SkeMo: Sketch Modeling for Real-Time Model Component Generation

Alisha Sharma Chapai and Eric J. Rapos
Department of Computer Science and Software Engineering
Miami University
Oxford, Ohio, USA
sharma26@mamioh.edu, rapose@miamioh.edu

***Abstract*—Software modeling is a powerful tool in the design and implementation of high-quality software systems. Models can be used from high-level design to formal code generation, with various applications in between. Often, software models are initially created informally, by sketching on a whiteboard or paper during the early design phase of the system, and eventually converted into formal models using advanced modeling tools. The formalization of sketches into actual model components can be time-consuming, error-prone, and laborious. To address these shortcomings, we present SkeMo, an environment for real-time model component generation from sketch-based inputs. We curated a sketch dataset of 3000 images of various class diagram components and implemented a powerful Convolution Neural Network to classify input sketches as model components. We integrated our sketch classifier into an existing web-based model editor and added a touch interface to support sketch-based modeling. We evaluated the SkeMo environment in two ways: through ten-fold cross-validation of the image classifier and collection of metrics and feedback from a 20-participant user study. Based on our results, sketch-based modeling demonstrates significant promise as an intuitive interface that is both easy to use and allows for faster model creation among most users.**

***Index Terms*—model-driven software engineering, machine learning, deep neural network, convolution neural network, image recognition, sketch recognition, class diagrams, classifiers, interface design, touch interface, collaborative modeling, assistive modeling, user studies**

I. Introduction

Model-driven software engineering (MDSE) allows engineers to create and maintain complex software systems through the development of abstract software models and the generation of code from these models. To best leverage the benefits of software modeling, various other means of interacting with models could be used to enable a wider range of practitioners to become proficient in modeling. With more and more devices offering touch interfaces and an increased familiarity with their use, it seems natural that we could leverage these expanded interfaces to informally input model elements to be used in formal model creation. Our work aims to leverage touch interfaces as a means of interacting with software modeling to create an environment that would enable users to create high-quality software systems using an MDSE approach.

A. Motivation

Sketching and diagramming are commonly used in design, architecture, and engineering [1]. Software architects and developers create informal sketches to understand, design, and share their ideas within the team [2]. Such sketches are created using different media, from analog media such as paper or a whiteboard to digital media such as smartphones and computers. Sketching is preferred in the early stage of software development [3] as it fosters creativity [4] and collaboration among various stakeholders [3]. Sketching provides a means of effective communication and collaboration by expressing ideas simply and abstractly. Moreover, the availability of digital devices such as mobile phones, tablets, touch-screen displays, and input devices with sufficient computing power has made free-form sketching more accessible and collaborative [5]. Additionally, web-connected applications have further facilitated collaboration by allowing users to refine and formalize their digital sketches collaboratively [5], [6]. However, such informally drawn sketches often tend to deviate from standard notation, such as Unified Modeling Language [2], [7]. Interpreting sketches with informal notation can be difficult later in the development process [6]. In these cases, the models often need to be converted to formal models as the development process begins. However, manually converting sketched models into formal models can be a tedious and time-consuming process. This can be avoided by developing tools that support sketching and transforming sketches into required standard models.

Although several attempts have been made to convert informal sketches into formal models, there are limited web-based modeling environments available [8]–[11]. Existing sketch-based modeling applications are mostly supported on desktop computers [6], [12], [13], e-whiteboards [13], [14], or mobile devices [5], [6], which require installation and setup. However, with the widespread availability of digital devices and the internet, we propose a web-based framework for real-time sketch recognition and formal model creation. The proposed framework leverages the simple and abstract nature of sketches and uses a Convolution Neural Network (CNN) for creating formal models in real-time, which can be easily accessible on any device with internet connectivity, making it a convenient and efficient solution for creating formal models.

B. Research Questions & Contributions

The goal of this study is to demonstrate the effectiveness of real-time sketch conversion as a means for creating software models. To demonstrate the effectiveness, we pose and provide answers to the following research questions:

RQ1: *How effective is our proposed approach for converting sketches to software models in real-time using an image classifier?*

RQ2: *Can sketch-based software modeling be faster than traditional drag-and-drop methods for creating software models?*

RQ3: *Can our approach to real-time sketch conversion contribute to collaboration when creating software models?*

Through answering these research questions, our work makes the following significant contributions:

1) creation of a structural modeling sketch dataset
2) development of a structural modeling sketch classifier
3) SkeMo: an implementation of a web-based modeling framework for sketch-based modeling using touch input
4) a formalized evaluation of the SkeMo environment for both effective model creation and collaboration

II. Background and Related Work

A. Software Modeling

Numerous modeling tools have been developed to facilitate model creation, transformation, and code generation in software development. These tools come in various forms, from open-source to commercial, complete framework to individual, and desktop-based to web-based modeling tools. It is worth noting that modeling tools are distinct from drawing tools despite the common misconception that they are interchangeable. A modeling tool may not necessarily function as a drawing tool, and vice versa [15]. A drawing tool can only be a modeling tool if it understands the semantics of the drawn elements [15].

A web-based modeling tool is a software development paradigm that utilizes web technologies to create software models. Unlike desktop-based tools, web-based modeling tools allow users to access them from any device with an internet connection without needing to download or install any applications. Web-based UML modeling tools, also known as online UML tools, facilitate the drawing and sharing of models online, thereby supporting collaborative work.

DiGennaro et al. [8] developed a web-based modeling environment called `SuMo` that supports the creation of valid structural models, model transformations, and the generation of transparent code. `Umple` [9], [10] is a web-based framework that supports Model-Driven Development with forward and reverse engineering capabilities. The web-based environment, known as `UmpleOnline`[1], provides users with an interface to draw a UML class diagram and automatically generates the corresponding code in languages such as Java and C++. Additionally, it also allows users to write their code and convert it into Umple models. However, it does not provide model-to-model transformation and the creation of instance models based on users' meta-models [8]. `GenMyModel`[2] is another web-based modeling tool that provides users with a UML editor to create UML diagrams, such as class and sequence diagrams. Additionally, it provides support to export models in XML Metadata Interchange (XMI), which is a standard format for exchanging models, and also has code generation capabilities from models. There are other online UML tools such as `Draw.io`[3], an online version of `Visual Paradigm`[4], and `UMLetino`[5].

