Converting OntoUML Diagram Images into Machine-Processable Formats to Enhance the Usability of Models

Simeon Kaishev, University of Twente, The Netherlands

ABSTRACT

OntoUML is an important ontology language, however, most of the diagrams written in this language exist only as images within published papers, rendering them impractical for research purposes. Despite existing research on converting UML diagram images to machine-processable formats, no studies address the conversion of OntoUML images. In this paper, I present the OntoUML Image Taxonomy Extractor (OITE), which detects OntoUML diagrams using transfer learning. OITE employs image processing techniques to translate OntoUML diagram images into the OntoUML Vocabulary language. The system involves class recognition through rectangle detection and OCR techniques to extract class elements, followed by line recognition and relationship type recognition to determine class relationships. Additionally, an experiment using ChatGPT was conducted to explore the potential of using visual large language models for this task. The results are used to demonstrate the feasibility of using LLMs for simple diagram translations, compare the performance of OITE and ChatGPT, and highlight areas for further research in ontology-driven conceptual modeling.

Keywords: OntoUML · Ontology-Driven Conceptual Modeling · Image Processing · Transfer Learning · Large Language Model

1 INTRODUCTION

1.1 Context

OntoUML is a conceptual modeling language that extends and refines the Unified Modeling Language (UML) to represent the ontological commitments of a domain. Unlike conventional UML models, OntoUML captures not only the structural aspects of systems but also their underlying ontological meta-properties, making it particularly suitable for modeling complex systems where importance is placed on clarity of semantics and on rigorous analysis. OntoUML finds applications in various domains, such as ontology engineering, conceptual modeling, knowledge representation, and ontology-driven software engineering, facilitating a deeper understanding of domain semantics and supporting the development of semantically rich information systems [9].

The language's origins can be traced back to 2005, when Guizzardi evaluated and re-designed a fragment of the UML 2.0 metamodel to reflect the ontological micro theories of the Unified Foundational Ontology (UFO) as a part of his Ph.D. thesis [8]. Since its inception, through further research and development efforts, OntoUML has been expanded and refined, and it has become one of the most used ontology-driven conceptual modeling languages, according to Verdonck and Gailly [21]. Additionally, an experiment by Verdonck et al. found it to significantly improve the quality of conceptual models without requiring additional effort when compared to a classical conceptual modeling language [22].

1.2 Problem Statement

As OntoUML has grown in popularity, more researchers in the conceptual modeling sphere have shown interest in working with the language. Barcelos et al. have compiled a structured and open-source catalog that contains OntoUML and UFO ontology models, designed to support ontology-driven conceptual modeling research [1]. While this catalog provides high-quality curated, structured, and machine-processable data, it only includes 168 models at the time of writing. While that may be enough for some researchers, in the fields of Artificial Intelligence, for example, more data is required, which is currently only available in image files of already published papers. Currently, these diagrams can be used after they are manually re-created, which takes time and effort from the researcher. In addition, this lack of usable data also slows down the progress of OntoUML, as it hinders the development of new algorithms and methods, such as specialized Machine Learning Models, that could advance the field of conceptual modeling.

In this research, I aim to help overcome this lack of data by exploring machine learning and image processing techniques to convert OntoUML diagram images into machine-processable formats, thus increasing their usability. To do this, I have established the following research question:

To what extent can OntoUML diagram images be translated to machine-processable formats, using machine learning and image processing techniques, to improve their usability in the field of ontology?

To help answer this question, I have compiled the following sub-questions:

RQ1: To what extent can computer vision techniques effectively detect OntoUML diagram images?

RQ2: What methodologies are suitable for converting OntoUML diagram images into machine-processable formats through a combination of image processing techniques and machine learning algorithms?

RQ3: To what extent is the utilization of a visual large language model viable for translating OntoUML images into machine-processable formats?

The following sections cover methodologies for classifying and converting OntoUML diagram images, including related works, the OntoUML Image Taxonomy Extractor (OITE) for classification and conversion, performance comparison with ChatGPT for conversion, and future research directions.

2 RELATED WORKS

Having diagrams saved in non-processable formats is not exclusive to the field of ontology, therefore there are a number of papers on classifying and converting standard UML diagrams. Through literature research, I could not identify any published work on converting OntoUML images into machine-processable formats. Because of the similarity between OntoUML and UML class diagrams, many of the techniques discussed in UML models apply to OntoUML as well. This section will cover the relevant ones.

2.1 Classification

While OntoUML diagram images can be extracted from papers with specialized PDF tools, they are likely not the only images in the paper. The following papers discuss classification of UML diagrams, and can be used for developing a system which recognizes OntoUML diagrams, so researchers do not have to manually filter them.

A first research that describes a system that could be reused in this research is that of Gosala et al. [7]. They used a Convolutional Neural Network (CNN) to detect class diagrams by classifying images into either UML class diagrams or not. After applying regularization techniques to improve performance, their model achieved an accuracy of 86.63%, which the authors claim is because of the small size of their training set.

Due to the relatively small number of machine-processable labeled OntoUML diagrams, it is favorable to explore techniques that do not require a lot of training data. Tavares et al. explore the use of transfer learning and data augmentation to classify six types of UML diagrams using a relatively small dataset of 200 images per diagram type [20]. In this context, transfer learning involves leveraging a model, pre-trained on non-technical images, to classify UML diagrams, adapting its learned features to the new task, thereby reducing the need for a large dataset. In their research, they used three widely known CNN architectures as their pre-trained models and found that transfer learning contributes to achieving good results even when using scarce data.

