Developing And Validating A Machine Learning-Based Student Attentiveness Tracking System

Andrew L. Sanders

DEVELOPING AND VALIDATING A MACHINE LEARNING-BASED STUDENT ATTENTIVENESS TRACKING SYSTEM

by

ANDREW SANDERS

(Under the Direction of Andrew A. Allen and Gursimran S. Walia)

ABSTRACT

Academic instructors and institutions desire the ability to accurately and autonomously measure the attentiveness of students in the classroom. Generally, college departments use unreliable direct communication from students (i.e. emails, phone calls), distracting and Hawthorne effect-inducing observational sit-ins, and end-of-semester surveys to collect feedback regarding their courses. Each of these methods of collecting feedback is useful but does not provide automatic feedback regarding the pace and direction of lectures. Young et al. discuss that attention levels during passive classroom lectures generally drop after about ten to thirty minutes and can be restored to normal levels with regular breaks, novel activities, mini-lectures, case studies, or videos (Young et al., 2009). The tracking of these "drops" in attention can be crucial for accurate timing of these change-ups in activities. This allows for maximal attention and a greater amount of deeply learned material. Autonomously collected data can be also used either real-time or post-hoc to be able to alter the design and presentation of lectures. Being able to keep track of student attention is vital to be able to have confidence in the delivery of material. Even if lectures do not break up presentation slides with attention-raising activities, they can still show more important information during periods of high attention and less important information during periods of low attention. This area of research has applications both in in-person classrooms and in online learning environments, which is especially relevant now during the COVID-19 pandemic. For large in-person classrooms, or classrooms where students' faces are obscured, such as behind computer monitors, this research area could prove invaluable.

INDEX WORDS: Attention tracking, Machine learning, Convolutional neural network, Eye tracking

DEVELOPING AND VALIDATING A MACHINE LEARNING-BASED STUDENT ATTENTIVENESS TRACKING SYSTEM

by

ANDREW SANDERS

B.S., Georgia Southern University, 2020

A Thesis Submitted to the Graduate Faculty of Georgia Southern University

in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

DEVELOPING AND VALIDATING A MACHINE LEARNING-BASED STUDENT ATTENTIVENESS TRACKING SYSTEM

by

ANDREW SANDERS

Major Professor: Andrew Allen and Gursimran Singh Walia
Committee: Lixin Li

Electronic Version Approved:
May 2022

DEDICATION

I dedicate this to my family, who encouraged me throughout my journey, and to Ariel Harper, the love of my life and the person who kept me sane through the all-nighters.

ACKNOWLEDGMENTS

I want to acknowledge Dr. Gursimran Walia and Dr. Andrew Allen for being great advisors throughout my Masters. Their advice kept me on track in periods of stress so I can't thank them enough. I want to acknowledge Dr. Lixin Li for being gracious enough to be a part of my thesis committee. Bradley Boswell and Md Shakil Hossain were both great helps when it came to research and development of the NiCATS tool. I want to acknowledge Jacob Carter, Austin Towler, and Chris Iverson for their help in the initial creation of the Engagement Tracker back in our Software Engineering project that eventually led to NiCATS.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Instructors know that effective instruction involves maintaining student attention. Experienced instructors use proxies such as facial expressions (Whitehill et al., 2014) and body posture to aid in assessing student attentiveness. The earlier an instructor can become aware of the students' attentiveness level, the earlier remedial actions can be applied. However, the use of these proxies is challenging in some environments, this is especially true in Computer Science or other lab settings where computer monitors are obscuring student's faces especially in large classes. Research (Young et al., 2009) has also shown that attention levels in students generally decrease in learning environments, such as lectures, after about 10-30 minutes. Understanding when individual students or the class as a whole are approaching that critical point in their attention levels would help the instructor to identify optimal points to introduce tasks. While active learning techniques such as task-related activities can be used to break up a lecture to actively engage students (Young et al., 2009), the ability to accurately track the level of students' attention remains an open research problem. For this work, attentiveness is defined as the short-term application of the mind towards a topic, while engagement is defined as the overall emotional commitment of a person's mind towards a topic. Though they are defined slightly differently, they are tracked and treated similarly in research designed to visually assess students' perceptible facial expressions as a proxy for the underlying feelings and attitudes a student holds, so a system designed to automatically perceive engagement is desirable.

Significant research has been done in combining machine learning and computer vision to automatically determine the attentiveness of a student (Tabassum et al., 2020)(Whitehill et al., 2014). Other researchers have shown that the location and general metrics (for example, average fixation duration) of a person's eyes generally reflect the intent of the person's mind (Sharafi et al., 2015)(Veliyath et al., 2019). While facial expressions give insights on students' perceived attentiveness from the perspective of the observer, eye tracking metrics can add another layer of insights concerning the students’ eyes searching and acquiring content. The assertion is that it would be beneficial for educators if they can track the attentiveness of students based on their facial image combined with selected eye metrics data. One approach to tackling theseis challenges is automation to accurately track and provide feedback on the attention levels of a classroom or lab in real-time. Through the collection and analysis of proxies for attentiveness, feedback can be provided to the instruction as an instruction aid to improve the classroom experience and student success. This automated feedback can not only guide the lecture delivery but also aid in instructional design as well. The focus of this research, therefore, is to investigate the visual proxies for attentiveness and design an approach for measuring and reporting the perceived attentiveness of students in a classroom or lab, as well as evaluating the effectiveness of the proposed approach. The advantage of using this approach is the non-intrusive nature or significantly lowered observational effect due to the passive nature of the system while having the similar attention-judging accuracy as domain expert humans. The passive nature of the system is aptly important as it can be applied to online classes that are ubiquitous as a result of the COVID-19 pandemic.

The dataset is generated from four different experimental settings, where their facial images (which are labeled for attentiveness), eye gaze data (which are used for correlation analysis with the labeled facial images), and screenshots (which are used to generate heatmaps for use in comprehension analysis) were recorded. Using the eye gaze and facial image data, attentiveness scores are calculated and evaluated for precision, recall and accuracy.

In this research, a Non-intrusive Classroom Attention Tracking System (NiCATS) is proposed and presented. NiCATS currently provides post-hoc log-file analysis (with real-time analysis in the future) on the overall level of attention the students in a classroom are displaying. To validate the usefulness of the NiCATS system, four controlled experiments were conducted where non-invasive monitor-mounted eye trackers and webcams were used to collect information about students while they attended the lecture. The facial images of subjects were collected and labeled along with collecting and calculating their eye metric data then compared them to check for underlying correlations. The results showed significant correlations between eye metrics and perceived student attentiveness and provided useful insights on using NiCATS to understand knowledge acquisition patterns of students.

CHAPTER 2

LITERATURE REVIEW

While this paper will use the word "attention", it is important to note that attention is an ill-defined and ambiguous term that maps onto multiple meanings (Anderson, 2021). Even from a hundred years ago, there has been a debate on whether attention should be considered a cause or a result ("In short, voluntary and involuntary attention may be essentially the same") (James, 1910). This paper will not be attempting to resolve this issue, but instead will use a specific definition that should clarify its use.

The definition of attention in this paper is similar to a psychological consequence. Attention, in this paper, refers to "the effect of events on the mental or emotional state of individuals resulting in a change in perception", where "events" is referring to classroom instruction, and "change in perception" refers to the application of the mind towards a topic. A few cited works refer to "engagement" instead of attention, but they will be treated synonymously due to the similarity in measuring methods. Engagement can be defined as "the overall emotional commitment of a person's mind towards a subject" and can be thought of as having a longer-term application, as opposed to the shorter-term application of attention. Some literature also breaks down engagement into "behavioral", "cognitive", and "emotional" engagement, but this paper will not be covering it (Fredricks et al., 2004).

Dewan et al. discuss attention detection methods and state they can be generally classified within three categories — automatic, semi-automatic, and manual (based on strategy and type of users' involvement) — and each work cited uses one or more to measure attentiveness (Dewan et al., 2019) (Figure 1). Automatic can be broken down into computer vision-based methods (facial expressions, gestures, postures, and eye movements), sensor data analysis, and log file analysis. Semi-automatic methods include engagement tracing, which utilizes the timing and accuracy of student responses for measuring attentiveness. Manual attention detection methods include observational checklists and self-reporting. Each of these methods has its limitations. Automatic methods need more work to become practical in a normal education setting. Semi-automatic methods require probabilistic inference and there are not many applications to build off of. Manual methods require significantly more time and effort from both the observers and students and are more susceptible to bias and the Hawthorne effect due to the presence of the self-reporting or the observer.

Efforts have been made to create more objective criteria for measuring engagement in a classroom for use in each detection method. Lane et al. developed the Behavioral Engagement Related to Instruction protocol (Figure 2) to be used to identify student behavioral engagement levels at any given time (Lane et al., 2015). The protocol gives descriptions of student behaviors that can be classified as engaged and disengaged. One of the major benefits of this protocol is that the researchers found that it has excellent interrater agreement across multiple courses with different instructors and pedagogy. The work done by Sanders et al. (our own work) and Tabassum et al. along with other researchers have used this protocol (Tabassum et al., 2020)(Sanders et al., 2021)(mentioned below).

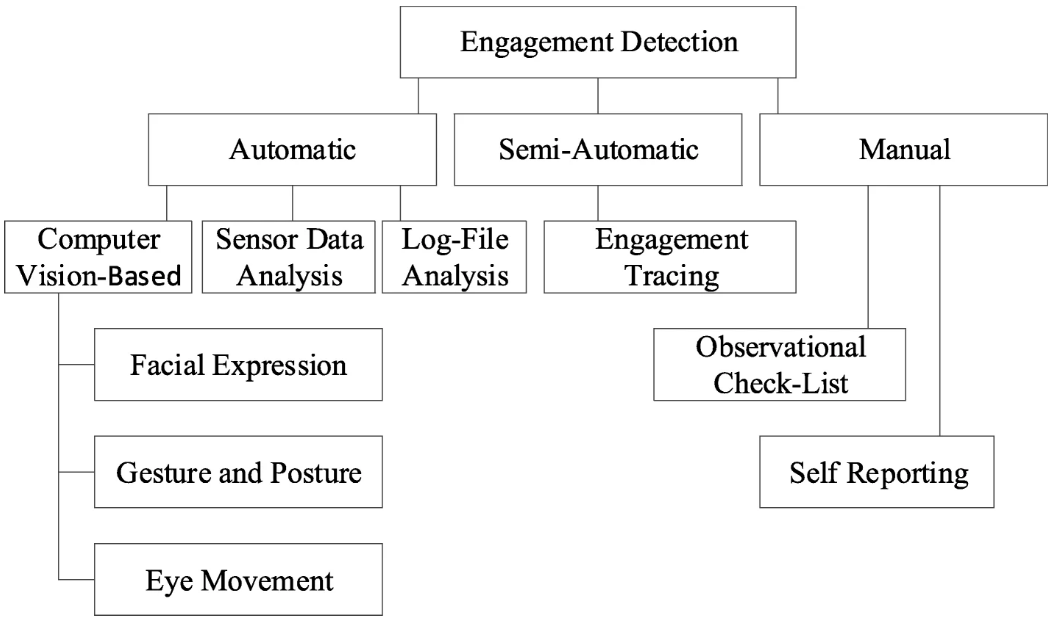


Figure 1. Taxonomy Of Learners' Engagement Detection Methods (Dewan et al., 2019)