Unlike drawing tools, our work focuses on creating an online modeling environment with support for sketch-based modeling. Furthermore, the existing web-based modeling tools do not support sketch-based modeling, as they only provide an editor to drag or select class diagram components. In contrast, our work aims to combine web-based and sketch-based modeling for real-time conversion of sketches into formal models. With the combined web-based and sketch-based modeling capabilities, SkeMo has the potential to open up new possibilities for software development and collaboration.

B. Image and Sketch Recognition

Several studies have proposed sketch recognition models based on convolutional neural networks (CNNs) for different purposes. Kabakus [16] developed a simple CNN-based sketch recognition model that achieved an accuracy of 89.53% on the `Quick Draw` dataset. Yu et al. [17] proposed `Sketch-a-Net`, a deep neural network-based sketch recognition model that outperformed hand-crafted feature-based techniques. Their model achieved an accuracy of 77.95% on the TU-Berlin dataset [18]. Google also developed a web-based drawing game called `Quick Draw!`[6], which uses neural networks to recognize sketches. Li et al. [19] proposed `Sketch-R2CNN`, an architecture for vector sketch recognition that used RNN for temporal ordering and grouping information analysis and CNN for sketch recognition. Brieler et al. [20] proposed a model-based recognition engine for recognizing sketched diagrams using transformers. Their approach specifically focused on solving issues related to the clustering and segmentation of image strokes. Similarly, Schäfer et al. [21] proposed `Arrow R-CNN` for detecting symbols and structures in handwritten diagrams or images containing flowcharts, graphs, or diagrams with directional elements.

Although different CNN architectures are designed for various image processing tasks, we opted to develop our own CNN architecture to address our specific use case instead of adapting an existing one. In comparison, a specific example like `Arrow-RCNN` is based on a pre-existing CNN and is considerably larger than our model. Our approach focuses solely on the classification of five sketches of class diagram components, and the benefits from its smaller and more dedicated nature make it well suited for our application.

[1]https://cruise.umple.org/umpleonline/
[2]https://www.genmymodel.com/
[3]https://www.draw.io/
[4]https://online.visual-paradigm.com/
[5]http://www.umletino.com/
[6]https://quickdraw.withgoogle.com/

C. Sketch-Based Modeling

Mangano et al. [22] developed a user-friendly interactive whiteboard system called `Calico`. It is a sketching tool focused on supporting the beautification of informally drawn sketches [23] rather than specific UML diagrams. Similarly, Wüest et al. [5] created an android application called `FlexiSketch` focusing on creating free-form sketches, defining their metamodel, and later reusing them. Chen et al. developed a Visual Basic application called `SUMLOW` [14] that allows users to sketch UML design elements, formalize the sketches, and export them to existing UML CASE tools. Similarly, `Tahuti` [12] is another tool that allows users to draw UML class diagrams with a computer mouse with support for multi-stroke sketch recognition. However, `SUMLOW` and `Tahuti` use two different views, one for sketching and another for viewing the formal diagram, requiring users to often switch between the two for drawing and viewing. Unlike these, SkeMo focuses on automatically converting user-drawn sketches into formal model components on the same view. OctoUML [13], [23] is closely related to our work as it supports creating both formal and informal diagrams on the same view. While OctoUML has expanded functionality, the conversion of sketches to model components is carried out through manual intervention instead of being executed instantenously. OctoUML requires users to select sketches for conversion; SkeMo automatically converts them. OctoUML does not recognize sketches of different types of relations since PaletoSketch [24], which OctoUML uses for sketch recognition [23], only recognizes basic shapes. SkeMo automatically adds different types of relations since it is built on its own sketch classifier for improved classification. With onscreen keyboard support, OctoUML supports adding/updating the properties of components, such as updating the name of attributes and class names. SkeMo also allows for the same updates through onscreen/hardware inputs and UI elements. OctoUML is an interactive whiteboard-based application, whereas SkeMo is a web-based app that enables access via any connected device. Additionally, OctoUML does not support code generation from models [13]. Since IML, the existing web-based environment SkeMo uses supports these model-to-code transformations, SkeMo shares this capability. Unlike the evaluation of OctoUML, SkeMo presents the observational results from the user study, in addition to technical evaluation, where we demonstrate that sketch-based modeling can be slightly faster than using traditional input methods.

III. Target Application

We chose to incorporate SkeMo into an existing web-based modeling tool, the Instructional Modeling Language (IML) [8], [25], to support sketch-based modeling on the web. The IML model editor supports creating structural models based on UML class diagrams. The selected environment is capable of modeling classes, attributes, and several types of relations. The web-based model editor primarily includes the following elements: **i) Modeling Pane:** the central canvas where users can add, edit, and delete model components such as classes, attributes, and relations, **ii) Palette:** a collection of icons representing model elements, which users can select and drag onto the modeling pane to add to the model, **iii) Properties Table:** a table that shows the properties of a selected model component for inspection and editing.

All these components together provide a seamless user interface for creating and editing UML-like class diagrams within the web-based modeling tool. Furthermore, the web-based modeling editor also supports model transformation and code generation from the created software models.

A. Action Selection

Our goal is to provide users with a platform that allows for the easy and enjoyable creation of structural models through real-time sketch conversion to model elements. In order to effectively implement sketch-based modeling in our web-based model editor, we need to define the actions that users can perform. It was important that we included the ability to add all possible elements in the editor so that no functionality was lost by the sketch-based editor. Based on this, we determined we needed to account for the following actions:

- **Adding a class:** Enables users to draw an informal sketch to add a class to the model editor.
- **Adding attributes to a class:** Allows users to sketch on existing classes in the editor to add attributes to them.
- **Adding a relationship between classes:** This action enables users to draw the required relationship between two existing classes on the model editor.
- **Select and manipulate model components:** This enables users to select and interact with model components by switching from drawing to selection mode with a specific touch gesture. Users can then move elements, update names and values, change attribute data types, and take advantage of other base IML functionality.
- **Deletion:** This action allows users to delete the selected model components in the editor, for example, selecting and deleting an attribute from a class.
- **Undo:** An undo feature allows users to undo any previous actions, allowing users to rectify errors or backtrack changes made during the modeling process.

In addition to the above list of actions, SkeMo allows for model updates via non-sketch interactions. The current version of the SkeMo does not support updating model properties via sketching. However, users can use on-screen or traditional keyboards to change names, cardinality, visibility, etc. Collectively, these actions encompass a comprehensive range of elements that can be incorporated into the SkeMo model editor for sketch-based modeling.