A different approach is employed by Ho-Quang et al., where they propose 23 image-features for classifying UML class diagram images [10]. They found that 19 of those features can be considered as influential predictors for classifying UML class diagram images. Additionally, they explored six classification algorithms and through them the prediction rate achieves nearly 96% correctness for UML class diagrams.

To address **RQ1**, transfer learning will be used to fine-tune an existing classifier to recognize OntoUML diagrams, as fewer images are needed to refine a model compared to training one. This allows for the use of the catalog compiled by Barcelos et al. [1] combined with data augmentation and thus reduces the requirement for extensive data collection.

2.2 Conversion

After the image has been identified to be of an OntoUML diagram, the next step is to convert it to a machine-processable format. The following papers explore possible approaches.

The work of Karasneh and Chaudron proposes a system used for extracting UML class models from images [13]. Their image processing consists of four consecutive steps: First, the classes are extracted through rectangle detection. Then, the attributes of each class are identified using optical character recognition (OCR). The third step involves detecting relationships between the classes using various image processing techniques. Finally, the different kinds of relationships are identified through shape recognition algorithms. Once these steps are completed a XML file is created with the gathered information. This method achieved high accuracy in recognizing classes and relations (with the limitation of not being able to detect polygonal relations), but sometimes it struggled to recognize symbols which denote the type of the relation.

This problem is addressed in ReSECDI (**Re**cognition of **S**emantic **E**lements in **C**lass **D**iagram **I**mages) [4], the system for automatically recognizing the semantic elements from UML class diagram images proposed by Chen et al. [3]. They introduce a double-recognition-approximation method to recognize relationship types in their two-part approach. In addition, their method introduces rectangle clustering to better recognize classes and their elements, and polygonal line merging to allow it to detect all relationships between classes. ReSECDI was tested using 30 images drawn by three popular UML tools and 50 diagrams collected from the open-source communities, and achieved promising performances [3].

In addressing **RQ2**, the approach proposed by Chen et al. [3] will serve as the foundation for this research. However, adaptations are necessary due to disparities between OntoUML and UML class diagrams. Notably, classes in OntoUML incorporate stereotypes within their name field, requiring further processing after the OCR text extraction. Furthermore, the process of relationship detection will have to be altered as certain relationships identified in class diagrams lack semantic significance in OntoUML. Lastly, while the system outlined by Chen et al. [3] stores extracted information in a semantic design model, the present study will store it in OntoUML Vocabulary, an implementation of the OntoUML Metamodel in the Web Ontology Language (OWL) which supports the serialization, exchange, and publishing of OntoUML models as linked data [17].

A different novel approach is proposed by Conrardy & Cabot [6]. They make use of popular Visual Large Language Models to generate the formal representation of (UML) models from a given pictures of whiteboard drawings. The best results were achieved by Chat GPT-4V [14], however while they find them encouraging, they conclude that the system still has limitations, which require a human in the loop to overcome. To address **RQ3**, Chat GPT-4V will be used to generate OntoUML vocabulary specifications from OntoUML diagram images of increasing complexity. The viability and accuracy of this approach will be assessed by analyzing the generated specifications for any mistakes or inconsistencies.

3 CLASSIFICATION

The following two sections describe the development of the **O**ntoUML **I**mage **T**axonomy **E**xtractor (OITE). This system is designed to classify and convert OntoUML diagram images into machine-readable formats. The code for OITE, along with the datasets used for training, is available on GitHub [11]. This section outlines how OITE detects whether an image contains an OntoUML diagram, covering the dataset compilation (Section 3.1), the model used and its training process (Section 3.2), the system's performance (Section 3.3), and proposed improvements and different approaches (Section 3.4).

3.1 Gathering a dataset

To ensure the best performance of the system, it was crucial to ensure the dataset represents the set of images found in papers containing OntoUML diagrams. Three classes of images were created for this purpose: OntoUML, UML Class diagram and non-UML images. This selection was made to ensure that the system can accurately detect the subtle differences between OntoUML and UML Class Diagrams while separating all other diagram types. To gather OntoUML images, I used the OntoUML/UFO Catalog [1]. Since the model contains both UFO and OntoUML models, I wrote a python script which extracts the images from the original-diagrams folder of each model that has OntoUML as representation style in its *metadata.yaml* file. This resulted in 643 OntoUML diagrams. For the other two types of image, I made use of the dataset of the Paper "Multiclass Classification of Four Types of UML Diagrams from Images Using Deep Learning" [18], as it contained 700 Class diagrams and 700 non-UML images. During the development, two datasets were compiled using the aforementioned data.

Half of the first dataset is composed of the 643 OntoUML images, and the other half comprised 321 randomly selected class diagrams and 322 randomly selected non-UML images from the dataset by Shcherban et al. [18]. The idea behind it was to have equal parts OntoUML and non-OntoUML images. Further in the text it will be referred to as the "binary dataset".

The second dataset consists of the 643 OntoUML images, 643 randomly picked class diagrams and 643 randomly picked non-UML images, taken from the same sources as the binary dataset. In further sections this dataset will be referred to as the 3-class dataset. Data augmentation in the form of horizontal image mirroring was applied to both datasets in order to increase their size by 100%.

