for Deep Neural Networks". In: *Proceedings of the 2023 12th International Conference on Software and Computer Applications*. ICSCA '23. Kuantan, Malaysia: Association for Computing Machinery, 2023, 273–282. ISBN: 9781450398589. DOI: 10.1145/3587828.3587869. URL: https://doi.org/10.1145/3587828.3587869.

[64] Weiming Xiang et al. *Verification for Machine Learning, Autonomy, and Neural Networks Survey*. 2018. arXiv: 1810.01989 [cs.AI]. URL: https://arxiv.org/abs/1810.01989.

[65] Jin Zhang and Jingyue Li. "Testing and verification of neural-network-based safety-critical control software: A systematic literature review". In: *Information and Software Technology* 123 (2020), p. 106296. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2020.106296. URL: http://dx.doi.org/10.1016/j.infsof.2020.106296.

[66] Máté Zöldy, Zsolt Szalay, and Viktor Tihanyi. "Challenges in homologation process of vehicles with artificial intelligence". In: *Transport* 35.4 (2020), pp. 447–453.

List of Figures

List of Tables