IV. SkeMo: Sketch-Based Software Modeling

We created SkeMo, our approach to helping software developers and other users create software models through real-time conversion of sketches. The primary purpose is to support the instant conversion of informal sketches to formal model components, which later can be utilized for other tasks such as code generation, documentation, etc. Here, the term "informal" indicates that the sketch lacks inherent meaning and serves as a mere representation of an element, with formal semantics being added upon its conversion into a model element. To achieve our goal, we extended an existing web-based modeling tool to support sketch-based modeling, allowing users to create a UML-like class diagram using sketches. This implementation of a sketch-based environment to the model editor includes the addition of other functionalities such as sketch recognition, analysis, and conversion to model elements. Fig. 1 provides an overview of the SkeMo architecture and our approach to implementing sketch-based modeling on the web platform. We provide the implementation details of our approach in the following sections.

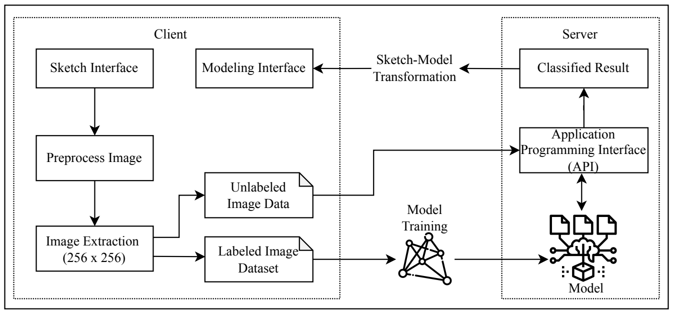


Fig. 1: SkeMo Application Architecture

A. Sketch Collection and Model Interface

Our approach to supporting sketch-based modeling includes two interfaces - sketching and modeling. The sketching interface is used to collect user-drawn sketches, whereas the modeling interface is used to add and interact with corresponding model elements.

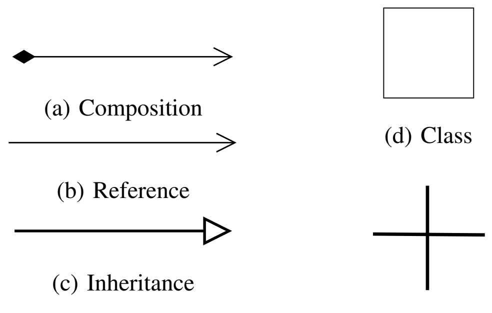

(a) Composition
(b) Reference
(c) Inheritance
(d) Class
(e) Attribute

Fig. 2: Sketchable Components of a Class diagram

In addition to the final sketch interface for SkeMo, we needed to first decide on a method for sketch recognition to generate the training data for our model. There are two main approaches to sketch recognition: during the drawing process or after completing the drawing. Sketch recognition during drawing involves analyzing and recognizing each stroke information as the user draws an object. In contrast, post-drawing recognition involves analyzing the entire sketch as an image once the user has finished drawing, using various image recognition techniques. In this work, we utilize the latter method for sketch recognition. After determining the sketch recognition methodology, we created a separate web-based sketch interface to collect sketch data, implemented using HTML, CSS, JavaScript, and Bootstrap for the front end and PHP as a server-side language to collect sketches drawn by the users on the back end. The interface provides users with a list of symbols to sketch, along with the necessary instructions. It consists of two buttons: a clear button to remove any incorrectly drawn sketches and a save button to allow users to save the sketch they have drawn. The collected data was then automatically labeled and used to train our image classifier.

After completing the sketch collection, we then adapted the stand-alone sketch interface to overlap with the modeling interface of the existing web-based modeling tool. We added a toggle UI element to implement sketch-based modeling in the tool. When a user turns on the switch element, the sketch interface is overlayed over the modeling interface to capture and classify user-drawn sketches and then automatically add the classified model component to the modeling interface.

When the sketch modeling feature is enabled, our application indicates that the user is in *drawing mode*, allowing them to draw freely on the sketch interface. However, when using this mode, users will not be able to directly interact with model elements (e.g., to move them). They must be in *selection mode* to select and move model elements added to the modeling interface. To switch between drawing and selection modes, users can easily tap the interface with two fingers or press and lift the stylus pen button (if using one). For example, when a user draws a rectangle on the sketch interface, it gets converted into a class element on the model interface. Suppose that the user wants to move the added class; in this case, tapping the interface with two fingers will transfer the pointer events from the sketch to the model interface, and the user will be able to select and move components around as needed. This reduces the need to turn the toggle UI element on/off to draw and move model elements on two different interfaces.

Each of these elements enables a seamless interaction with the existing modeling interface, while supplementing it with the benefits provided by the sketch-based modeling environment.

B. Sketch Classification

To convert sketches to model elements in real time, we required a trained CNN capable of processing images and indicating their intended representation in the model. To achieve this within SkeMo, we built a classifier capable of recognizing informal sketches of model components. We began by collecting sufficient sketch data to map to the formal model components. In order to map informal sketches to formal class diagram components, we opted to choose a similar set of icons from the existing web-based model editor. Fig. 2 illustrates the list of symbols that we use for our sketch data collection, which correspond directly to the possible model elements. For example, we consider the informal sketching of a rectangle as adding a class and a plus sign within a class as adding an attribute to that class in the model editor.

We provide detailed descriptions of our data collection, data processing, and classifier development methodology in the following subsections.

1) Data Collection: We collected training data from researchers within our research group. Six users participated in the data collection process, where each participant was asked to draw using the canvas area according to the instructions provided on the sketch interface. Users were asked to draw each of the five components from Fig. 2, one at a time and, save the sketches after completing them. The sketch collection process was carried out in two ways: initially, users were asked to draw using their fingers, and later, they were requested to repeat the same process using a stylus pen. In both cases, we collected 50 images for each of the five components, resulting in a total of 500 sketches from each user, amounting to 3000 sketches in total from all users. Data from users was collected using a Microsoft Surface Pro 2, a device supporting touch inputs via both a finger and a pen.