3.2 Model Training

As discussed in Section 2.1, I elected to use transfer learning as limited training data is available. InceptionV3 [19] was chosen as the base model to be fine-tuned, as Tavares et al. found it to have the best performance when classifying UML diagrams between the models they tested [20]. Following their example the first 175 layers have their weights frozen (meaning that they will not be modified during training), while the remaining layers are left to have their weights adjusted using the dataset. The Keras framework [5] was used to obtain the InceptionV3 model, which was then fine-tuned into two OntoUML classification models using both of the compiled datasets. Both times the dataset was split in the following manner: 20% of the model is taken as the test set, afterwards 20% of the remaining data is reserved for validation during training, with the remainder being used to train the model.

The first model was trained using the binary dataset, and will therefore be referred to as the binary classifier. A *GlobalAveragePooling2D* layer is added to the InceptionV3 model, along with a single neuron dense layer which uses the sigmoid activation function. This differs from the final layer used by Tavares et al. [20], as this work's model will be performing binary classification (OntoUML or not) compared to their six class classifier. For the same reason the model is trained using the "binary crossentropy" function. The model is trained for 10 epochs, employing early stopping, meaning that training will be concluded early if performance on the chosen metric (in this case loss on the validation set) stops improving during training.

The second model was trained using the 3-class dataset and will therefore be referred to as the 3-class classifier. Similar to the binary classifier, a *GlobalAveragePooling2D* layer is added, however the final layer is a three neuron dense layer that uses the *softmax* activation function. This model is also trained for 10 epochs, employing early stopping, however the "sparse categorical crossentropy" loss function is used to account for the fact that the model is working with more than two classes.

3.3 Results

For the binary classifier the best performance on the validation set was achieved during the 10th epoch. Therefore, the model saved at this checkpoint was selected and used for subsequent steps. It achieved an accuracy of 83% on the test set. The confusion matrix, presented in ***Figure 1***, illustrates the model's performance across the different classes.

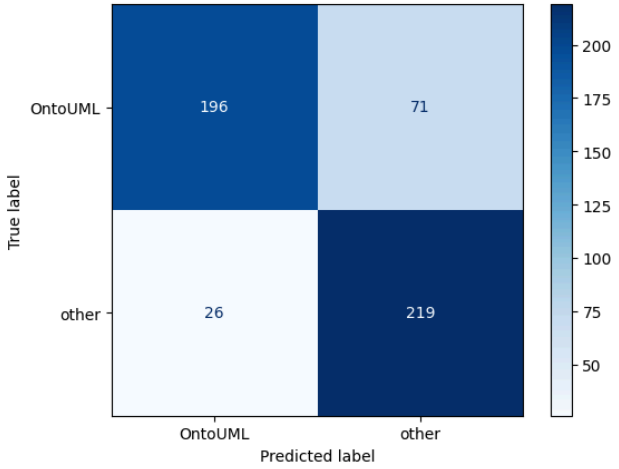

Figure 1 *Binary classifier confusion matrix*

Figure 1 shows that 71 OntoUML images were classified as other, and 26 of the "other" images were classified as OntoUML. After going over the wrongly classified "other" images, I found that 18 of the 26 were class diagrams and one closely resembled a sequence diagram. This can be attributed to the close resemblance between OntoUML and UML, making their distinction difficult for the binary classifier.

The suboptimal performance of the binary classifier led to the implementation of the 3-Class classifier. After 10 epochs of training, it could achieve 91% accuracy on the test set.

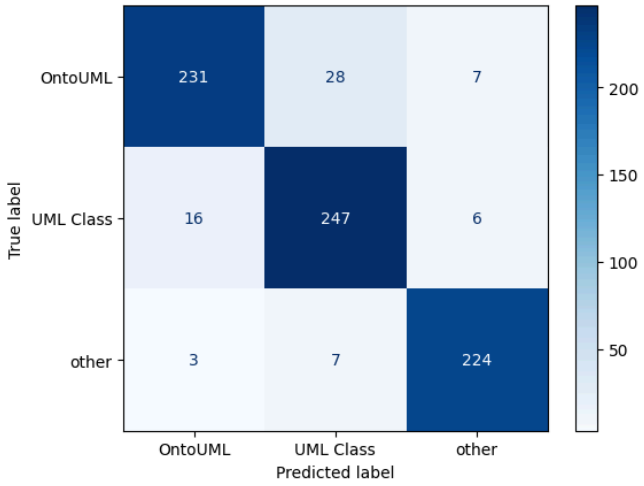

Figure 2 *3-class classifier confusion matrix*

The confusion matrix of this model can be found in ***Figure 2***. It can be inferred that most of the wrong classifications are once again between class and OntoUML diagrams because of their similarity. This is not considered a significant issue, as the conversion process incorporates a step that further ensures the image is of an OntoUML diagram, as discussed in Section 4.2.3.

3.4 Future Research

While the 3-class classifier achieved good performance, improvements can always be found both in measured accuracy and real-world performance. I elected to combine pre-existing datasets for training due to time limitations. This means that the "other" category consists of not non-UML diagrams, but also various pictures not related to modeling. It would be beneficial for the model's real-world performance if a dataset was compiled from images gathered from published papers that utilize OntoUML diagrams. This would ensure a more concise representation of the "other" images that can be found in these papers, which could allow the model to better classify them. Additionally, it would be interesting to explore the use of other pre-trained models for transfer learning. Finally, unsupervised learning techniques, such as autoencoders, may offer a viable approach for identifying the distinctive characteristics found in OntoUML diagrams and identifying instances where these features are absent.