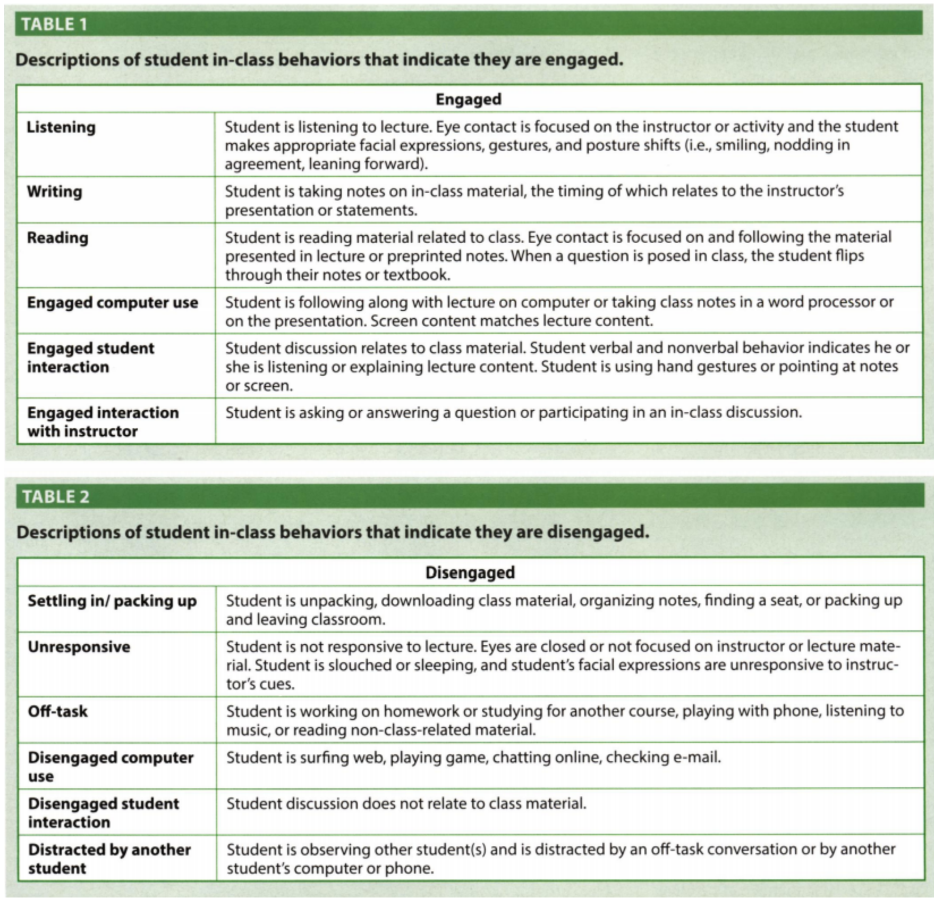


TABLE 1

Descriptions of student in-class behaviors that indicate they are engaged.

Engaged	
Listening	Student is listening to lecture. Eye contact is focused on the instructor or activity and the student makes appropriate facial expressions, gestures, and posture shifts (i.e., smiling, nodding in agreement, leaning forward).
Writing	Student is taking notes on in-class material, the timing of which relates to the instructor's presentation or statements.
Reading	Student is reading material related to class. Eye contact is focused on and following the material presented in lecture or preprinted notes. When a question is posed in class, the student flips through their notes or textbook.
Engaged computer use	Student is following along with lecture on computer or taking class notes in a word processor or on the presentation. Screen content matches lecture content.
Engaged student interaction	Student discussion relates to class material. Student verbal and nonverbal behavior indicates he or she is listening or explaining lecture content. Student is using hand gestures or pointing at notes or screen.
Engaged interaction with instructor	Student is asking or answering a question or participating in an in-class discussion.

TABLE 2

Descriptions of student in-class behaviors that indicate they are disengaged.

Disengaged	
Settling in/ packing up	Student is unpacking, downloading class material, organizing notes, finding a seat, or packing up and leaving classroom.
Unresponsive	Student is not responsive to lecture. Eyes are closed or not focused on instructor or lecture material. Student is slouched or sleeping, and student's facial expressions are unresponsive to instructor's cues.
Off-task	Student is working on homework or studying for another course, playing with phone, listening to music, or reading non-class-related material.
Disengaged computer use	Student is surfing web, playing game, chatting online, checking e-mail.
Disengaged student interaction	Student discussion does not relate to class material.
Distracted by another student	Student is observing other student(s) and is distracted by an off-task conversation or by another student's computer or phone.

Figure 2. Behavioral Engagement Related To Instruction Protocol (Lane et al., 2015)

2.1 COMPUTER VISION WITH FACIAL EXPRESSIONS

Computer Vision, in combination with cameras, has been used to measure the facial expressions of students in classrooms and its relationship to attentiveness.

Whitehill et al. used students' facial expressions to train a machine learning model to predict engagement levels (Whitehill et al., 2014). They collected data in the form of video recordings of the subject's face from 34 undergraduate students in a "Cognitive Skills Training" experiment. In the experiment, they collected video and synchronized task performance data. They used the pre-and post-test performance on one of the games as the dependent variables.

Using a team of labelers, the video clips (with audio muted) were labeled for the perceived appearance of engagement from 1 (being least engaged) to 4 (being most engaged). Instead of following pre-established criteria for identifying engagement, they defined their own criteria. “1” meant the student was looking away from the computer and obviously not thinking about the tasks and/or their eyes were completely closed. “2” meant the student’s eyes were barely open or they were clearly not invested with the task. “3” meant the student required no warning to stay on task. “4” meant the student was clearly focused on the task and had a high level of engagement. As with all facial expressions-based labeling tasks, the labelers were instructed to label the video clips on how engaged they perceived the student to be.

Using the labeled data, they created a machine learning-based automatic engagement detector. They found that using binary classification, the automatic engagement detector had similar accuracy to humans. They found that both human labels and automatic engagement labels correlated decently with task performance.

One of the unique findings of this paper was that static images contain the bulk of the information about the appearance of engagement. When comparing individual frames with the video clips they originated from, there was a Pearson correlation of r=0.85. This was a useful finding because it could help ease the logistical overhead of future research.

Yun et al. used a pre-trained convolutional neural network (CNN) and transfer learning to create a model that is useful for recognizing engagement levels (Yun et al., 2020). Because of a childrens’ face images training set shortage, they used VGG Face pre-trained networks and modified the models for the recognition of children engagement. They proposed an automatic children engagement recognition method based on CNNs for the future.

2.2 EYE TRACKING

Researchers have used eye trackers to predict attentiveness. It is a fast-growing (Figure 3) research field and it has applications in the measurement of cognitive properties.

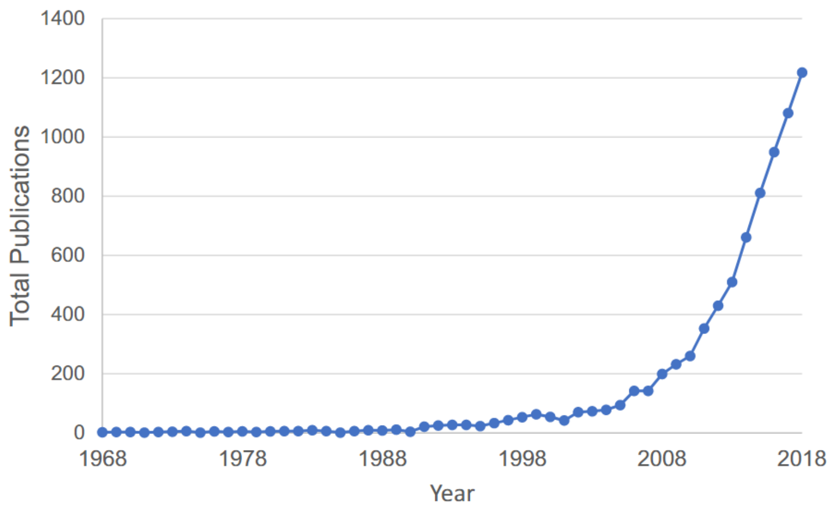


Figure 3. Total Number Of Publications Per Year From 1968 To 2018 Using The Search Term "Eye tracking" Or "eye-tracking" Or "eyetracking" In Web Of Science (Carter et al., 2020)

Research commonly uses the terms that follow. Eye gaze data is the immediate direction of a person's eyes and can be represented in xy coordinates when looking at a computer monitor. Fixations are the stabilization of the eye on part of a stimulus for a period of time and are usually around 200-300ms (Sharafi et al., 2015). Saccades are the quick and continuous eye movement between fixations and are usually around 50ms (Sharafi et al., 2015). For each of these, generally, an eye tracker is used. An eye tracker can be eye-attached (like a contact lens), optical (reflected infrared), or electric potential (electrodes placed around the eyes). General consumer-grade eye trackers are optical tracking, with some being head-mounted and some being computer monitor-mounted.

Veliyath et al. used non-intrusive Tobii 4c eye trackers with self-reporting to predict attentiveness during class lectures (Veliyath et al., 2019). For their data collection, they mounted eye trackers on the monitors of a computer lab and had students periodically (every five minutes) answer how engaged they believed their lecture to be on a Likert Scale from 1 (not very engaging) to 10 (very engaging) while collecting their eye gaze points. Using this data, they trained multiple machine learning models to be able to predict attention. They had a peak accuracy of 77% with an Extreme Gradient Boosting classifier, as compared to 52% of a baseline model. They concluded that gaze data can work with other features to achieve a higher level of accuracy.

Rosengrant et al. used Tobii Glasses to track student eye movements during a lecture and determine the causes of inattention (Rosengrant et al., 2012). They used eight volunteers and recorded what they were looking at during a physical science lecture. They found that students rarely focus on the actual professor except when the instructor was moving with emotion, drawing something, being humorous, or using analogies that were not on the presentation slides. New slides tended to keep student interest or divert it to the board. They usually looked at pictures or diagrams before looking at text. The researchers also found that students with printed-out notes before class tended to pay less attention during class. Phones or students entering or leaving the room also caused disruptions in attention. The position of the students in the room also affected how often they didn't pay attention (where students in the last row or on the side tended to be more easily distracted).

2.3 FULL BODY MOTION SENSORS

Researchers have used full-body motion sensors to detect attentiveness. These are devices that are designed to track the human body in 2D and 3D. This can provide valuable information in facial and body behaviors.

Zaletelj et al. used the 2D and 3D data obtained by a Kinect One sensor to build a feature set characterizing facial and body properties of students to train machine learning models that predict attentiveness (Zaletelj et al., 2017). They used human observers' estimation of attention levels as ground truth to train against. The human observers estimated the observed attention levels of the students during the lecture from one to five and then calculated the mean of the scores every second.

They tested seven classifiers for attention estimation from Kinect features: Decision tree with 4 splits, decision tree with 20 splits, K-nearest neighbors with 100 neighbors and equal distance weight, K-nearest neighbors with 10 neighbors, Bag of decision trees with 30 learners, and 20 as the max number of splits, and Subspace K-NN. They validated their automatic attention prediction model by using a dataset of 18 people and had moderate accuracy of 0.753. They concluded that there is a possibility for using full-body motion sensors for affordable tracking of attention, which would be helpful for evaluating lectures.

2.4 OTHER MONITORING METHODS

Researchers have used other methods for measuring attentiveness. This includes wearable computers and sensors, along with heartbeat monitoring.These attentiveness measuring methods are less conventional but still provide valuable information on the human body.

Zhu et al. used the sensors on wearable computers to track both hand motions and heart activity to predict "interest level" and "perception of difficulty" (Zhu et al., 2017). They asked 30 volunteers to wear Moto 360 smartwatches during two lectures with 14 topic periods. They collected motion data at 25 Hz and PPG data at 12.5Hz and surveyed the students for their opinions on their interest level and perception of difficulty towards each topic covered in the lectures. They created classification machine learning models from this data and found that decision trees and support vector machines provide the best performance. They had an accuracy of 98.99% for interest and 95.79% for perception of difficulty. The researchers concluded that using wrist-worn smartwatches provides excellent accuracy in attention detection and that leveraging other physiological sensors could potentially improve the accuracy.

2.5 COMBINING MONITORING METHODS

Some researchers have combined previous methods to produce a more accurate attention detection system. The benefit of this is that the accuracies of different methods can complement each other and provide a more holistic approach to attention detection.