The collection interface automatically labeled the sketch drawn by the user according to the provided instructions. For example, if a user is asked to draw a reference relation, the image is labeled as a reference once the user finishes drawing and saves it to the server. After collecting user sketches, we manually inspected the collected images to confirm all images were correctly labeled. For example, if a user accidentally draws a reference relation instead of a composition, the wrong image would be labeled here, affecting the model training. In any case of any labeling errors observed during our manual inspection, we asked the user to redraw mislabeled images again.

The 3000 images were used to train and evaluate our CNN for classifying user sketches. The evaluation process is explored further in Section V-A.

2) Training Our Classifier: Data processing is crucial in machine learning, as it involves preparing high-quality data to train machine learning algorithms. One important aspect of data processing, especially in image classification tasks, is ensuring that the images used for training are of suitable size and quality. Image resolution can significantly impact the performance of a CNN model. Therefore, using images of appropriate sizes and quality is important for better model performance. Larger image sizes tend to contain more detailed information, which can potentially improve the accuracy of the model's predictions. However, using larger images also requires more computational resources and longer processing times during training and inference. On the contrary, reducing the image resolution too much can result in the loss of important features that are essential for accurate classification. This can lead to a decrease in the model's performance, as the reduced resolution may not capture the necessary details needed for accurate predictions. Therefore, to create a balance between resolution and computation resources, images are commonly scaled to an appropriate size for optimal performance, typically ranging from 64 x 64 to 256 x 256 resolution for training CNNs [26].

For SkeMo, we collected images of size 256 x 256 as our training data for a CNN model designed for sketch classification. To extract the image, we first capture the sketch from the original canvas and transfer the image data to another canvas of 256 x 256 size. Our approach records touch points associated with each user's stroke while sketching on the interface. These touch points provide information about the location of the user's input on the canvas. From the recorded touch points, we extracted the x and y coordinates from the top-left corner and the bottom-right corner of the canvas to understand the spatial extent of the drawing. Once the user finishes drawing, we obtain additional image information, such as width and height. We compute image width and height from the x and y coordinate information extracted earlier, allowing us to understand the overall size of the user-drawn sketch. The overall process can be considered image cropping from the original canvas and copying it to another canvas of the required size to generate all images of the same size, preserving the aspect ratio.

We used the data collected from the sketch interface to train a convolutional neural network (CNN) model. We used single channel image data for our training. We used the Keras Sequential Model API to create the layers of the neural network. The final CNN architecture consists of seven convolutional layers followed by two fully connected layers. Additionally, max-pooling layers were incorporated after each convolutional layer, and ReLu (Rectified Linear Unit) was used as the activation function. We also added a data augmentation layer, and the final CNN layer employed the LogSoftmax activation function, and we used Adam as an optimization algorithm. It is important to note that the classifier is trained to recognize individual model elements rather than more complex sketches by design. Since the underlying editor has extensive error checking and guidance, element-by-element insertion allows SkeMo to leverage these benefits. Therefore, the CNN model was trained to classify sketches into one of five classes: class, attribute, reference, inheritance, and composition. The trained model was then ready to be deployed alongside the modeling interface.

C. Model Application

When the user is in *drawing mode*, our approach automatically extracts the sketch drawn on the interface when a user lifts the stylus pen or finger and does not touch down again on the interface within 750ms. This specific time interval accommodates multi-stroke images, providing users ample time to lift their pens and finalize their drawings before extraction, while also avoiding excessive delays that may impact the user experience. An example of the SkeMo interface where a user has sketched a relation, but it has not yet been replaced is shown in Fig. 3. Following the 750ms delay, we extract the image of size 256 x 256, which is consistent with the image size used in the training data of the model used for classification. Once the image is extracted from the

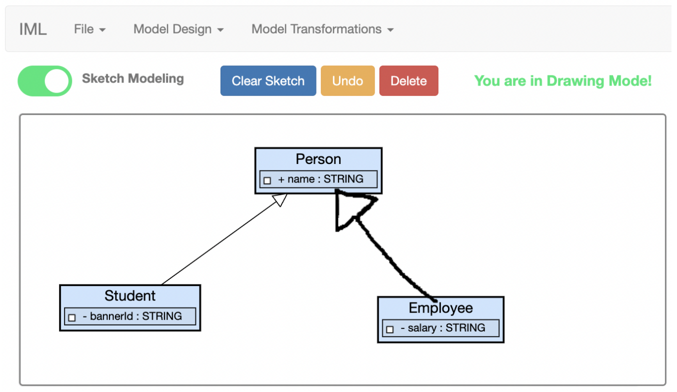


Fig. 3: SkeMo Model Editor

sketch interface, the client calls the API on the Node.js server with the extracted image data as input for classification. The server receives the request from the client and extracts the bitmap information from the image data using the `PNG.js`[7] library, which is a JavaScript library specifically designed to encode and decode PNG images in Node.js applications. With the bitmap information, the server prepares the image for prediction by converting it into single-channel image data. The server then uses the trained model to classify the image based on its single-channel representation.

Once the classification is completed, the server returns the classified result to the client. The client then adds the corresponding element to the model directly, resulting in real-time sketch conversion into an interactive model element. The model element is inserted into the modeling interface at the same position where the user initially drew the sketch. Each of the modeling functionalities SkeMo is capable of is explored in the following paragraphs.

Addition of a class: If a user draws a rectangle and it is correctly classified, a class with the next sequential default name is added to the modeling interface. The coordinates of the rectangle's top-left corner on the drawing canvas are used as the top-left corner of the class inserted to maintain consistency between the sketch and model.

Addition of attribute: When a user sketches a plus sign over any class present in the model, if the sketch is correctly classified, it will add an attribute to the underlying class with the next sequential default name. If the user tries to draw plus outside of any class in the interface, it will inform the user that attributes can only be added to the classes. Currently, the "+" adds an attribute and does not refer to public visibility. While defaulting to the public, once an attribute is added, its visibility can be changed to private or protected using the dropdown menu. The potential future plan includes incorporating informal sketches of "-" and "#" for adding/changing attributes to private and protected visibility, respectively.

Addition of relationships: The addition of relationships occurs when a user draws any of the three types of relations (reference, inheritance, and composition) on the sketch interface, attempting to establish a relationship between two classes present on the modeling interface. Once our sketch classifier identifies the type of relation, the corresponding relation is added between the two classes in the model. Users can also add self-reference relations by drawing informal sketches of reference overtop the existing class on the modeling interface. It is important to note that the underlying model editor does not support the addition of a single bi-directional relation; however, users can add two distinct relations with unique names and properties in the same way as adding any other relation using SkeMo as means for adding bi-directional relations between classes.