4. CONVERSION

This section delves into the part of OITE designed for translating OntoUML images into machine-processable formats. This paper will primarily explore the extraction of syntactical information such as classes and generalizations from the models, providing a foundation for further research in the OntoUML domain. Subsection 4.1 covers the preprocessing of the images, which differs significantly to the one used for classification. Subsections 4.2 and 4.3 detail the process of class recognition and relationship recognition, respectively, and Subsection 4.4 discusses how the extracted data is converted into OntoUML Vocabulary. Finally, Subsection 4.5 discusses how this part of OITE was tested and 4.6 discusses the possible improvements that can be made to it.

4.1 Preprocessing

The preprocessing stage, based on the methodology detailed by Chen et al [3], is designed to prepare images for subsequent class and relationship recognition. Unlike the preprocessing used for classification, the resolution of the images is kept the same to maintain maximum detail. Instead, images are converted to a single channel (grayscale) format and Gaussian sharpening is applied to enhance edges, thereby improving recognition in low-quality images. Chen et al. employ the morphology operations of erosion and dilation to reduce noise in the images and improve edge detection. However, testing revealed that these methods did not enhance edge detection, and significantly degraded the quality of text within the images. Consequently, this step was omitted from the preprocessing procedure. Finally, a review of the code supplied with the paper by Chen et al. revealed a thresholding step, where all pixels with values below 150 are turned white and those above 150 are turned black, resulting in a binary image which could enhance the detection of edges. Testing indicated that thresholding improved edge detection only in images created using specific software, while in other cases, the sharpened grayscale image performed better. Therefore, in the final system, the shape recognition process described in the following sections is applied to both the sharpened grayscale image and the binary one. The results from both approaches are then compared, and the one yielding better results (more recognitions) is selected.

4.2 Class Recognition

For class recognition, the approach was once again based on that proposed by Chen et al. [3]. The following subsections will cover the three main parts of class recognition and their limitations.

4.2.1 Rectangle Recognition

In OntoUML, classes are depicted as rectangles, therefore I adopted a rectangle recognition algorithm to identify all such shapes within an image. The 'cv2.findContours' function is used to extract the outlines of all objects in the image. Afterwards, the 'approxPolyDP' function is used to approximate each contour shape into a polygon with fewer vertices.

To isolate all rectangles, the algorithm selects approximated shapes with four vertices, ensuring that their area is greater than 60 pixels but less than half of the image area. To further validate these shapes as rectangles, pairs of opposite sides and their diagonals are compared to ensure they are within a specific threshold of each other. Additionally, the angles between all sides are verified to be approximately 90 degrees.

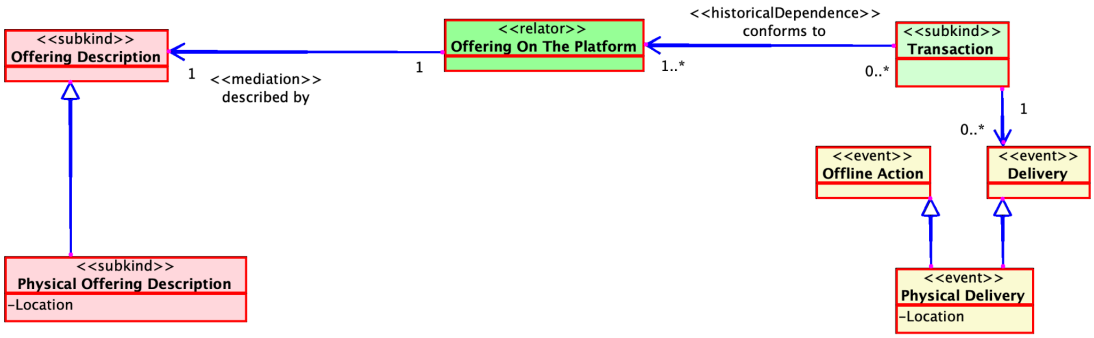

Figure 3 *Classes and relationships detected in OntoUML diagram image*

This meticulous process enables OITE to accurately recognize all class rectangles, as illustrated in ***Figure 3***, which displays the detected rectangles outlined in red in OntoUML diagram image.

4.2.2 Rectangle Clustering

Since a class can be represented by up to four rectangles, the next step involves grouping together all rectangles that belong to the same class. This is achieved by clustering rectangles whose horizontal distances between their top-left corners are within 3-5 pixels and whose vertical distances, accounting for their height, are also within 3-5 pixels. This approach also helps filter out the rectangle, outlining the entire class.

Once the rectangles are grouped, they are sorted based on their positions into top, middle, and potentially bottom sections. The top section contains the class name and stereotype, the middle section contains the fields, and the bottom section contains the functions, if available. Finally, a new, more accurate outline of the entire class is created by taking the top-left point and width of the top rectangle and the combined heights of all the rectangles.

4.2.3 Text Recognition

Once all the rectangles containing the classes are clustered, their top sections are cropped out and resized to three times their original size to improve text recognition performance. Subsequently, optical character recognition (OCR) is applied using the Python Tesseract library to extract the textual information, including the class name, stereotype, and potentially the package. Regular expressions are then employed to detect and extract the stereotype and class name, while the package name is removed.

To validate that the model being translated conforms to OntoUML standards, the Levenshtein distance (also known as 'edit distance') [2] of each identified stereotype is computed with respect to all OntoUML class stereotypes. If any of the distances to OntoUML class stereotypes are less than or equal to 2, the respective stereotype is deemed to be one of the OntoUML standard stereotypes. This serves as confirmation that the model adheres to the OntoUML specification.