Tabassum et al. used a deep learning convolution neural network and the outputs from a cloud-based emotion recognition service to find correlations between emotions and attentiveness (Tabassum et al., 2020). They collected data by having the webcam images of students be recorded while they attend their class lecture. The videos were run through a motion detection algorithm to extract faces whenever motion was detected. The face images were labeled as either attentive or inattentive based on the BERI protocol. A convolution neural network was built using these label face images. They then used the Amazon Rekognition service to get the facial emotions of the images. The emotions were given a value between 0 and 100 and the possible emotions were happy, sad, angry, confused, disgusted, fear, surprised, and calm. The researchers then applied correlation analysis and regression analysis between the outputs from the CNN model and each emotion that was outputted by Amazon Rekognition for an image. Each emotion was found to be statistically significant in regression analysis between the emotion and attentiveness, showing it is possible to use facial emotions to improve the accuracy of attention detection models. They had an accuracy of 93% when detecting attentive students in a classroom.

CHAPTER 3

PROPOSED APPROACH AND ITS IMPLEMENTATION

The long-term goal of the Non-Intrusive Attentiveness Tracking System (NiCATS) is to provide instructors with real-time feedback on the attentiveness of students in their classroom. As a first step, this thesis explores different aspects of students' attentiveness (e.g., facial images, eye movements) to understand how these inputs can be used to train a machine learning model (inbuilt in NiCATS) for predicting the measure of student attentiveness. To provide an overview, NiCATS utilizes webcams (for facial attentiveness) and eye trackers (for patterns related to cognitive activities) that can be mounted on student machines. NiCATS collect webcam images, gaze points of their eye movements, and screenshots (of their screen) that can be analyzed for understanding student attentiveness. While other researchers have primarily focused on perceived facial attentiveness, this work tried to utilize student attentiveness informed by their facial expressions and statistics of their gaze patterns in real-time.

To that end, Figure 4 shows a high-level overview of NiCATS in terms of data collection, preprocessing, and analysis (discussed in subsequent sections). While NiCATS can be used post-hoc (as shown in Experiment 1), the novelty of this work comes from the ability for instructors and institutions to use NiCATS in real-time and use it to validate interventions (or changes to instruction material or pedagogy) based on the NiCATS output (student attentiveness).

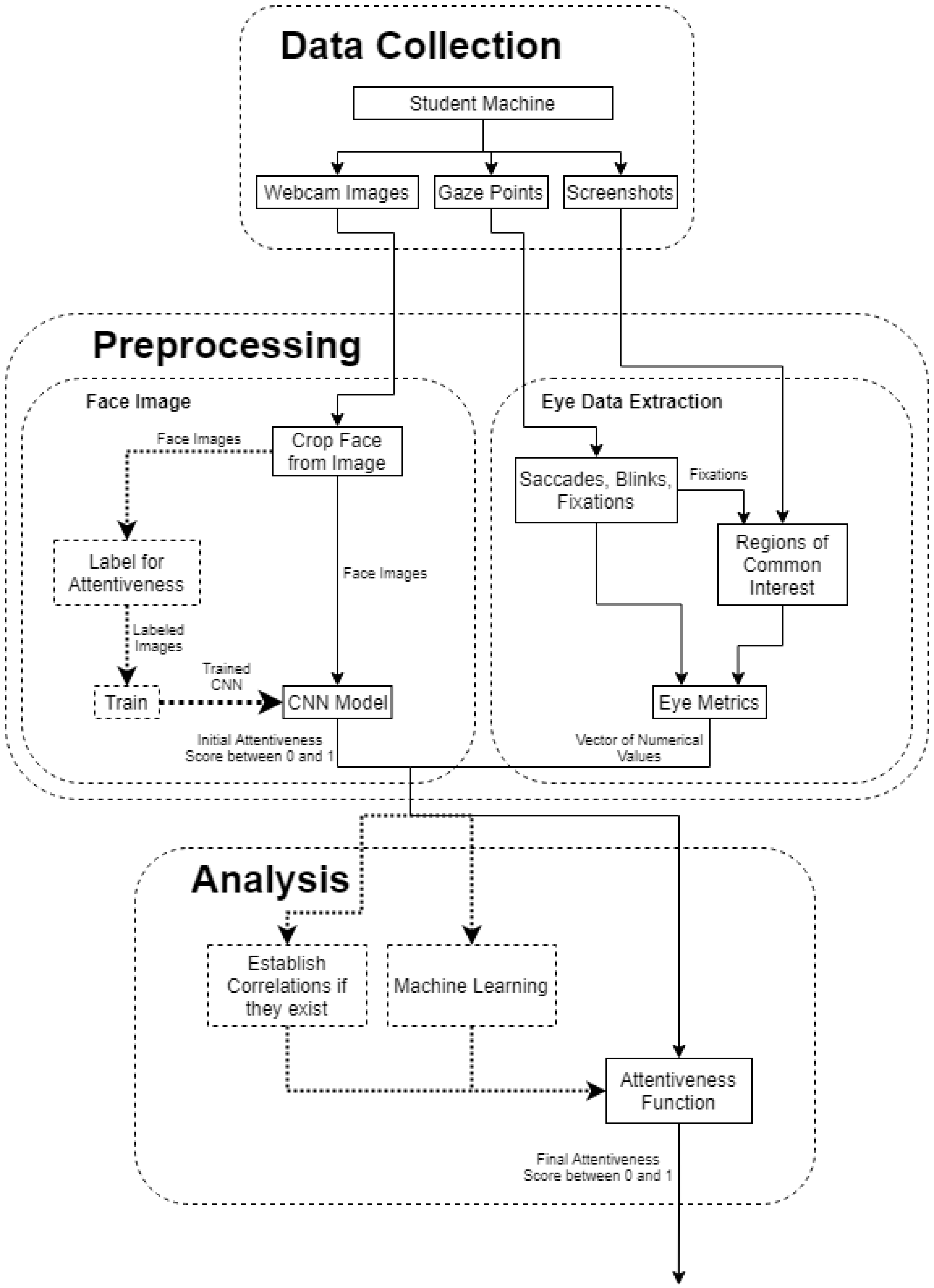


Figure 4. High-Level Design Of NiCATS

3.1 DATA COLLECTION

The NiCATS system is set up to capture three distinct data types about the student from the student's machine.

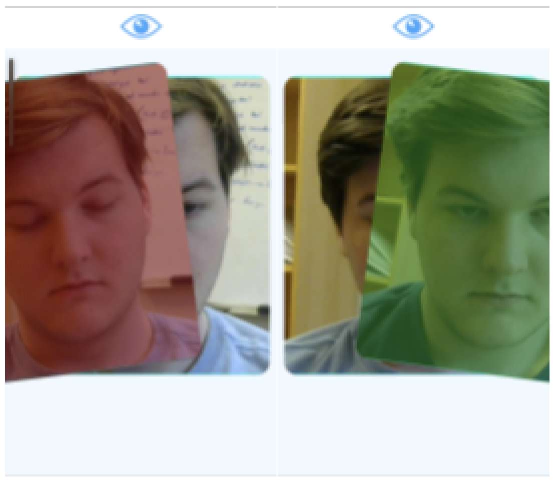
- Facial image - A computer monitor-mounted webcam periodically captures and sends images of the student's face to the server at a 5-second interval. A continuous video stream of the student's face could be collected, but this provides a marginal benefit compared to static images when determining attentiveness (Dewan et al., 2019).
- Screen Capture - A screenshot capture will be triggered whenever the student interacts with their machine via keyboard/mouse input or, in the case where no input was received after the pre-defined time interval of 15 seconds.
- Eye movement - The gaze points of the student, with regard to their computer monitor, are captured continuously throughout the lecture. Anytime a screenshot is prepared to be sent to the server, the most recent chunk of gaze points since the last screenshot was transmitted to be sent along with the screenshot for pre-processing.

3.2 PRE-PROCESSING

The preprocessing for each data item collected (facial images, eye movements, and screenshots) is explained in the following subsections. Each preprocessing step describes design decisions (e.g., how to best label images, how to create regions of interest) to be able to use the system in real-time and analyze the collected data in a post-hoc manner.

1. Face image: To capture isolated face images of the student for labeling, a smaller image is cropped from the original, which contains only the student's face, by using Haar cascade classifiers (James, 1910). To generate a set of attentive and inattentive images for comparison with the extracted eye metrics, multiple labelers were asked to label the face images based on the validated Behavioral Engagement Related to Instruction (BERI)

protocol (Fredricks et al., 2004). The human labeling was handled via the NiCATS mobile web application which allows human-labelers to swipe images right or left on their mobile devices to label images as being attentive or inattentive respectively (Figure 5). From the labeled image set, the attentiveness score was arrived at by summing all "attentive" labels on an image and dividing it by the total number of labels (attentive or inattentive) a face image receives.



(Swipe Left - Inattentive) (Swipe Right - Attentive)

Figure 5. Attentiveness Labeling Mobile App

2. Eye Data Extraction: The gaze points are pre-processed to extract relevant eye metrics (e.g., fixation, saccades) that can be used to predict student attentiveness and comprehension. Using the gaze points, fixations (the stabilization of the eye on the part of a stimulus for a period of time (200-300 ms) (Anderson, 2021)) and saccades (the quick and continuous eye movements within 40-50 ms from one fixation to another (Anderson, 2021)) are calculated. A subset of eye metrics (relevant to this work) that can be collected from fixation and saccade calculations are listed below:

- Fixation Count (total # of fixations). This can be collected for the entire lecture period or for specific lengths of time.
- Average Fixation Duration: This is measured by adding the durations of all fixations divided by the number of fixations.
- Number of fixations per second: Total number of fixations divided by the total duration of a recording session.
- Saccades Count (total # of saccades): This can be collected for the entire lecture period or for certain lengths of time.
- Average Saccade Duration: This is measured by adding the durations of all the saccades divided by the number of saccades.
- Saccades per second: Total number of saccades divided by the total duration of a recording session.

The collection of eye metrics will then be compared with the results of the human-labeled face images to determine if any correlations exist between the eye metrics and a student's attentiveness level for that interval of time.

3.3 CONVOLUTIONAL NEURAL NETWORK

The convolutional neural network (CNN) is being utilized in this research to predict student attentiveness based on their facial images alone. The CNN model is being trained on facial images that were manually labeled according to the Behavioral Engagement Related to Instruction protocol (Lane et al., 2015) and then results produced by CNN model (on student attentiveness) are validated against the ground truth (manual labeling of student attentiveness). Additionally, the output of CNN model is also being used to understand correlations with eye metrics to better understand the independent variables that can improve the performance of CNN model in future.

3.3.1 CONVOLUTIONAL NEURAL NETWORK MODEL ARCHITECTURE

The CNN model was trained on a set of face images that were labeled on perceived facial attentiveness. The dataset consisted of 630 equally split images (315 images labeled inattentive, 315 images labeled attentive). The original dataset contained 2,122 images but it was not evenly split. It was shrunk down using random sampling so the dataset could be balanced. Each image was 350 pixels wide, 350 pixels tall, and was RGB.

The CNN model consisted of a number of sequential layers. The layers consisted of 2D Convolution, Activation, 2D Max Pooling, Dropout, Flatten, and Dense layers.

- 2D Convolution: Convolution layers are used to extract features from the input. It consists of element-wise multiplication and addition using a kernel. The kernel "slides" over the input data, performing an element-wise multiplication.
- Activation: The activation layers apply activation functions to an output. The purpose of the activation function is to help the CNN learn patterns in the data. In this model, the activation function after each 2D convolution layer and the second-to-last dense layer is ReLU (Rectified Linear Unit), which applies max(0,x) to every output (where x is the output value). ReLU has the benefit of reducing the chance that the gradient vanishes as x increases and is less computationally expensive than other activation functions as it only needs to compute the max between 0 and x. The activation function for the final dense layer is sigmoid, which applies 1 / (1 + exp(-x) to the final dense layer output (where x is the dense layer output value). Sigmoid is useful for models that predict the probability of the class as an output.
- 2D Max Pooling: The goal of max pooling is to downsample the input along its height and width. It accomplishes this by taking the maximum value over an input window (2x2

in this model). Pooling layers are used to reduce the dimensions of the input so that the number of parameters to train and the amount of time needed to train is reduced.