One of the main challenges in adding a relationship between classes is identifying the source and destination classes from the sketch. To address this challenge, SkeMo uses the density of image pixels rather than considering the drawing order of the sketch, as users can draw relations in various directions. First, we check the orientation of the sketch and then divide the image into two equal halves along its length, considering the half with more pixels as the destination and the other half as the source for reference and inheritance relations. However, this case is reversed for the composition relation where the diamond shape is at the source. Using coordinate information of the source and destination ends from the sketch, the appropriate classes are sought in the model by finding the class nearest each end.

V. Evaluation

In this section, we present the evaluation of SkeMo and provide answers to our three research questions.

Since the sketch classifier is a crucial component of our approach, the first step in evaluating our approach is to assess the performance of our trained model. We used standard metrics commonly used in machine learning, including Precision, Recall, and F1-score, to evaluate the accuracy and effectiveness of our model in predicting sketches drawn by users. Precision measures how many images are correctly predicted out of the total predicted images (ratio of true positives to the sum of true positives and false positives). Specifically, precision determines how precisely the model can predict sketches correctly, while recall measures the ability of the model to identify all relevant sketches (ratio of true positives to the sum of true positives and false negatives). The F1-score represents the harmonic mean of the model's precision and recall. We further discuss the details of evaluating our image classifier in Section V-A, where we provide detailed insights into the performance of our trained model.

In addition to evaluating the sketch classifier, we also conducted a user study to collect qualitative and quantitative data from users about their experiences using SkeMo to create software models. We carried out this portion of the study to answer **RQ2** and **RQ3**, which focus on the speed and collaboration aspects of our approach. We present the results of our study in Section V-D, providing valuable information on

[7]https://github.com/pngjs/pngjs

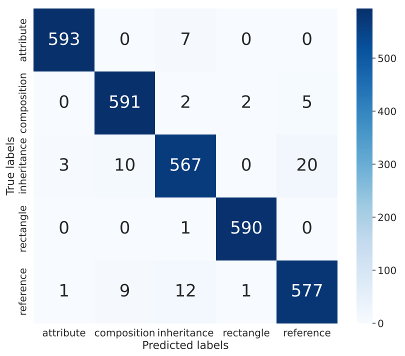


Fig. 4: Sum of Confusion Matrices from Cross-Validation

user experiences and perceptions while using sketch modeling in the context of our web-based approach.

Data from our evaluation, including our training data, back-end source code, and data from our user study is available via the Open Science Foundation [27].

A. Evaluation of Sketch Classification

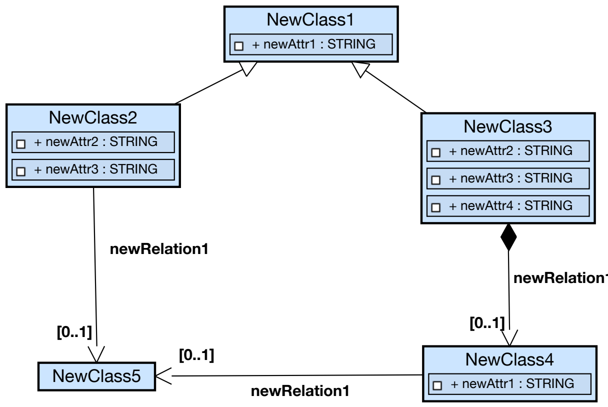
To evaluate the performance of our CNN model, we used a stratified 10 fold cross-validation approach. In this approach, we divide our entire dataset into 10 equal sized folds ensuring that each fold has same class distributions (having same proportion of samples from each class). We trained our model for 10 iterations, where in each iteration, 9 of 10 folds were used for training the model, and the remaining 1 fold was used for validation. The process enables training and validation of our model with different sets of data to ensure that our evaluation is not biased by the presence of any specific portion that may impact the evaluation results.

B. Classifier Results

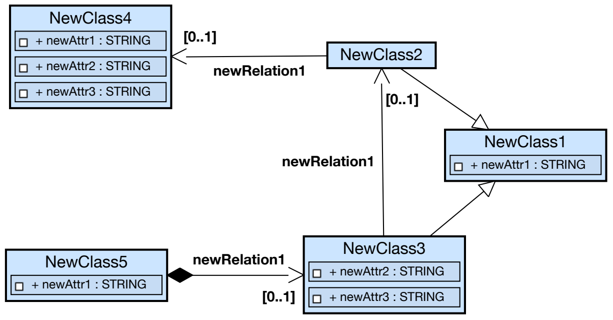
During each iteration, we trained our model with a training set and evaluated the performance of the model by using validation set. We calculated standard evaluation metrics that included precision, recall, and f1 score for each iteration. After completing all iterations, we calculated the average results of all three metrics. We obtained an average precision of 97.64%, average recall of 97.57%, and average F1-score of 97.55%. Fig. 4 shows the sum of the confusion matrices for all iterations. We observe from the figure that our model has minor issues accurately classifying reference and inheritance relations, as a significant number of reference has been classified as inheritance and vice-versa. Additionally, some inheritance relations were also misclassified as composition relations. Since there is a close similarity in sketches of each of three relations, this could have contributed to the confusion. However, the model is able to predict rectangle and attribute more accurately than the three relations. In general, the overall performance is quite strong.

C. User Evaluation

Although our model achieved an average classification accuracy of 97.57%, it was crucial to assess its actual performance



(a) Baseline Model (using Mouse and drag-and-drop)



(b) Target Model (using sketch)

Fig. 5: The baseline and target models for the user study

when participants used SkeMo to construct real models. This evaluation was necessary to determine the effectiveness of our approach (RQ1), as well as to gather insights related to the speed (RQ2) and collaboration aspects of software model creation (RQ3). We conducted a user study, which has been reviewed and approved by the Miami Research Ethics and Integrity Office under protocol ID 04242e, to evaluate our approach and collect valuable data from user experiences. We used a survey to collect data about participants' perceptions of using our tool.

We recruited users to participate in our study by sending invites to students at our institution. The subjects were undergraduate students who had completed or were completing Miami University's CSE 201 (Intro to Software Engineering) or an equivalent course. Participants were required to have a core understanding of software development, specifically knowledge of software modeling, such as UML, to be eligible for participation.