Furthermore, to maintain consistency with evolving OntoUML standards, a record of old OntoUML stereotypes is retained. If any of these outdated stereotypes are encountered during the translation process, they are to be replaced with their corresponding updated counterparts. This ensures the alignment of the translated model with the latest OntoUML specifications.

4.2.4 Limitations

After testing the system, it has become evident that certain limitations exist within it. Notably, when text within a rectangle is positioned too close to the edge or when there are interruptions in the continuous line defining the rectangle, the recognition process encounters obstacles, leading to unrecognized parts of classes. Another notable limitation arises when the text within a class is not in English; the OCR attempts to transcribe it as English, resulting in suboptimal outcomes. These limitations underline the need for further refinement and optimization of the system, however, addressing these issues is out of this work's scope and is suggested as a future work.

4.3 Relationship Recognition

When it comes to relationship recognition, the method outlined by Chen et al. [3] was initially implemented, however, it consistently failed to detect all relationships during testing. As a result, an alternative approach was devised and implemented. While this solution demonstrated better performance in testing, it also introduced additional limitations. The implementation specifics and constraints will be discussed in the following subsections.

4.3.1 Relationship Detection

The first step in relationship recognition consists of detecting the relationships and the classes they connect. First, contour detection is applied on an image from which the class rectangles have been removed, leaving only the relationship lines visible. Afterwards, the contours are approximated to polygons, thereby generating a set of points in the image that could be utilized to redraw the relationships. Finally, each point within a relationship is examined to assess its proximity to a class rectangle. Those points found to be in close proximity are designated as endpoints, and saved paired with the corresponding class to which they were connected. As the focus of this paper lies solely on generalization relationships, all relationships that connect fewer than two classes can be discarded. This exclusion criterion aligns with UML standards, as a class cannot generalize itself. Furthermore, this approach helps in filtering out falsely detected relationships stemming from other elements in the model image, such as text within the diagrams or cardinalities. An example illustrating detected relationships (highlighted in blue) is depicted in ***Figure 3***.

4.3.2 Relationship Type Recognition

Once relationships and the classes they connect are detected, the next step is to determine the type of each relationship. To achieve

this, an algorithm was developed to crop the rectangular area where a relationship connects to a class, allowing for the analysis of the relationship sign. As a result of a review of the images in the OntoUML Catalog, two main factors were found to determine the size of the relationship sign: the size of the image (higher resolution images use more pixels to depict the sign), and the number of classes (diagrams with more content tend to have smaller relationship signs). Based on this information, the following approach was adopted: the diagonal length of the image is calculated, scaled, and adjusted inversely proportional to the square root of the number of classes. This result is used as the diagonal of the rectangle cropped at the connection points of each relationship to a class. These rectangles are then cropped out from the image of the diagram with removed class diagrams for analysis.

As the primary objective is to determine whether the relationship is a generalization, a triangle recognition algorithm was initially used to analyze the resulting cropped images. This method, however, had several limitations. Generalization signs sometimes had their tips removed during the process of isolating classes, leading to missed recognitions. Additionally, noise in the cropped area was occasionally identified as a rectangle, resulting in false positives.

To enhance performance, an alternative approach was adopted. During the testing of the other parts of the system, the cropping algorithm was used to compile a dataset comprising 60 images for each of three relationship categories: Generalization, Arrow, and Tail. The Generalization category includes images of triangles at the end of a generalization relationship, the Arrow category contains images of association arrows, and the Tail category consists of images of the tail end of a relationship (just a line connecting to the class). The images were resized to 100x100 pixels and augmented using the following techniques: each image was rotated by 90, 180, and 270 degrees to account for relationships connecting to classes on different sides, and the resulting set of four images was horizontally mirrored. This augmentation increased the dataset from 60 to 240 images per relationship category. A version of the 3-Class Classifier used in OntoUML detection, modified to handle 100x100 images, was then trained on this dataset. The resulting model is utilized by OITE to classify the ends of detected relationships, enabling the determination of their type and direction.

4.3.3 Limitations

An important limitation of this approach is its inability to handle cases where the line is interrupted by elements such relationship labels. However, testing using the OntoUML catalog indicated that it is rare for generalizations to include labels and, consequently, the system performed well in most instances. Additionally, due to a lack of images for aggregation and composition relationships, the model wasn't trained on these types, meaning they could be misclassified as generalizations, which would lead to errors in translation. Retraining the relationship detection model with a comprehensive dataset, including all relationship types, would enhance OITE's ability to translate more complicated diagrams.

4.4 Conversion to OntoUML Vocabulary

Finally, all the data gathered from class and relationship recognition is used to reconstruct the model in OntoUML Vocabulary. This is done by translating the detected classes and relationships into Resource Description Framework (RDF) triples using RDFLib, a Python library for working with RDF data [16]. RDF is a standard model for data interchange on the web, which encodes information in triples of subject, predicate, and object [23]. These triples capture the structure and connections of OntoUML models according to the OntoUML Vocabulary standardization. The resulting RDF graph provides a machine-readable format of the model, which can be saved as a Turtle file for easy integration into various applications and tools. This process ensures that the reconstructed model is accessible and usable for further analysis and utilization in ontology research.

4.5 Testing

For testing class detection and relationship recognition 10 random diagrams were selected from the OntoUML catalog and both class recognition and relationship detection were applied. Overall 83/92 classes were detected and 71/82 relationships were detected. The types of mistakes were detailed in Sections 4.2.4 and 4.3.3, and a table of the diagrams used for testing and the number of mistakes for each diagram can be found on the Git repository [11].