- Dropout: Dropout layers are masks used to nullify the contribution of some neurons from the previous layer towards the next layer. The purpose of the dropout layer is to prevent overfitting on training data.
- Flatten: Flatten layers transform the input layer into a 1-dimensional fully-connected output. They are generally used after all of the convolutional layers so that the data can make connections to the final layer.
- Dense: Dense layers are fully-connected layers that are used in the final stages of the neural network. Each neuron in the dense layer is fully connected to each neuron in the previous layer.
- Loss Function: The loss function is what is used to calculate the gradients, which are used to update the weights of the convolutional neural network. Because the model needs to predict between two classes (attentive or inattentive), the loss function is binary crossentropy.
- Optimizer: Optimizers are used to alter the weights and learning rate of a CNN model in order to minimize the loss (from the loss function). In this case, the optimizer being used is AdaDelta. AdaDelta is a stochastic gradient descent optimization that is based on adaptive learning rate per dimension. The purpose of using AdaDelta in this research is that it continues learning even after many updates have been done, which is useful for continued training after the collection of data.

Figure 6 provides a visual representation of the model and its layers.

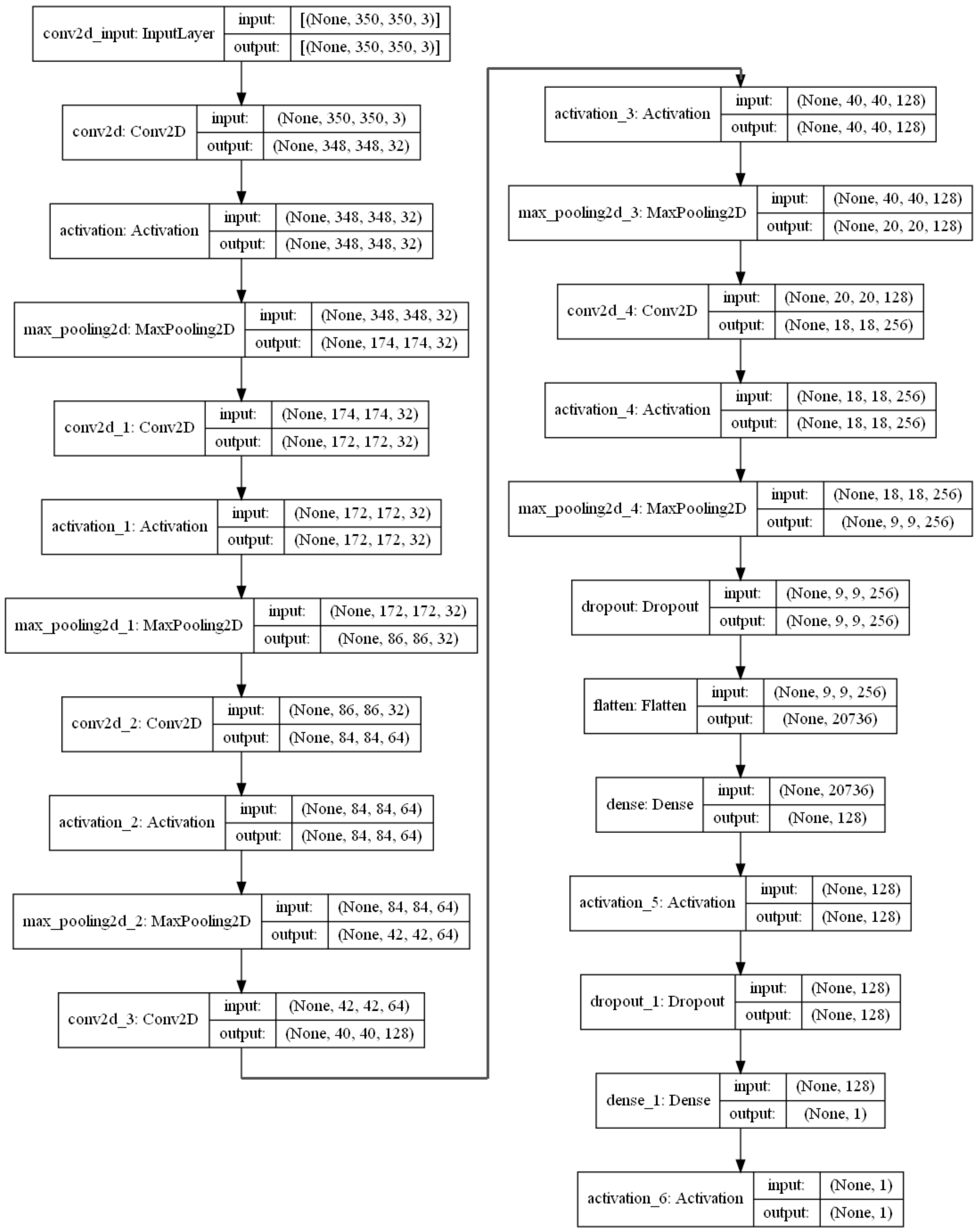

conv2d_input: InputLayer | input: [(None, 350, 350, 3)] | output: [(None, 350, 350, 3)]
conv2d: Conv2D | input: (None, 350, 350, 3) | output: (None, 348, 348, 32)
activation: Activation | input: (None, 348, 348, 32) | output: (None, 348, 348, 32)
max_pooling2d: MaxPooling2D | input: (None, 348, 348, 32) | output: (None, 174, 174, 32)
conv2d_1: Conv2D | input: (None, 174, 174, 32) | output: (None, 172, 172, 32)
activation_1: Activation | input: (None, 172, 172, 32) | output: (None, 172, 172, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 172, 172, 32) | output: (None, 86, 86, 32)
conv2d_2: Conv2D | input: (None, 86, 86, 32) | output: (None, 84, 84, 64)
activation_2: Activation | input: (None, 84, 84, 64) | output: (None, 84, 84, 64)
max_pooling2d_2: MaxPooling2D | input: (None, 84, 84, 64) | output: (None, 42, 42, 64)
conv2d_3: Conv2D | input: (None, 42, 42, 64) | output: (None, 40, 40, 128)
activation_3: Activation | input: (None, 40, 40, 128) | output: (None, 40, 40, 128)
max_pooling2d_3: MaxPooling2D | input: (None, 40, 40, 128) | output: (None, 20, 20, 128)
conv2d_4: Conv2D | input: (None, 20, 20, 128) | output: (None, 18, 18, 256)
activation_4: Activation | input: (None, 18, 18, 256) | output: (None, 18, 18, 256)
max_pooling2d_4: MaxPooling2D | input: (None, 18, 18, 256) | output: (None, 9, 9, 256)
dropout: Dropout | input: (None, 9, 9, 256) | output: (None, 9, 9, 256)
flatten: Flatten | input: (None, 9, 9, 256) | output: (None, 20736)
dense: Dense | input: (None, 20736) | output: (None, 128)
activation_5: Activation | input: (None, 128) | output: (None, 128)
dropout_1: Dropout | input: (None, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 1)
activation_6: Activation | input: (None, 1) | output: (None, 1)

Figure 6. Visualization Of The Cnn Model

3.3.2 MODEL TRAINING AND TESTING

The machine learning model was trained on the images from experiment 3. The images were labeled for attentiveness (attentive or inattentive) by three labelers following the BERI protocol. For the purposes of reliability, images were only considered for training if all three labelers labeled the image with the same label. Doing this, the number of labeled images went from 2,122 to 1,662. Of the 1,662 images, only 315 were labeled as inattentive. Because a balanced dataset was needed to train the model, all 315 labeled inattentive images were included in the dataset while 315 labeled attentive images were randomly chosen to be included in the dataset.

To train the CNN model, the dataset was chunked in batches and iterated on for a number of epochs. Because of the size of the images (350x350x3), the batch size was 2. Because of the relatively low number of images, the number of epochs was 50. To compensate for the number of images, an image data generator was used to help with the generalizability of the model. The generator generated batches of tensor image data with randomly applied shearing transformations and randomly applied zooming inside the images. The number of steps per epoch was the number of samples divided by the batch size of 2, meaning every image was iterated upon while training. The data was split 80:20 training:testing. 252 attentive images and 252 inattentive images were used during the training and 63 attentive images and 63 inattentive images during the testing of CNN model as shown in Tables 1 and 2 respectively.

Multiple evaluation metrics are in the following sections. The model classification report is shown in Table 1 and Table 2. Key terms of the classification report are as follows:

- Accuracy: The accuracy of the model indicates the ratio of time the model correctly labels the image. It is defined as (true positive + true negative) / (true positive + true negative + false positive + false negative). These terms are defined in 3.3.3.

- Precision: The precision of the model indicates the ratio of time that a label is correctly given over the total predictions of that label. It is defined as (true positive) / (true positive + false positive).
- Recall: The recall of the model indicates the ratio of time that a label is correctly given over the total number of images that actually have that label. It is defined as (true positive) / (true positive + false negative).
- F1 Score: The F1 score indicates the equilibrium between precision and recall. It is defined as (2*precision*recall)/(precision + recall).

	Accuracy	**precision**	**recall**	**f1-score**
Attentive	0.52	0.52	0.67	0.58
Inattentive		0.53	0.38	0.44

Table 1. Model Classification Report For Training Set

	Accuracy	**precision**	**recall**	**f1-score**
Attentive	0.52	0.51	0.67	0.58
Inattentive		0.52	0.37	0.43

Table 2. Model Classification Report For Testing Set

Based on the results presented in Tables 1 and 2, the performance of the machine learning model is better at identifying attentive students as compared to identifying inattentive students correctly. More details on the false positives and false negatives are discussed in the following subparagraphs.

3.3.3 CONFUSION MATRIX

A confusion matrix allows for visualization of the performance of a classification machine learning model. It allows for insight into the types of errors the model is making. Importantly, it shows the true positives, true negatives, false positives, and false negatives.

- True Positive: A true positive indicates that the model correctly predicts the positive label. In this case, the model predicts the image being attentive and the label is attentive.
- True Negative: A true negative indicates that the model correctly predicts the negative label. In this case, the model predicts the image being inattentive and the label is inattentive.
- False Positive: A false positive indicates that the model incorrectly predicts the positive label. In this case, the model predicts the image being attentive and the label is inattentive.
- False Negative: A false negative indicates that the model correctly predicts the negative label. In this case, the model predicts the image being inattentive and the label is attentive.