To answer RQ3, which focused on evaluating collaboration in creating software models, it was necessary to recruit pairs of participants who could work together on a collaborative task. Since participants had no prior experience with the underlying tool, we randomly paired them up based on their availability considering all participants to have equal experience. We then invited two participants at the same time to conduct the study. First, we asked each participant to individually create a model, followed by working together on a collaborative task. In the individual portion, each participant was asked to create two models using two different approaches. The first model was created with the traditional approach (using a mouse, drag-and-drop method), and the other with a sketch-based approach (creating sketches with a stylus pen/finger). The model created using the traditional approach formed the baseline model for comparison, while the SkeMo model created was the target model for the study. We provided the participants with two images of UML class diagrams and asked them to create them using the two different methods. Users were asked to create an exact model with the same classes and attributes in order and position. They were asked to create a base model first and later create the target model. Figures 5a and 5b respectively show the baseline and target models. Users were provided with a stylus pen to create the target model individually on a Microsoft Surface Pro2. For the collaboration task, users were asked to use their fingers on a large 70" touchscreen device to create the target model. Fig. 6 shows the target model for the collaborative study. All these models have the same number of model elements; however, they differ in structure, and users were asked to create each from scratch. While it is possible to construct more intricate models, the choice to utilize three simplified models for evaluation aimed to represent those seen in classrooms to gain modeling familiarity, which closely aligns with one of the objectives of IML.

We collected both qualitative and quantitative data from users during the study. Our quantitative data includes the time taken to create the models, the number of errors encountered, user ratings on their experience, and their preferences for the tool. Our qualitative data includes any comments or feedback provided by users on their experience with the tool. We collected data on time and errors during modeling tasks, and the remaining data was collected after completing the modeling tasks. We administered a survey to collect data about users' experiences, perceptions, and feedback on using our tool, which was crucial in addressing our research questions, particularly RQ1 and RQ3.

The survey contained both qualitative and quantitative questions on the topics of accuracy of conversion, intuitiveness, easy of use, usefulness, speed, overall experience, ease of collaboration, and effectiveness of collaboration. It is important to note that the survey data was intended to collect the participants' opinions and perspectives on these topics, rather than the actual evaluations, which came from the observational data. When users were asked to rate one of these criteria, we used a scale from 1 to 10, with 10 being the highest rating.

D. User Study Results

Twenty users participated in our study, and all users successfully created both the baseline and target models.

We compute the tool's accuracy in model creation by dividing the number of correct steps to successfully create the model by the total number of steps the user takes, while considering the sketch recognition error. In the event of no error, a user can successfully create the target model in a total of seventeen (17) steps, which we consider to be the correct steps. As shown in the table I, six out of twenty users successfully created the model with zero sketch recognition errors and 100% accuracy. We observed a minimum accuracy of 80.95% with the participant who produced the most sketch recognition errors. Among all participants, there was an average of 1.25 errors and an average accuracy of 93.5%. The survey question (question 1) related to the accuracy of the tool supports this, as the SkeMo was highly rated for accurately converting sketches to model components, with an average score of 9.2 out of 10. For most users, the tool accurately captured their sketches and translated them into model components. The minimum rating for the tool's accuracy in converting sketches to model components was 7, while the maximum rating was 10, and most users rated 9 out of 10.

Regarding intuitiveness, sketch-based modeling was also positively rated, with an average rating of 9.3 out of 10. The minimum intuitiveness rating was 7, while the maximum rating was 10, and most users rated 10 out of 10. Further, users found it relatively easy to create the given model using sketches as the input method. The average rating for easiness in creating the model using the sketch-based approach was 9.6, the minimum rating was 8, and most users rated 10 out of 10. Most users mentioned that they find it easy and straightforward to create the model using sketches. The overall experience with the tool, including enjoyment and perception of usefulness, was also highly rated, with an average score of 9.2 out of 10. Most users rated it a 9 out of 10, while the minimum and maximum ratings were 7 and 10, respectively. This indicates that users perceived the tool as enjoyable and useful. Additionally, all users reported that a sketch-based modeling approach is a useful tool in model creation, highlighting its potential value in the software development process. Each of these results contributes to the answer to our first research question **RQ1**; our sketch-based modeling approach to creating real-time model components is effective in terms of accuracy, intuitiveness, and ease of use.

To address **RQ2**, we collected data on the time taken by each user to complete the base and target models. From the data presented in Table I, we observed that the maximum time for creating the base model was 218s, while the maximum time for the target was 176s. Most users who took more time to create the model had problems with correctly adding an attribute to a class that inherits from the parent class, as the tool doesn't allow adding the same attribute to both the child and the parent class. Similarly, the minimum time for completing the base and target model was 75s and 56s, respectively. The average time to complete the base model was 113.8s, while the average time for the target model was 91.9s, which equates to a speedup of 19.3% on average. If we calculate the speed-up per participant and take the average of those, this equates to an average participant speed-up of 17.5%. Aside from a small number of exceptions, the data strongly suggests that creating models using SkeMo is faster than traditional means of model creation. Further, the survey results show that all users believed sketch modeling could be faster than the traditional drag-and-drop method (question

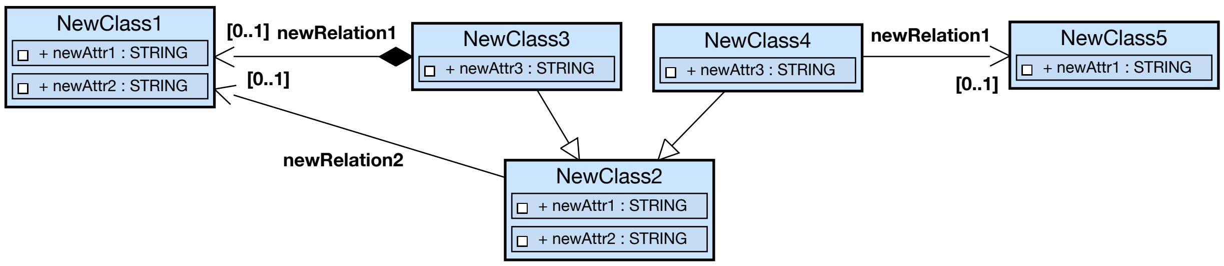


Fig. 6: Target Model for Collaborative Modeling Task

8). 95% of users preferred sketch-based modeling over the traditional method (question 9). This data provides an answer to **RQ2**; based on the time to complete required models, sketch-based modeling is more efficient than the traditional drag-and-drop method in terms of timing and user perception of efficiency in creating software models.