To test the model used for relationship type recognition, 20% of the dataset used for training were reserved for testing. The model achieved 88.5% accuracy on this test set.

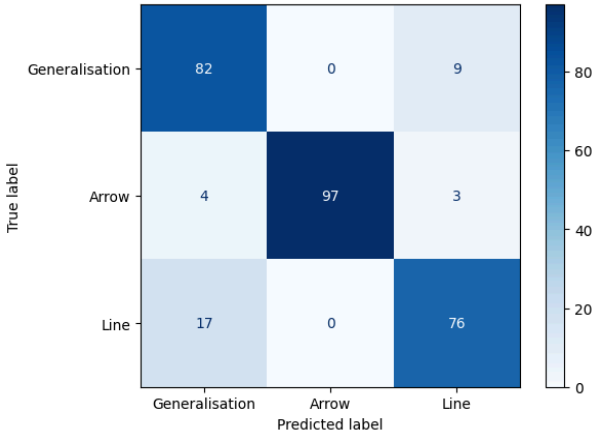

Figure 4 *Relationship Type Classifier confusion matrix*

The confusion matrix shown in ***Figure 4*** highlights that the most common error is that Line (Tail) is occasionally misclassified as Generalization. Examination of misclassified instances revealed that this was due to the presence of parts of generalization arrows within the cropped images. To address this issue, the scaling factor for determining the rectangle diameter prior to cropping can be adjusted from $1/5^{th}$ of the image diagonal to $1/10^{th}$.

4.6 Future Works

Besides future works already mentioned during this section, OITE could have its performance enhanced by these two potential improvements:

1. To improve class label's recognition, a language detection algorithm could be implemented to determine what language the class names are in, which would allow for more accurate OCR.

2. Relationship detection could be enhanced by the addition of an algorithm to split the relationships into the separate lines that they are composed of, similar to that implemented in the work of Chen et al.. This enables the verification of whether two associations are the same (when they are, e.g., divided by a label) or distinct.

5. LLM TRANSLATION

While the previous two sections focused on developing a tool for translating OntoUML diagram images into computer-readable formats, this one will examine the potential of utilizing existing Large Language Models (LLMs) for this task. The following subsections will cover the methodology of the experiment conducted to assess this approach's plausibility and present its results.

5.1 Methodology

This experiment is based on the one performed by Conrardy & Cabot [6], where they tested a selection of LLMs to determine the feasibility of using them to translate hand-drawn UML class diagrams. In their experiment Conrardy & Cabot found that Chat GPT performed best for translating the images, so for this one the GPT4O model in the latest version of ChatGPT Plus [15] was used. Four diagrams of increasing complexity from the OntoUML catalog were selected for the test:

- **Easy**: Media object [5 classes, 2 generalizations]
- **Medium**: offline [7 classes, 3 generalizations]
- **Hard**: full_1_human_rights_problems [11 classes, 6 generalizations]
- **Very Hard**: user [18 classes, 4 generalizations]

The selected images can be found in the "Test Images" directory of the git repository [11]. They were uploaded into ChatGPT along with the following prompt:

"Can you turn this OntoUML diagram into the corresponding class diagram in OntoUML vocabulary in turtle format? Only extract the semantic information, like the classes(only names and stereotypes) and generalization relationships. Make sure to not create false relationships between classes which are both connected to the same class but not each other.

"This is what the turtle syntax for a class should look like:
ontouml:{unique identifier} a ontouml:Class ;
ontouml:name "{name}"@en ;
ontouml:stereotype ontouml:{stereotype} .

This is what the correct syntax for a generalization should look like:
ontouml: {unique identifier} a ontouml:Generalization ;
ontouml:general ontouml: {general class identifier} ;
ontouml:specific ontouml: {specific class identifier}

For the unique identifiers generate UUIDs.
Make sure you include all the generalization relationships."

The prompt was derived from the short prompt used by Conrardy & Cabot [6]. It defines the input (*OntoUML diagram*), desired output (*Diagram in OntoUML vocabulary in turtle format*) and what should be included (*classes and generalizations*). As Conrardy and Cabot found the shortest prompt they used to perform the best, an attempt was made to limit the prompt length, however an example of the OntoUML Vocabulary syntax had to be included, as during initial testing ChatGPT was not able to produce correct results without it. The test was performed three times for each image as the output of ChatGPT is not deterministic. Each conversion was performed in an empty new conversation to ensure that no retained context from the previous attempts would affect the current one. For each attempt the number of mistakes was counted. A mistake is every class that was missed during conversion, or generalizations that were missed or invented by the LLM. In the next subsections, the results of the experiment are displayed and discussed.

5.2 Results

The full results of the experiment can be found in ***Table 1***.

Table 1 *ChatGPT test*

Image	Mistakes of ChatGPT			Mistakes of OITE
	Attempt 1	Attempt 2	Attempt 3	
Easy	0	0	0	0
Medium	0	0	0	0
Hard	2	4	2	2
Very Hard	7	4	4	4

The table shows that the LLM was able to translate both the easy and medium diagrams without making any mistakes in each attempt. On the other hand, the model made at least 2 mistakes in each of its attempts when translating the hard image. A recurring mistake in each attempt was the 'skipping' of a class when it was both the general class for one generalization, and the specific class in another, as illustrated in ***Figure 5***. In this example ChatGPT would detect a generalization from **A** to **B** and from **C** to **A**, instead of **A** to **B** and **B** to **C**. This causes the information that **A** is a generalization of **B** to be lost, and thus constitutes a mistake.