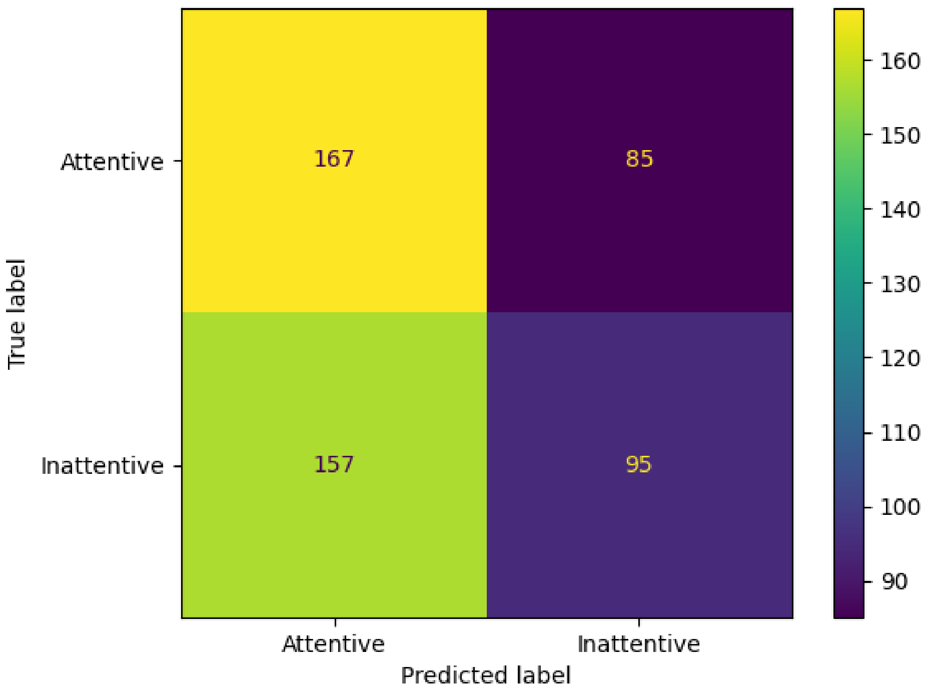


Figure 7. Model Confusion Matrix For Training Set

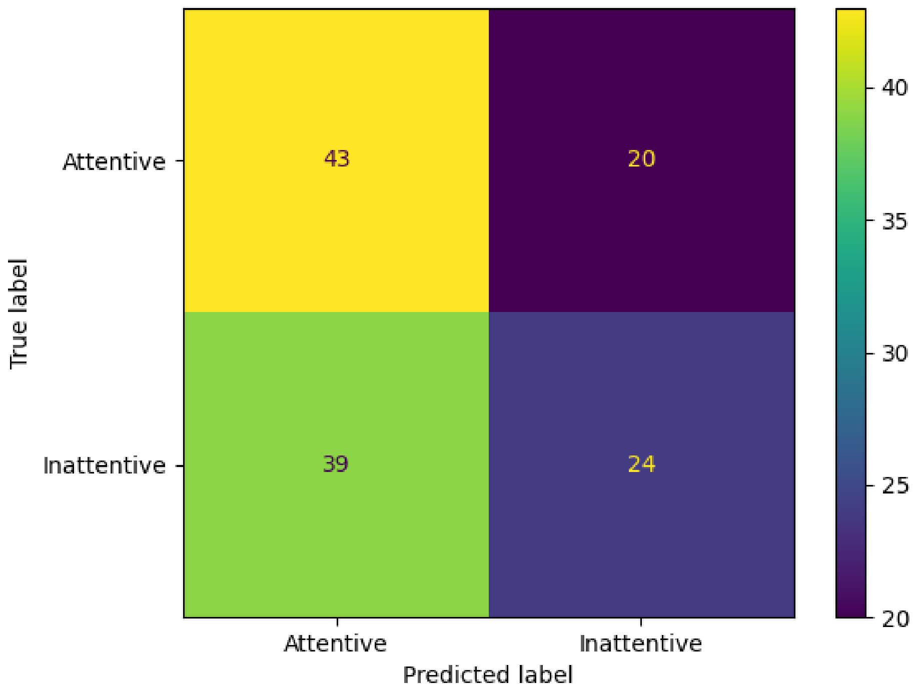


Figure 8. Model Confusion Matrix For Testing Set

From these confusion matrices in Figure 7 and Figure 8, a few observations can be made.

- The CNN model is generally biased towards labeling an image as attentive. The training set had 252 attentive images and 252 inattentive images, but the model labeled 167 of the attentive images as attentive (66% of the time an attentive image is labeled correctly as attentive) and 85 of the attentive images as inattentive. The testing set had 63 attentive images and 63 inattentive images, but the model labeled 43 of the attentive images as attentive (68% of the time an attentive image is labeled correctly as attentive) and 20 of the attentive images as inattentive. While the model labeled 95 of the inattentive images as inattentive (38% of the time an inattentive image is labeled correctly as inattentive) and 157 of the inattentive images as attentive. The testing set had 63 attentive images and 63 inattentive images, but the model labeled 24 of the inattentive images as inattentive

(38% of the time an inattentive image is labeled correctly as inattentive) and 39 of the inattentive images as attentive.

- If a student is attentive, the model is more likely to correctly identify it than if the student was inattentive. This matches the labelers' experience with labeling the images. Generally, all labelers quickly agreed if a person appears attentive in an image. In images where the participant is not clearly attentive, the labelers generally had trouble labeling the image with the same label as one another. These results mirror actual settings, where it is much harder to identify inattentive students when compared to attentive students.

To be able to apply the model to different datasets, this thesis evaluated the performance of the model on different datasets (though all of them were computer science students).

CHAPTER 4

RESEARCH EVALUATION FRAMEWORK

Four experiments were carefully planned and executed to collect data and validate different aspects of the NiCATS systems' usage in classrooms. While the research question being investigated across four experiments is the same, each experiment design evaluates that question in different settings and is built on the results from the earlier experiment. The relevant details of each of the experiments (research questions, independent and dependent variables, study participants, artifacts, experiment steps, and data gathered pre, post, and during the study) are shown in the following subsections.

Each of the experiments had a slightly different research question and dependent variables, but the independent variables remained the same. To answer each research question and to measure the usefulness of NiCATS in a controlled setting, the following independent variables were manipulated and their impacts were measured on dependent variables (listed in each experiment design section).

INDEPENDENT VARIABLES

1. Eye Gaze Points: The eye gaze points (x-coordinate, y-coordinate, and timestamp for each gaze point) collected during the experiment varied for different subjects. These gaze points were used to calculate the eye metrics, which are in turn analyzed for correlation with perceived facial attentiveness.
2. Amount of screenshots: The number of screenshots varied depending on the user interaction during their recording session (3.A).

4.1 EXPERIMENT DESIGNS

In order to determine the significance of the links between various eye metrics and perceived facial attentiveness, the four experiments were carefully designed to leverage what was learnt in previous experiments and to contrast these experiments. The progression allowed for the evaluation of a small dataset of images that would later be used to train the convolutional neural network, followed by the performance evaluation of the convolutional neural network in the experiments that used it. Experiment 1 did not use the CNN model as it was used as a preliminary test to determine if there were any correlations between the various eye metrics and perceived facial attentiveness of manually labeled images.

Experiment 1 was a highly controlled environment where each participant was asked to sit alone in front of a computer and watch a 15 minute pre-recorded lecture. Experiments 2 and 3 was a less-so controlled environment where participants were asked to sit in front of a computer in a computer lab with other students present and review fault seeded Java code. Experiment 4 was the least controlled environment where participants' data was simply collected while they took a CS1 exam. The purpose of the difference in the environmental context of each experiment was to allow the results of the eye metric and perceived facial attentiveness correlations to become more generalizable.

4.1.1 EXPERIMENT DESIGN 1

This section contains an overview of the first experiment design. This includes the research question, dependent variables, participants, setting, artifacts, procedures, and data collected during the experiment.

RESEARCH QUESTIONS

The experiment focused on understanding how perceived facial attentiveness can be predicted using facial images and eye gaze metrics on students viewing a pre-recorded video lecture. The following research question was investigated during the experiment:

Research Question - Can NiCATS data (facial image, gaze metrics) be used to understand and predict students' perceived attentiveness when reviewing the pre-recorded video lecture?

Dependent Variables: The effect of independent variables were measured on the following dependent variable:

- Perceived attentiveness: The attentiveness scores were calculated that ranged between 0 (inattentive) and 1 (attentive) representing the level of perceived attentiveness of the student. The attentiveness labels were produced by labelers following the BERI protocol (Carter et al., 2020). The face image data collected during the experiment was used to calculate the attentiveness score. This experiment was used to determine the feasibility of correlation between perceived facial attentiveness (manually labeled) and eye metrics. Therefore, this experiment did not use a CNN model to produce an attentiveness label but instead used a post-hoc manual labeling of collected images to create the attentive and inattentive dataset. The following experiments (experiment 2 and onwards) did use the CNN model to produce attentiveness labels and the performance of the model was evaluated against manually labeled images.

Participants and Environment: Participants consisted of varied ages and backgrounds, with most having computer science experience. The NiCATS system was tested on a computer with an i7-3770 CPU and 16 GB of RAM to make sure that data was collected correctly. Each participant in the actual experiment was asked to review a pre-recorded lecture on Human Error Abstraction Training and answer questions related to the lecture before and after watching the video.

Experiment Artifacts: Each participant was presented with a pre-recorded 15-minute lecture regarding Human Errors and their applications in everyday life. This lecture was selected due to it being generic enough that prior knowledge of computer science would not heavily affect the measured attentiveness. This video was prepared as part of an REU grant (by someone external to the research team - mitigating researcher bias) and was used to train the general public on the significance of human errors. The pre-recorded lecture had slides of varying font sizes and information density, along with visual aids. The diversity of slide content and layout allowed us to better distinguish participant attentiveness.

A computer classroom environment was simulated by having the participants sit at a computer equipped with a monitor-mounted 1080p webcam and a monitor-mounted Tobii Eye Tracker 4c. The webcam was mounted at the top-center of the monitor and the eye tracker was mounted at the bottom-center of the monitor. The participant computers varied in hardware but were all adequate to reliably and accurately collect face images, eye gaze points, and screenshots using the NiCATS client software. Figure 9 shows an example of the experiment setup.

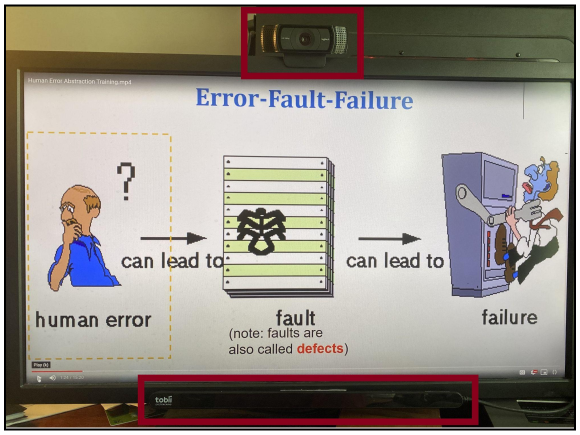


Figure 9. Webcam And Eye Tracker Mounted

Experiment Steps:

Step 1 — Pretest: The participants were instructed to take an online pre-test with 10 questions related to the pre-recorded lecture. The purpose of this test, when paired with a post-test, is to measure the baseline of knowledge each participant has and the knowledge acquisition process.

Step 2 — Calibration: The participants were instructed to install the NiCATS client software and Tobii Eye Tracking Core Software to their machines. The Tobii Eye Tracking Core Software was calibrated to the individual so its collection of data would be accurate. While the informed consent of the participants was collected before asking them to run the program, the

NiCATS client software prompts the participant with a message describing the data that will be collected during the experiment session and asks them to opt-in or opt-out of the data collection. After the participant opts in, the NiCATS client software initiates the data collection features until the experiment session has ended.

Step 3 — The participants were instructed to watch the 15-minute recorded lecture while their gaze points, face images, and screenshots were collected for analysis. The recorded lecture was about Human Error Abstraction Training. After the video, the participants were instructed to end the NiCATS client software.

Step 4 — The participants were instructed to take an online post-test with the same 10 questions as the pre-test. The analysis of their performance on the pre and post test give insight into the knowledge acquisition process and they can be used to give feedback on the instructional design of materials amongst other things.

4.1.2 EXPERIMENT DESIGN 2 AND 3

This section lists the research questions, lists independent and dependent variables, provides information about the study settings (participants, artifacts), study steps, and data gathered pre, post, and during the study.

The research question investigated in the study is listed below:

Research Question: Can NiCATS data (facial image, gaze metrics) be used to predict students' perceived attentiveness at different points during the code review exercise in a classroom?