To address **RQ3**, a collaborative modeling study was conducted with two users using a large touchscreen monitor that supported touch gestures. The goal was to create a given target model (Fig. 6) through collaborative efforts. The time taken and errors encountered during the model creation process were recorded and analyzed. From the data, which is summarized in Table II, we observed that the maximum time taken to complete the model was 236s, while the minimum was 81s. The average time to complete the model was 155.5s, which was longer than the average time that users took to create the model individually, regardless of the creation method. The model used for collaboration was found by participants to be more complex than those in the individual study. While not the original intent, this complexity led to the necessity for collaboration in design rather than having one person create the model independently. The complexities allowed for joint problem solving, which led to longer completion times in most cases.

Furthermore, we observed that only one pair of users was able to successfully create the model with 100% accuracy, while the minimum accuracy observed was 77.27%. The average accuracy of the tool when used for collaboration on the large touchscreen monitor was reduced to 89.14%. Despite the challenges faced by users in adding correct relations and encountering more sketch recognition errors, which resulted in increased time for model completion, these aspects played a significant role in the collaborative process. Users engaged in discussions with each other and made efforts to draw sketches more accurately and add relations correctly, indicating the importance of communication and collaboration in overcoming these challenges during the modeling process. Although it may have taken longer to achieve the goal, the participants were able to work collaboratively to overcome the challenges faced.

To answer **RQ3** using specific data, users were asked to rate the ease of collaboration with another person for the creation of models using sketches on a scale of 1 to 10. The average rating for ease of collaboration was 8.4, with a minimum rating of 3 and a maximum rating of 10, with most participants rating it 10 out of 10. This suggests that on average, participants found collaborating with another person for model creation using sketches relatively easy. In the open response comments regarding the ease of collaboration in sketch-based modeling, some participants mentioned that collaborating with another person for model creation was enjoyable. Some mentioned that creating a model alone was better than collaboration. However, most users commented that they were unable to draw sketches simultaneously to create the model. Our tool did not support multi-touch drawing from two users simultaneously, which was the biggest disappointment for users in collaborative modeling. Additionally, in terms of the effectiveness of sketch-based modeling in helping with collaborative modeling, the participants provided an average rating of 8.3 for effectiveness. The minimum rating was 4, and the maximum rating was 10, while most users frequently rated 8 and 9.

Despite their criticisms, users provided an average rating of 8.4 and 8.3 for the ease and effectiveness of sketch-based modeling to support collaboration for creating software models. The overall results and feedback from participants allow us to answer **RQ3**. The proposed approach to supporting collaboration using sketch-based modeling offers room for improvement; however, the study results indicate that most participants perceived sketch-based modeling as effective in supporting collaborative model creation.

VI. Conclusion

We conclude this research by presenting our discussion of possible threats to validity, along with limitations and how both may be addressed leading into future work.

A. Threats to Validity

External Threats: In our study, students having knowledge of software development, specifically software modeling, were asked to complete a specific modeling task. Variability in modeling tasks and user experience can affect the external validity of the study. Our findings may not be generalizable to different modeling tasks or user groups with different levels of experience or backgrounds. Additionally, our focus was on real-time sketching for software modeling, and we do not claim that the results would represent other contexts or domains. Since we observe varying model performance on two different input devices (Microsoft Surface Pro vs. High-resolution touchscreen devices), another external threat to validity comes from the ability of the model to perform in varying environments, such as different resolution input

TABLE I: Individual Study Results

SN	Baseline Time (s)	Sketch Time (s)	Sketch Recognition Error	Accuracy
1	119	176	3	85.00
2	142	78	1	94.44
3	**75**	85	2	89.47
4	128	93	1	94.44
5	218	133	2	89.47
6	99	78	0	100.00
7	168	134	3	85.00
8	114	81	0	100.00
9	90	81	1	94.44
10	114	72	1	94.44
11	102	83	1	94.44
12	97	87	2	89.47
13	91	85	0	100.00
14	140	126	4	80.95
15	87	61	0	100.00
16	84	64	1	94.44
17	137	99	2	89.47
18	87	**56**	0	100.00
19	99	78	0	100.00
20	85	88	1	94.44
Average	**113.8**	**91.9**	**1.25**	**93.50**

TABLE II: Collaborative Study Results

SN	Time (s)	Sketch Recognition Error	Accuracy
1	236	3	85.00
2	145	2	89.47
3	144	1	94.44
4	214	4	80.95
5	186	4	80.95
6	104	1	94.44
7	123	1	94.44
8	110	1	94.44
9	81	0	100.00
10	212	5	77.27
Average	**155.5**	**2.2**	**89.14**

devices. This could impact the generalizability of our model's performance to real-world scenarios where users may use different devices with varying resolutions. Another threat arises from the possibility of reactivity in participants' behavior during the study, due to their awareness of being observed, which could introduce biases in the collected data.

Internal Threats: We collected varying data from six users to reduce the selection bias for preparing a labeled dataset. However, there may still be biases in selecting participants for sketch data collection, as different users follow different approaches to drawing. Therefore, the data we collected may not represent the broader population of potential users of the sketch-based modeling tool. Similarly, the quality and characteristics of the collected sketches can also introduce potential biases, as they may affect the model's ability to classify or interpret sketches accurately. Finally, since the evaluation took place using simplified models, the survey questions apply only in the limited setting, and while they may likely generalize to the other uses of SkeMo, this cannot be asserted from the data alone.

TABLE III: Summary of User Study Feedback

Question	Min	Max	Mode	Total (/200)	Average
1	7	10	9	184	9.2
3	7	10	10	186	9.3
5	8	10	10	192	9.6
7	-	-	-	-	100% Yes
8	-	-	-	-	100% Yes
9	-	-	-	-	95% Yes
10	7	10	9	184	9.2
12	3	10	10	168	8.4
14	4	10	8, 9	166	8.3

B. Limitations and Future Work

Our approach to sketch-based modeling has certain limitations that can impact its usability and accuracy. One of the limitations is the tool's inability to recognize the text to identify and update the names of the components, such as the names of classes and attributes, through sketching. While it is still possible to update via non-sketch interactions, this limitation hinders our approach's practical applicability to model creation using sketching alone. We plan to explore options including text recognition and voice dictation to better interact with model properties after initial insertion via sketching.