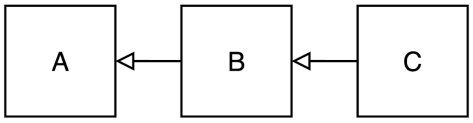

Figure 5 *Class "Skipping" Generalization*

This behavior was also observed in similar situations in the Very Hard example. Another mistake, found in the Hard example, was a generalization relationship that was missed in each of the 3 attempts. This is likely due to the fact that the particular relationship was rather long and was illustrated further away from all the classes and other relationships, which were clustered in the middle of the image. The additional two mistakes in attempt 2 were caused by two normal relationships being confused for generalizations.

During the translation of the Very Hard image, the LLM made the following four mistakes on every attempt: There were two cases of class “skipping” and two generalizations that were not detected. On the first attempt, two classes were not detected and, together with the generalization between them, constituted three additional mistakes.

Overall, ChatGPT performed well on simple diagrams, however it started making mistakes on diagrams similar to the “hard” one in this experiment. As complexity increased, so did the mistakes. Another notable behavior was observed during other tests, where ChatGPT would invent generalizations between classes with similar names. An example of that was when during the translation of the diagram “users” from the OntoUML Catalog, the LLM invented a generalization between the classes “User” and “Management Users” even though they are not directly connected in the original diagram.

For anyone interested in re-creating the experiment, a GPT model was created using the prompt employed in the experiment, allowing users to upload images of OntoUML diagrams for translation [12].

5.3 Comparison With OITE

The diagrams utilized to test ChatGPT were also translated using OITE, as shown in the last column of **Table 1**. While both methods yielded similar results, the errors differed between the two approaches.

ChatGPT did not exhibit the limitations observed in OITE. The LLM was able to consistently identify classes where the text overlaps with the class rectangle, which OITE struggled with due to the limitations of rectangle detection. Additionally, ChatGPT was able to correctly detect Portuguese text within class images, while OITE attempted to read it as English, resulting in the loss of special characters. When it comes to relationship recognition, ChatGPT demonstrates the ability to separate relationship lines that overlap each other, unlike OITE.

While ChatGPT excelled in the mentioned items, it also had its own limitations, as discussed in Section 5.2. These limitations are exacerbated by the fact that ChatGPT operates as a ‘black box’ to the user. Consequently, users have no way of understanding why it “skips’’ classes in generalizations or misses generalizations entirely, making performance improvements a challenge. Furthermore, it is important to note that utilizing ChatGPT’s visual capabilities has a financial cost associated with it, either through a subscription to ChatGPT Plus or through the use of its paid API.

5.4 Future Works

This experiment demonstrates that using LLMs to translate images of OntoUML diagrams into machine-readable formats is feasible. ChatGPT successfully translated simple diagrams autonomously, while further testing showed that the translation of more complex diagrams was possible through a human-in-the-loop approach, where a person reviewed ChatGPT’s output and asked it to correct it in the case of mistakes. Additional prompt engineering might be able to prevent the issue of class “skipping” in the initial prompt, as ChatGPT can correct it when asked to in further prompts.

Additionally, given the different limitations of ChatGPT and OITE, a promising future direction would be to develop a hybrid approach that leverages the strengths of each. For instance, OITE could generate a machine-readable Turtle file, which ChatGPT could then verify against the diagram image and correct any mistakes. This hybrid method presents a valuable opportunity for future research and improvement in ontology translation.

6. CONCLUSION

This study explored translating OntoUML diagram images into machine-processable formats using machine learning and image processing techniques to enhance their usability in ontology. It covers the development and testing of the OntoUML Image Taxonomy Extractor (OITE), which classifies and converts OntoUML diagram images, and an experiment using ChatGPT to assess its feasibility for this task.

Transfer learning with the InceptionV3 model was effective for detecting OntoUML diagram images, achieving 83% accuracy with the binary classifier and 91% with the 3-class classifier. However, distinguishing OntoUML from UML class diagrams was challenging because of their visual similarity, indicating a need for more specialized datasets or the implementation of additional functionality.

Most of the tested methodologies for converting OntoUML diagram images proved suitable. Preprocessing by converting images to gray scale and applying Gaussian sharpening enhanced edge detection. Class identification was achieved through rectangle recognition and clustering, followed by text extraction using OCR and regular expression check. For relationship detection, contour detection was used and a 3-class classifier determined the relationship types. While functional, this approach had limitations affecting overall effectiveness. The gathered data was then translated into OntoUML Vocabulary with the help of RDF, resulting in a machine-processable model.

The use of a visual large language model like ChatGPT4V has proved viable for translating OntoUML images into machine-processable formats. ChatGPT performed well with simple and medium-complexity diagrams, but its accuracy diminished with more complex diagrams, suggesting the need for a human-in-the-loop approach for higher complexity.

Comparing OITE to the LLM revealed that, although their performance levels were similar, the types of their mistakes differed. To maximize performance, a hybrid approach is proposed combining OITE’s initial translation with ChatGPT used for validation and correction.

Overall, this research offers valuable insights and lays a foundation for future advancements in translating OntoUML diagrams, aiming to benefit the field of ontology.

During the preparation of this work, the author utilized ChatGPT to generate ideas, assist with writing the text, and aid in some code generation. The author carefully reviewed and edited the content as necessary, taking full responsibility for the final version of the work.