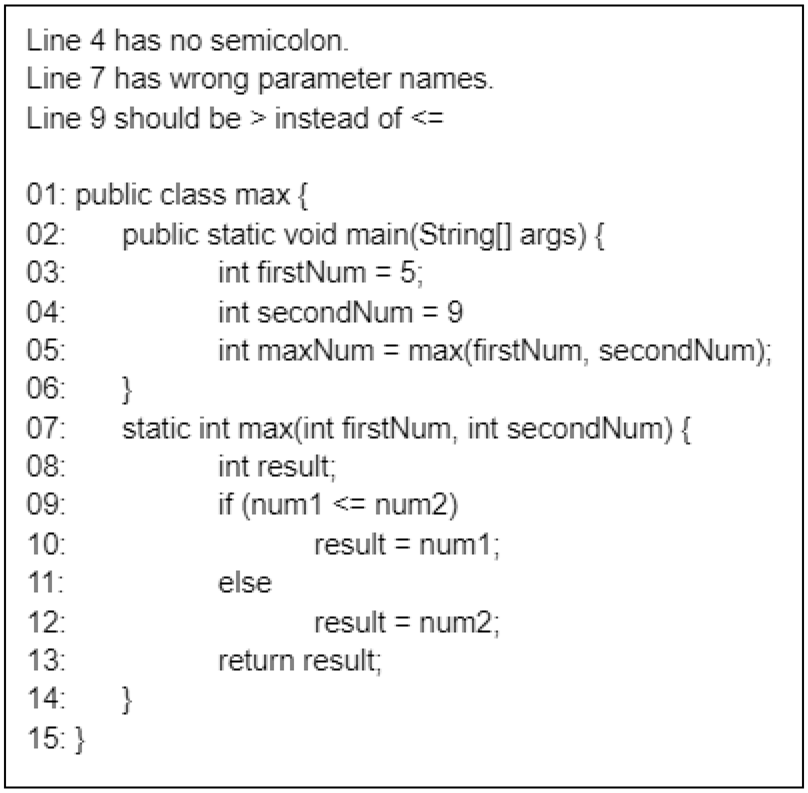
Dependent Variables: The effect of independent variables was measured on the following dependent variable:

- Perceived attentiveness: The attentiveness scores were calculated that ranged between 0 (inattentive) and 1 (attentive) representing the level of perceived attentiveness of the student. The attentiveness labels were produced by a convolutional neural network that was trained on face images. The face images were labeled by labelers following the BERI protocol (Carter et al., 2020). The face image data collected during the experiment was used to calculate the attentiveness score.

Participants and Environment: Participants consisted of undergraduate computer science students that were enrolled in the second sequence of the introductory programming course (CS2). Like experiment 1, experiments 2 and 3 were also voluntary. The NiCATS system was tested on a computer with an i7-3770 CPU and 16 GB of RAM to make sure that data was collected correctly. Each participant in both experiments reviewed Java code samples of varying complexity that contained faults (that represented the nature of errors students in CS2 commit during the code development). Post code review, students completed a free-response quiz to indicate the line numbers that contained the faults and with an explanation on why they are faults. The experiments were timed so that for each code sample, the students were given an equal amount of time to review the source code and about another four minutes to take the quiz. In total, the experiments took about 40 minutes each to complete. There were no common participants in experiment 2 (19 participants) and experiment 3 (participants). The only difference in experiments 2 and 3 was the code samples that students reviewed. These students were from the same course (CS2) in the same semester.

The participants were instructed to sit in a computer classroom environment on computers equipped with a monitor-mounted 1080p webcam and a monitor-mounted Tobii Eye Tracker 4c. The webcam was mounted at the top-center of the monitor and the eye tracker was mounted at the bottom-center of the monitor. The student computers contained an i7-3770 CPU and 16 GB of RAM, which is adequate for running the NiCATS student client program. Figure 9 shows an example of the experiment setup.

Experiment Material: Each participant in both experiments was presented with four Java source code samples that contained faults. The code samples were different for experiment 2 and experiment 3. The code samples were displayed at the same time in full screen images that covered the participants' entire computer monitor. This was so the data collected could be consistent across participants. With full screen images, the extracted eye data patterns could be compared between participants. Each code sample contained multiple faults, ranging from three to seven. The faults consisted of syntax and logic errors. Each fault was representative of common errors the participants would have encountered in their courses. Figure 10 shows one of the Java code samples. In addition to the code samples, the participants were asked to take an online quiz that displayed the same code sample with an open-ended response for where the faults were located and why they were faults. While the eye data, screenshots, and face images were collected while the participants were reviewing the full screen code sample images, they were not collected while the participants were taking the quiz.



Line 4 has no semicolon.
Line 7 has wrong parameter names.
Line 9 should be > instead of <=

```
01: public class max {
02:     public static void main(String[] args) {
03:             int firstNum = 5;
04:             int secondNum = 9
05:             int maxNum = max(firstNum, secondNum);
06:     }
07:     static int max(int firstNum, int secondNum) {
08:             int result;
09:             if (num1 <= num2)
10:                     result = num1;
11:             else
12:                     result = num2;
13:             return result;
14:     }
15: }
```

Figure 10. Example Java Code Sample With Faults

Experiment Steps:

Step 1 — The participants were instructed to install the NiCATS client software to their machines. The Tobii Eye Tracking Core Software was calibrated to the individual so its information would be accurate. While the informed consent of the participants was collected before asking them to run the program, the NiCATS client software prompts the participant with a message describing the data that will be collected during the experiment session and asks them to opt-in or opt-out of the data collection. After the participant opts in, the NiCATS client software initiates the data collection features until the experiment session has ended.

Step 2 — In experiment 2, the participants were instructed to view the Java source code that was being streamed to their monitors by the instructor computer. In experiment 3, the participants were instructed to download, open, and full screen the first faulted Java code sample image. They were asked to review the code sample for any faults they could detect. These settings simulated different ways “peer code review” is utilized in the CS2 classroom. The NiCATS client software collected their face images, eye gaze points, and screenshots during their review of the code sample. After review (each review lasted 5 minutes), the image was automatically closed.

Step 3 — The participants were instructed to take the quiz corresponding to the code sample they reviewed. They were presented with the same faulted code sample. They were given 4 minutes to enter the code faults they could detect into the open-ended response area and submit the quiz. The NiCATS software did not collect their data while they were taking the quiz.

Steps 2 and 3 were repeated for all four code samples.

Step 4 — The participants were instructed to close the NiCATS program.

4.1.3 EXPERIMENT DESIGN 4

This section lists the research questions, lists independent and dependent variables, and provides information about the study settings (participants, artifacts), study steps, and data gathered pre, post, and during the study.

The research question investigated in the study is listed below:

Research Question: Can NiCATS data (facial image, gaze metrics) be used to predict students' perceived attentiveness at different points during the timed exam in a CS1 classroom?

To answer the above research question, the following independent variables were manipulated and their impacts were measured on dependent variables (also listed below).

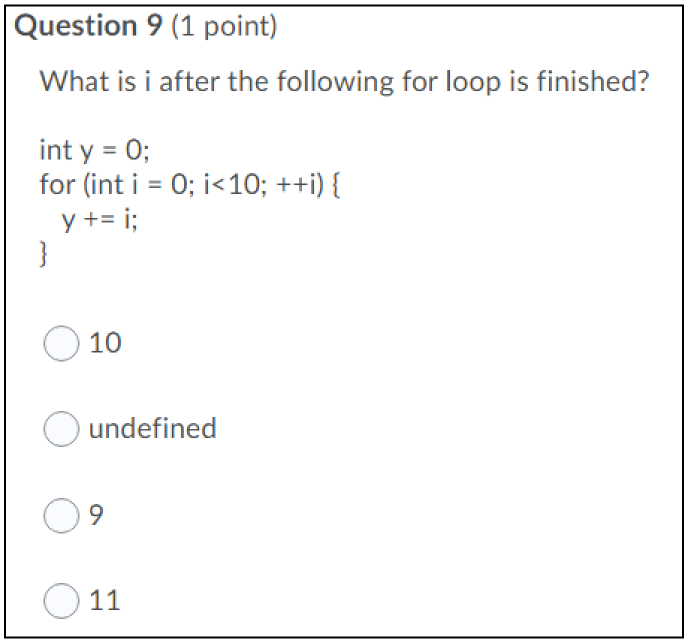
Dependent Variables: The effect of independent variables was measured on the following dependent variable:

- Perceived attentiveness: The attentiveness scores were calculated that ranged between 0 (inattentive) and 1 (attentive) representing the level of perceived attentiveness of the student. The attentiveness labels were produced by a convolutional neural network that was trained on face images. The face images were labeled by labelers following the BERI protocol (Carter et al., 2020). The face image data collected during the experiment was used to calculate the attentiveness score.

Participants and Environment: Participants consisted of undergraduate computer science students that were enrolled in the first of a two sequence programming (CS1) course. All participants volunteered to be in the experiment. Before the experiment run, the NiCATS system was tested on a computer with an i7-3770 CPU and 16 GB of RAM to make sure that data was collected correctly.Each participant in the experiment was asked to complete their CS1 midterm exam while the NiCATS client software collected their screenshots, face images, and eye data.

The participants were instructed to sit in a computer classroom environment on computers equipped with a monitor-mounted 1080p webcam and a monitor-mounted Tobii Eye Tracker 4c. The webcam was mounted at the top-center of the monitor and the eye tracker was mounted at the bottom-center of the monitor. The student computers contained an i7-4790 CPU and 4GB of RAM, which is adequate for running the NiCATS student client program. Figure 9 shows an example of the experiment setup.

Experiment Material: Each participant was asked to take their beginner CS1 midterm exam. The content covered basic Java syntax and logic. The exam consisted of 15 multiple choice and true/false questions and an additional programming section that asked the students to write a simple program. The exam lasted for 75 minutes and every participant stayed for the entire duration. Figure 11 This shows an example of one of the questions the participants were asked to answer. While the content was not synced between students (the students didn't see the same questions at the same time), screenshots of the questions were recorded. This allows for the possibility of post hoc comparison of eye data on the same questions.



Question 9 (1 point)

What is i after the following for loop is finished?

```
int y = 0;
for (int i = 0; i<10; ++i) {
  y += i;
}
```

- ◯ 10
- ◯ undefined
- ◯ 9
- ◯ 11

Figure 11. Example Test Question

Experiment Steps:

Step 1 — The participants were instructed to install the NiCATS client software to their machines. The Tobii Eye Tracking Core Software was calibrated to the individual so its information would be accurate. While the informed consent of the participants was collected before asking them to run the program, the NiCATS client software prompts the participant with a message describing the data that will be collected during the experiment session and asks them to opt-in or opt-out of the data collection. After the participant opts in, the NiCATS client software initiates the data collection features until the experiment session has ended.

Step 2 — The participants were instructed to complete their beginner CS1 exam while the NiCATS client software ran.

The NiCATS client software collected their face images, eye gaze points, and screenshots during their midterm exam.

Step 3 — The participants were instructed to close the NiCATS program at the conclusion of their exam.

4.2 CALCULATION METRICS

An overview of metrics and their calculations for all four experiments are presented in Table 3. The "evaluation criteria" provides a brief explanation of the calculation of individual data points that were then averaged for the entire experiment. For experiments that used the CNN model for evaluation of attentiveness, Table 3 presents the evaluation criteria. Table 4 presents the evaluation criteria for the CNN model for each experiment that uses it.

Metric	Feature	Evaluation Criteria
D1	Perceived Facial Attentiveness	For each student's face image collected during the entire session, the sum of all CNN-produced face image labels were divided by the total # of images (labeled attentive/inattentive).
IV1	Fixations per second	Total numbers of fixations within a 5-second interval of a face image (i.e., all fixations between 2.5 seconds before the face image and 2.5 seconds after) divided by time interval (5 seconds).
IV2	Average fixation duration (ms)	All fixation durations within a 5-second window of a face image (between 2.5 seconds before and 2.5 seconds after) are summed up and divided by the number of fixations within the 5-second window.
IV3	Saccades per second	Total numbers of saccades within a 5-second interval of a face image (i.e., all fixations between 2.5 seconds before the face image and 2.5 seconds after) divided by time interval (5 seconds).
IV4	Average Saccade Duration (ms)	All saccade durations within a 5-second window of a face image (between 2.5 seconds before and 2.5 after) are summed up and divided by the number of fixations within the 5 second window.
IV5	Regression Rate	The duration of all saccades within a 5-second window of a face image (between 2.5 seconds before and 2.5 seconds after) whose directions are between 135° and 225° divided by duration of all saccades within that window range.