Another limitation is the lack of multitouch support for simultaneous sketching. This can limit the collaborative aspect of the modeling process, as multiple users may not be able to sketch or interact with the system simultaneously. This may impact the real-time collaboration and communication among team members during the modeling process, especially in scenarios where multiple stakeholders need to provide input or feedback concurrently. As this is something that wasn't really considered during our implementation and came to light via user feedback, so this is a clear candidate for immediate inclusion in the next iteration of SkeMo.

Another limitation we observed from our study is the inconsistent model's performance on devices with varying resolutions, which may affect the practical applicability and accuracy of the tool in real-world scenarios. To realize this, we could potentially expand our data collection using different devices and many participants to ensure optimal sketch recognition performance.

C. Summary

SkeMo, our approach to sketch-based modeling, involves converting user-drawn sketches into formal class diagram components in real-time, with an easy integration of modeling and sketch interfaces. We used our own trained CNN to classify user sketches and convert them into model elements. We evaluated our approach by evaluating our classifier and through a user study. The results of the user study showed that sketch-based modeling was effective in terms of accuracy, intuitiveness, and ease of use, with most users preferring it over drag-and-drop methods and finding it slightly faster. We obtained the model's average cross-validation accuracy of 97.57%, an average tool accuracy of 93.5% when creating models individually, and 89.14% for collaborative modeling. Users found real-time sketch-based modeling to be effective in supporting collaboration for model creation; however, they find it to be slightly more challenging than creating models individually, with slightly longer completion times and higher sketch recognition errors.

REFERENCES

[1] J. Walny, J. Haber, M. Dörk, J. Sillito, and S. Carpendale, "Follow that sketch: Lifecycles of diagrams and sketches in software development," in *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, 2011, pp. 1–8.

[2] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 557–566.

[3] D. Wüest and M. Glinz, "Flexible sketch-based requirements modeling," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2011, pp. 100–105.

[4] V. Goel, *Sketches of thought: A study of the role of sketching in design problem-solving and its implications for the computational theory of mind*. University of California, Berkeley, 1991.

[5] D. Wüest, N. Seyff, and M. Glinz, "Flexisketch: A mobile sketching tool for software modeling," in *International conference on mobile computing, applications, and services*. Springer, 2012, pp. 225–244.

[6] D. Wuest, N. Seyff, and M. Glinz, "Flexisketch team: Collaborative sketching and notation creation on the fly," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 685–688.

[7] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 530–541.

[8] N. DiGennaro, M. Stephan, and E. J. Rapos, "SuMo: A supportive modeling language environment for guided model transformations," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2021, pp. 566–575.

[9] T. C. Lethbridge, A. Forward, O. Badreddin, D. Brestovansky, M. Garzon, H. Aljamaan, S. Eid, A. H. Orabi, M. H. Orabi, V. Abdelzad *et al.*, "Umple: Model-driven development for open source and education," *Science of Computer Programming*, vol. 208, p. 102665, 2021.

[10] M. A. Garzón, H. Aljamaan, and T. C. Lethbridge, "Umple: A framework for model driven development of object-oriented systems," in *2015 ieee 22nd international conference on software analysis, evolution, and reengineering (saner)*. IEEE, 2015, pp. 494–498.

[11] M. Dirix, A. Muller, and V. Aranega, "Genmymodel: an online uml case tool," in *ECOOP*, 2013.

[12] T. Hammond and R. Davis, "Tahuti: A geometrical sketch recognition system for uml class diagrams," in *ACM SIGGRAPH 2006 Courses*, 2006, pp. 25–es.

[13] B. Vesin, R. Jolak, and M. R. Chaudron, "Octouml: an environment for exploratory and collaborative software design," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 7–10.

[14] Q. Chen, J. Grundy, and J. Hosking, "An e-whiteboard application to support early design-stage sketching of uml diagrams," in *IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003*. IEEE, 2003, pp. 219–226.

[15] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[16] A. T. Kabakus, "A novel sketch recognition model based on convolutional neural networks," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 2020, pp. 1–6.

[17] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Sketch-a-net: A deep neural network that beats humans," *International journal of computer vision*, vol. 122, no. 3, pp. 411–425, 2017.

[18] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?" *ACM Transactions on graphics (TOG)*, vol. 31, no. 4, pp. 1–10, 2012.

[19] L. Li, C. Zou, Y. Zheng, Q. Su, H. Fu, and C.-L. Tai, "Sketch-r2cnn: An attentive network for vector sketch recognition," *arXiv preprint arXiv:1811.08170*, 2018.

[20] F. Brieler and M. Minas, "A model-based recognition engine for sketched diagrams," *Journal of Visual Languages & Computing*, vol. 21, no. 2, pp. 81–97, 2010.

[21] B. Schäfer, M. Keuper, and H. Stuckenschmidt, "Arrow r-cnn for handwritten diagram recognition," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 24, no. 1-2, pp. 3–17, 2021.

[22] N. Mangano, A. Baker, and A. van der Hoek, "Calico: a prototype sketching tool for modeling in early design," in *Proceedings of the 2008 international workshop on Models in software engineering*, 2008, pp. 63–68.

[23] R. Jolak, B. Vesin, M. Isaksson, and M. R. Chaudron, "Towards a new generation of software design environments: Supporting the use of informal and formal notations with octouml." in *HuFaMo@ MoDELS*, 2016, pp. 3–10.

[24] B. Paulson and T. Hammond, "Paleosketch: accurate primitive sketch recognition and beautification," in *Proceedings of the 13th international conference on Intelligent user interfaces*, 2008, pp. 1–10.

[25] E. J. Rapos and M. Stephan, "Iml: Towards an instructional modeling language." in *MODELSWARD*, 2019, pp. 417–425.

[26] V. Thambawita, I. Strümke, S. A. Hicks, P. Halvorsen, S. Parasa, and M. A. Riegler, "Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images," *Diagnostics*, vol. 11, no. 12, p. 2183, 2021.

[27] A. Sharma Chapai and E. J. Rapos, "SkeMo source and evaluation data," April 2023. [Online]. Available: https://osf.io/s35gh/?view_only=51f6d5f581ad43528bbd548067a4fa08