REFERENCES

[1] Barcelos, P. P. F., Sales, T. P., Fumagalli, M., Fonseca, C. M., Sousa, I. V., Romanenko, E., Guizzardi, G., & Kritz, J. (2022). A FAIR model catalog for ontology-driven conceptual modeling research. In J. Ralyté, S. Chakravarthy, M. Mohania, M. A. Jeusfeld, & K. Karlapalem (Eds.), *Conceptual modeling. ER 2022. Lecture notes in computer science* (Vol. 13607). Springer, Cham. https://doi.org/10.1007/978-3-031-17995-2_1

[2] Black, P. E. (Ed.). (2019, May 15). Levenshtein distance. In *Dictionary of algorithms and data structures*. Algorithms and Theory of Computation Handbook. CRC Press LLC. Retrieved 2024, June 21, from https://www.nist.gov/dads/HTML/Levenshtein.html

[3] Chen, F., Zhang, L., Lian, X., & Niu, N. (2022). Automatically recognizing the semantic elements from UML class diagram images. *Journal of Systems and Software, 193*, 111431. https://doi.org/10.1016/j.jss.2022.111431

[4] Chiefeweight. (n.d.). UML code trace. Gitee. Retrieved June 26, 2024, from https://gitee.com/chiefeweight/uml-code-trace

[5] Chollet, F., & others. (2015). Keras. https://keras.io

[6] Conrardy, A., & Cabot, J. (2024). From image to UML: First results of image-based UML diagram generation using LLMs. *arXiv preprint arXiv:2404.11376*. https://doi.org/10.48550/arXiv.2404.11376

[7] Gosala, B., Chowdhuri, S. R., Singh, J., Gupta, M., & Mishra, A. (2021). Automatic classification of UML class diagrams using deep learning technique: Convolutional neural network. *Applied Sciences, 11*(9), 4267. https://doi.org/10.3390/app11094267

[8] Guizzardi, G. (2005). *Ontological foundations for structural conceptual models* [PhD Thesis - Research UT, graduation UT, University of Twente]. Telematica Instituut / CTIT. https://research.utwente.nl/en/publications/ontological-foundations-for-structural-conceptual-models/

[9] Guizzardi, G., Benevides, A., Fonseca, C., Porello, D., Almeida, J., & Prince Sales, T. (2022). UFO: Unified foundational ontology. *Applied Ontology*. https://doi.org/10.3233/AO-210256

[10] Ho-Quang, T., Chaudron, M. R. V., Samúelsson, I., Hjaltason, J., Karasneh, B., & Osman, H. (2014). Automatic classification of UML class diagrams from images. In *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference* (pp. 399-406). Jeju, Korea (South). https://doi.org/10.1109/APSEC.2014.65

[11] Kaishev, S. (2024). OntoUML image converter [Computer software]. GitHub. https://github.com/SimeonKaishev/OntoUML_IMG_Converter

[12] Kaishev, S. (2024). OntoUML: Image to vocab. Retrieved from https://chatgpt.com/g/g-8l61Egeo8-ontouml-image-to-vocab

[13] Karasneh, B., & Chaudron, M. R. V. (2013). Extracting UML models from images. In *Proceedings of the 2013 5th International Conference on Computer Science and Information Technology* (pp. 169-178). Amman, Jordan. https://doi.org/10.1109/CSIT.2013.6588776

[14] OpenAI. (2023, September 25). GPT-4V(ision) system card. OpenAI. Retrieved 2024, June 21, from https://openai.com/index/gpt-4v-system-card/

[15] OpenAI. (2023). ChatGPT Plus. Retrieved [June 21st, 2024], from https://www.openai.com

[16] RDFLib Development Team. (2023). RDFLib: A Python library for working with RDF. GitHub. https://github.com/RDFLib/rdflib

[17] Sales, T. P., Fonseca, C. M., & Barcelos, P. P. F. (2022). OntoUML vocabulary specification - Version 1.1.0. Retrieved from https://w3id.org/ontouml/vocabulary/docs

[18] Shcherban, S., Liang, P., Li, Z., & Yang, C. (2021). Dataset of the paper "Multiclass classification of four types of UML diagrams from images using deep learning". https://doi.org/10.5281/zenodo.4595957

[19] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*. https://doi.org/10.48550/arXiv.1512.00567

[20] Tavares, J. F., Costa, Y. M. G., & Colanzi, T. E. (2021). Classification of UML diagrams to support software engineering education. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)* (pp. 102-107). Melbourne, Australia. https://doi.org/10.1109/ASEW52652.2021.00030

[21] Verdonck, M., & Gailly, F. (2016). Insights on the use and application of ontology and conceptual modeling languages in ontology-driven conceptual modeling. In I. Comyn-Wattiau, K. Tanaka, I. Y. Song, S. Yamamoto, & M. Saeki (Eds.), *Conceptual modeling. ER 2016. Lecture notes in computer science* (Vol. 9974). Springer, Cham. https://doi.org/10.1007/978-3-319-46397-1_7

[22] Verdonck, M., Gailly, F., Pergl, R., Guizzardi, G., Martins, B., & Pastor, O. (2019). Comparing traditional conceptual modeling with ontology-driven conceptual modeling: An empirical study. *Information Systems, 81*, 92-103. https://doi.org/10.1016/j.is.2018.11.009

[23] World Wide Web Consortium (W3C). (2014). *RDF 1.1 Concepts and Abstract Syntax*. https://www.w3.org/TR/rdf11-concepts/