Table 3. Overview Of Metrics

D1.1	Accuracy	All correct labels output by the CNN model divided by all labels.
D1.2	Precision	All correct positive labels output by the CNN model divided by positive labels.
D1.3	Recall	All correct positive labels output by the CNN model divided by all correct positive labels and all incorrect negative labels.
D1.4	F-1 Measure	(2*precision*recall)/(precision + recall)

Table 4. Overview Of Performance Metrics For Cnn Model

CHAPTER 5

RESULTS AND DISCUSSION

This section provides an analysis of the data collected during each of the experiments. Experiment results are organized around research questions listed in Section 4.

5.1 MACHINE LEARNING MODEL

For each experiment (except for experiment 1), the CNN model was validated against the manually-labeled ground truth to determine the accuracy of the model. Experiment 1 did not use the CNN model and was manually labeled to test for the ground truth of attentiveness. The purpose of experiment 1 was to test the feasibility of finding correlations between perceived facial attentiveness and eye metrics. Once they were established from experiment 1, experiments 2, 3, and 4 used the CNN model for the labeling of images.

5.1.1 MODEL EVALUATION

A classification report was generated for each experiment that used the CNN model to determine the effectiveness of the CNN model for predicting student attentiveness. The results are presented in Figures 5., 6., and 7. The results are summarized below.

	Accuracy	**precision**	**recall**	**f1-score**	**support**
Attentive	0.77	0.85	0.88	0.87	1832
Inattentive		0.16	0.13	0.14	333

Table 5. Model Classification Report For Experiment 2

The accuracy of the machine learning model for experiment 2 is 77%. This means that the model correctly predicts the label of the image 77% of the time. The precision for attentiveness is 85% and the recall is 88%. These results match the evaluation of the CNN model during training and testing. In general, the model is better at predicting truly attentive images as being attentive than it is at predicting truly inattentive images as being inattentive. This experiment in particular has a higher accuracy than what was initially indicated during the training and testing. This performance can be attributed to the settings where it was a highly-controlled experiment environment as students were asked to review source code (for short intervals) in short intervals. As is shown, 1832 images collected were labeled attentive which translates to higher precision and recall (since the model is shown to identify attentive students better).

	Accuracy	**precision**	**recall**	**f1-score**	**support**
Attentive	0.67	0.82	0.75	0.78	1347
Inattentive		0.22	0.30	0.25	315

Table 6. Model Classification Report For Experiment 3

The accuracy of the machine learning model for experiment 3 is 67%. This means that the model correctly predicts the label of the image 67% of the time. The precision for attentiveness is 85% and the recall is 88%. Just like with the training and testing set, and the results from experiment 2, the model seems to be better at predicting the label of images with a ground truth of attentive rather than inattentive. This experiment has higher accuracy than what was produced in the training and testing of the model. Just like with experiment 2, this can be attributed to the experiment environment (source code review in short time intervals) contributing to a better performance in labeling.

	Accuracy	**precision**	**recall**	**f1-score**	**support**
Attentive	0.5	0.85	0.51	0.63	4163
Inattentive		0.14	0.47	0.14	722

Table 7. Model Classification Report For Experiment 4

The accuracy of the machine learning model for experiment 4 is 50%. This means that the model correctly predicts the label of the image 50% of the time. The precision for attentiveness is 85% and the recall is 51%. As is the same with the previous experiments, the model is better at predicting attentive labels than inattentive labels. This classification report differs from the previous experiments in that the accuracy is closer to 50% than the accuracy of the previous experiments. This can be interpreted as exams generally producing more ambiguous images that are harder to accurately label and may be due to writing code vs attempting multiple-choice questions, scrolling/moving between questions, and working with multiple windows during the exam. While the accuracy in this experiment was below expectation, some useful insights were gained with respect to the eye-tracking data which can be used in the future to determine the feasibility of using NiCATS for attentiveness prediction in exams.

5.1.2 CONFUSION MATRIX

Accuracy alone can potentially be misleading if the datasets are imbalanced. A confusion matrix allows for visualization of the performance of a classification machine learning model. It allows for insight into the types of errors the model is making. Importantly, it shows the true positives, true negatives, false positives, and false negatives.

- True Positive: A true positive indicates that the model correctly predicts the positive label. In this case, the model predicts the image being attentive and the label is attentive.
- True Negative: A true negative indicates that the model correctly predicts the negative label. In this case, the model predicts the image being inattentive and the label is inattentive.
- False Positive: A false positive indicates that the model incorrectly predicts the positive label. In this case, the model predicts the image being attentive and the label is inattentive.

- False Negative: A false negative indicates that the model correctly predicts the negative label. In this case, the model predicts the image being inattentive and the label is attentive.

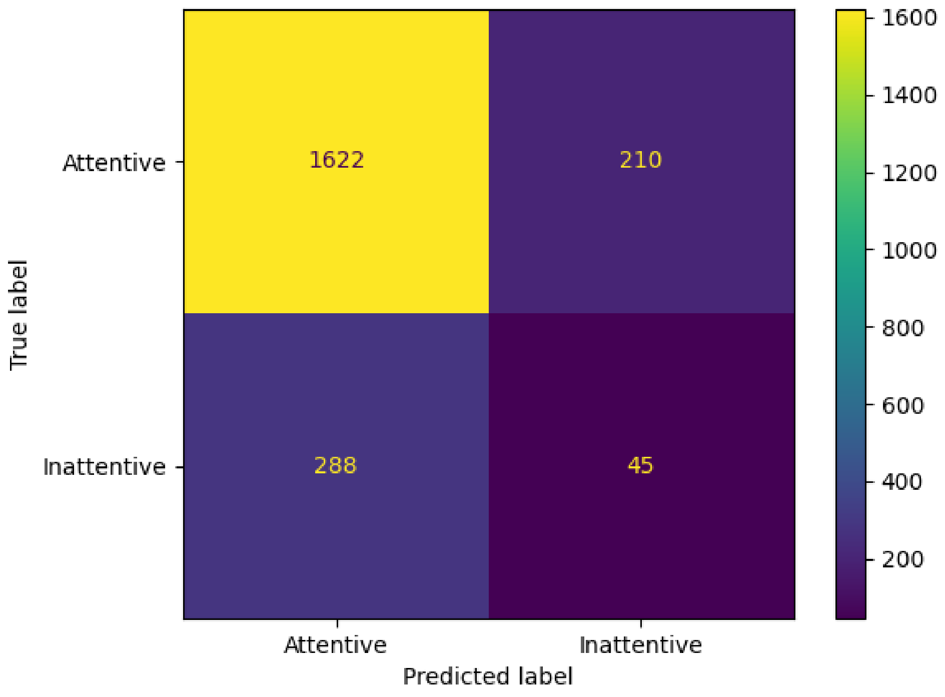


Figure 12. CNN Model Confusion Matrix For Experiment 2

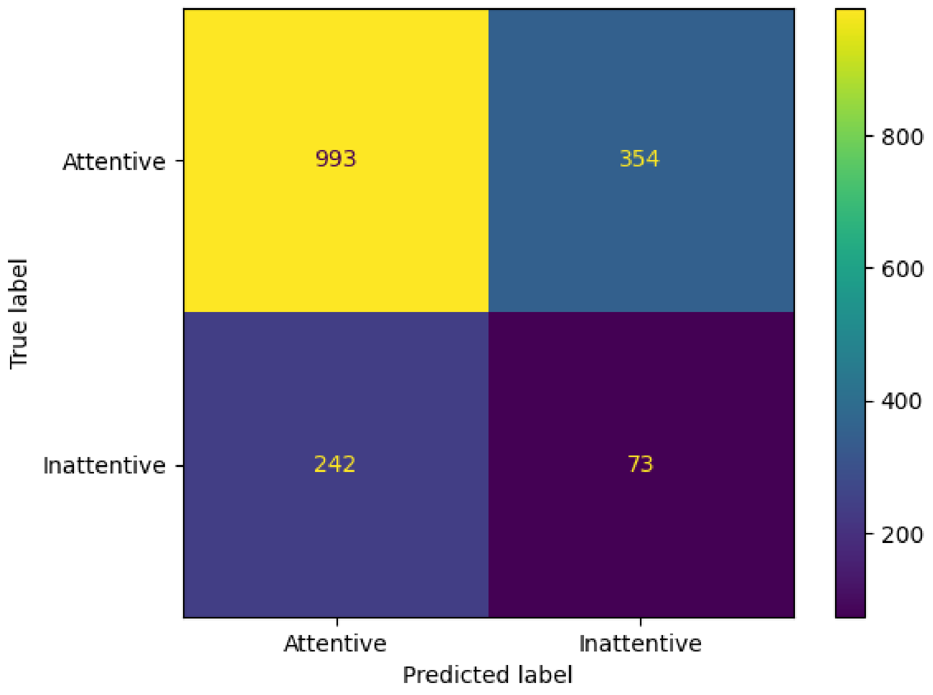


Figure 13. CNN Model Confusion Matrix For Experiment 3

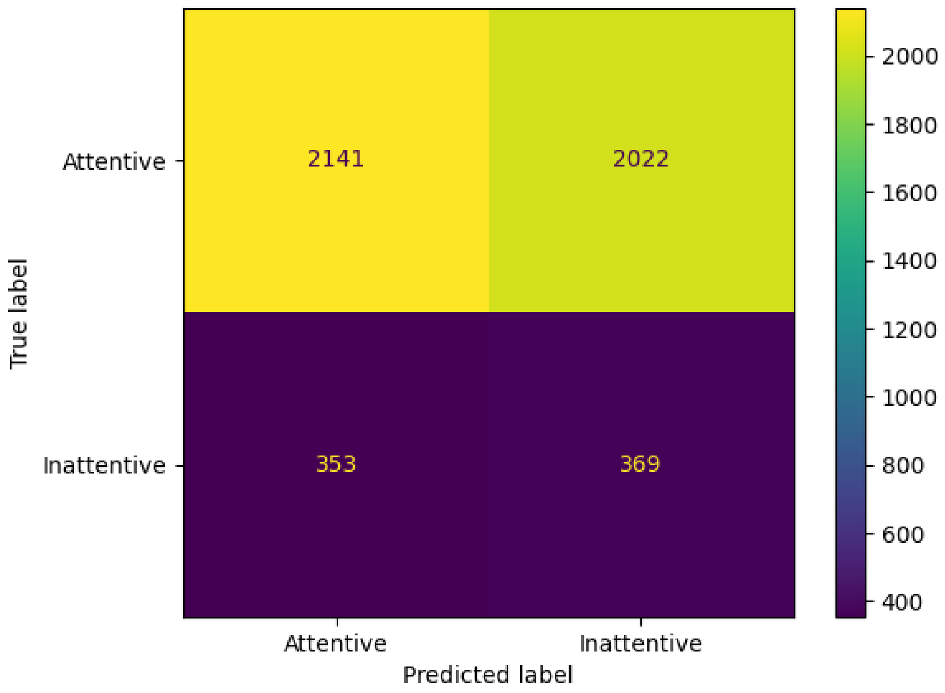


Figure 14. CNN Model Confusion Matrix For Experiment 4

As shown in the confusion matrices (Figure 12, 13, and 14) for experiments 2, 3, and 4, the CNN model is much better at classifying attentive images than classifying inattentive images. These findings are similar to the confusion matrix findings when training and testing the set.

The experiment 2 dataset had 1,832 attentive images and 333 inattentive images, but the model labeled 1,622 of the attentive images as attentive (89% of attentive images were correctly labeled as attentive) and 210 of the attentive images as inattentive. The experiment 3 dataset had 1,347 attentive images and 315 inattentive images, but the model labeled 993 of the attentive images as attentive (74% of attentive images were correctly labeled as attentive) and 354 of the attentive images as inattentive. The experiment 4 dataset had 4,163 attentive images and 722 inattentive images, but the model labeled 2,141 of the attentive images as attentive (51% of attentive images were correctly labeled as attentive) and 2,022 of the attentive images as inattentive.

Just like with the training and testing sets, if an image is attentive, the model is more likely to correctly identify it than if the image was inattentive. This matches the labelers' experience with labeling the images. Generally, all labelers quickly agreed if a person appears attentive in an image. In images where the participant is not clearly attentive, the labelers generally had trouble labeling the image with the same label as one another labeller.

5.2 EXPERIMENT RESULTS

5.2.1 EXPERIMENT 1

In Table 8, an overview of the dataset is provided (independent and dependent variables) and present the average values calculated after the first experiment run.

Metric	Feature	Mean	S.D.
D1	Perceived Facial Attentiveness (manually-labeled)	0.80	0.27
IV1	Fixations per second	1.59	0.89
IV2	Average fixation duration (ms)	322.44	266.13
IV3	Saccades per second	1.39	1.08
IV4	Average Saccade Duration (ms)	111.65	63.52
IV5	Regression Rate	0.32	0.22
D2	Pre-Test Score	4	1
D3	Post-Test Score	6.8	2.7

Table 8. Overview Of Experiment 1 Dataset

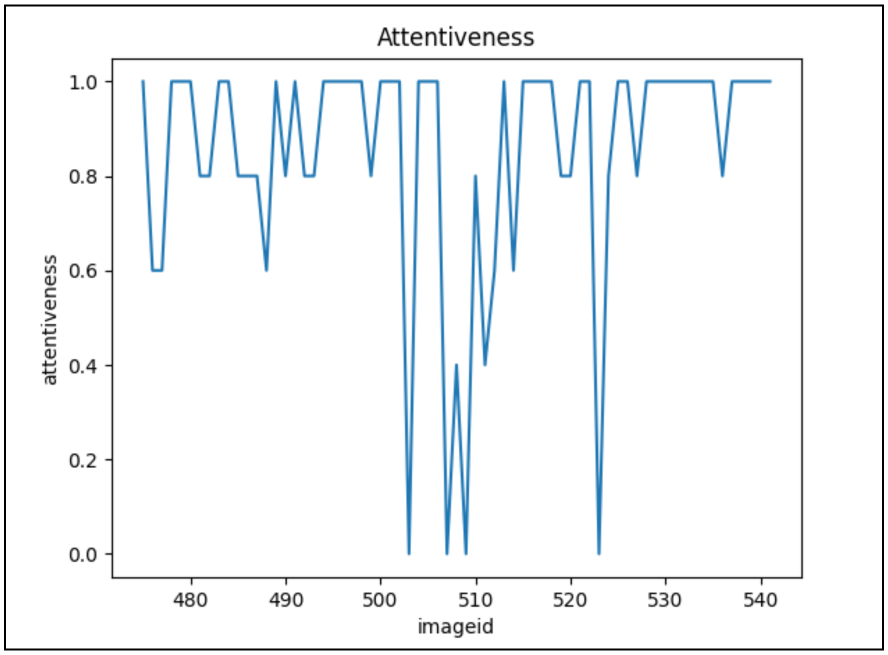


Figure 15. Example Of A Single Participant's Attentiveness Over Time

Figure 15 shows a single participant's attentiveness over time. Each participant displayed varying levels of attention throughout the recording session. In addition, the properties of each participant's fixations and saccades varied throughout the duration of the video along with their performance on pre and post-test scores.

To better understand the impact of eye metrics (IV1 through IV5) on student perceived attentiveness (D1), linear regression analysis was performed that measured the correlation intercept and coefficient along with the correlation strength (Adjusted R^2-value) and its statistical strength (p-value). The relationships between each eye metric and attentiveness are shown in Table 9.

Eye Metric	Coefficient	Adjusted R^2- value	P-Value
Fixations per second	0.031	0.009	**0.004**
Average fixation duration (ms)	<0.001	0.005	**0.042**
Saccades per second	-0.027	0.011	**0.002**
Average Saccade Duration (ms)	>-0.001	0.047	**<0.001**
Regression Rate	0.078	0.002	0.131

Table 9. Correlations Between Each Eye Metric And Perceived Facial Attentiveness

Based on the results provided in Tables 8 and 9, the relevance of the strongest and statistically significant correlations w.r.t research goals are presented:

- Fixations per second, the rate at which the participant fixates per second, has a positive correlation with perceived facial attentiveness. While the correlation is weak with an adjusted R^2-value of 0.009, it is statistically significant with a p-value of 0.003. This positive correlation can be interpreted as higher values meaning more visual effort being exerted, or potentially a higher efficiency in finding relevant information. The dataset has an average number of fixations per second of 1.594 (S.D of 0.88). With this information, it could be possible to assess whether some students in a classroom need more assistance in learning the information or not if their fixations per second are much higher or lower than the average of the class.
- Average fixation duration, which is the average duration of a participant's fixation, also has a positive correlation with perceived facial attentiveness. Yet again, the correlation is weak but statistically significant (p-value of 0.042). This positive correlation can be interpreted as higher values indicating more effort being used to find and interpret relevant information. An average length of fixation duration of 322 ms (S.D of 266 ms)

can indicate that the dataset has a slightly higher average fixation duration than normal, as normal fixations are generally between 200-300 milliseconds (Dewan et al., 2019).

- Saccades per second, the rate at which the participants' eyes rapidly move between two fixation points, has a negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.011) and statistically significant (p-value of 0.002). This negative correlation can be explained by a lower value indicating more focused searching of relevant information. The dataset has an average number of saccades per second of 1.397 (S.D of 1.080). Given more data, it could be determined if one set of students was better at searching for information than another set if their average number of saccades were significantly different from one another.
- Average saccade duration, which is the average duration of a participant's saccade, has a negative correlation with perceived facial attentiveness. The correlation is somewhat weak (R^2-value of 0.047), but is statistically significant (p-value of <0.001). This negative correlation can be interpreted as a lower value indicating quicker and more focused searching of relevant information. An average saccade duration of 111 ms (S.D of 63 ms) could indicate that the dataset was slightly worse at searching for relevant information than in general as saccades are generally between 40-50 milliseconds (Dewan et al., 2019).
- Regression rate, which is the percentage of time spent reading backward, and perceived facial attentiveness did not have a statistically significant correlation with a p-value of 0.131. The positive correlation is very weak with an R^2-value of 0.002. This weak positive correlation means that as the regression rate increased, perceived facial attentiveness slightly increased. An average regression rate of 0.32 (S.D. of 0.22) could be used with different artifacts to determine which one was more difficult to understand.

5.2.2 EXPERIMENT 2

In Table 10, an overview of the mean and standard deviation (S.D.) values of independent and dependent variables is provided (collected across the entire study) and the average values calculated after the experiment run are presented.

Metric	Feature	Mean	S.D.
D1	Perceived Facial Attentiveness (CNN Output)	0.58	0.20
IV1	Fixations per second	0.65	0.30
IV2	Average fixation duration (ms)	676.50	333.24
IV3	Saccades per second	1.19	0.94
IV4	Average Saccade Duration (ms)	116.00	185.57
IV5	Regression Rate	0.25	0.25

Table 10. Overview Of Experiment 2 Dataset

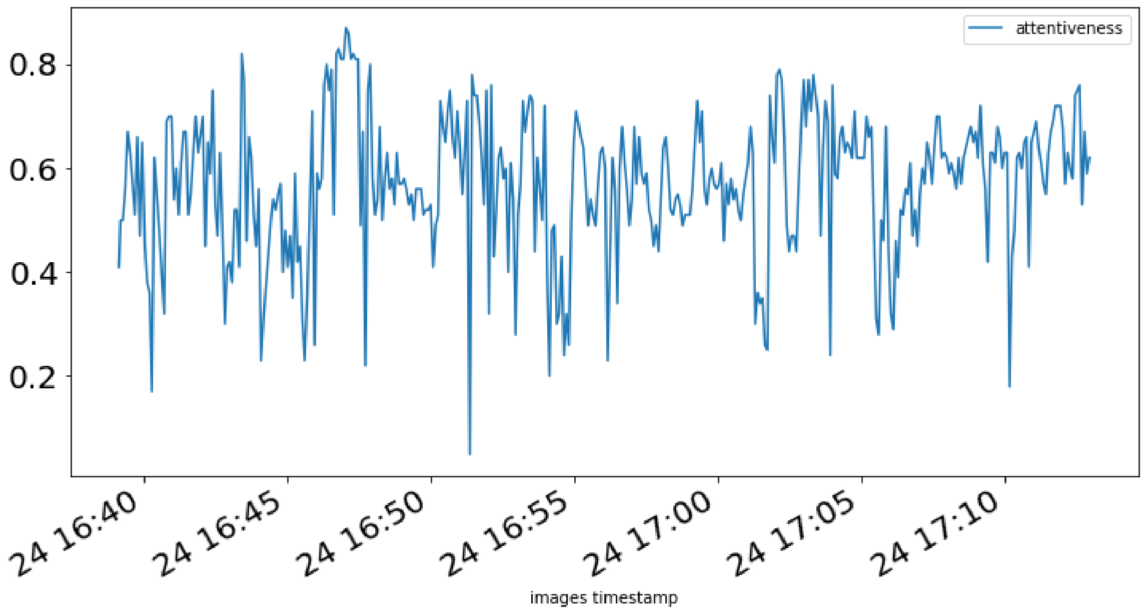


Figure 16. Example Of A Single Participant's Attentiveness Throughout Experiment 2

Figure 16 shows a single participant's attentiveness over time. The CNN model outputs a value between 0 and 1 to indicate the level of perceived attentiveness. An attentiveness level of 1 indicates the student appears attentive and an attentiveness level of 0 indicates the student appears inattentive. Each participant displayed varying levels of attention throughout the recording session. In addition, the properties of each participant's fixations and saccades varied throughout the duration of each of the Java code fault artifacts. Figure 17 shows a single participant's average saccade duration and average fixation duration throughout the recording session.

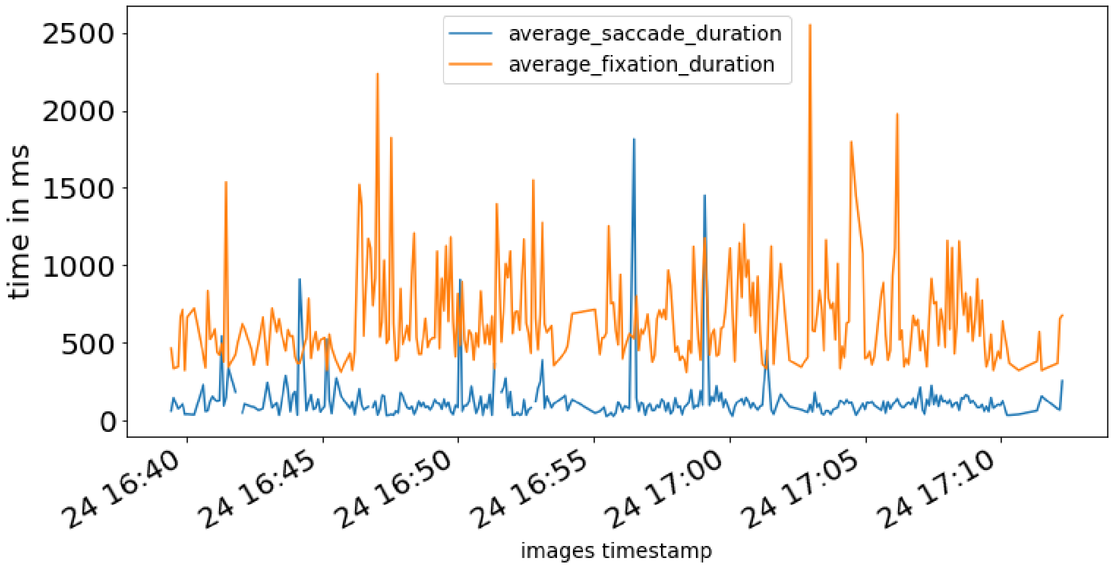


Figure 17. Example Of A Single Participant's Average Fixation Duration And Average Saccade Duration Throughout Experiment 2

To better understand the impact of eye metrics (IV1 through IV5) on student perceived attentiveness (D1), linear regression analysis was performed that measured the correlation intercept and coefficient along with the correlation strength (Adjusted R^2-value) and its statistical strength (p-value). The relationships between each eye metric and attentiveness are shown in Table 11.

Eye Metric	Coefficient	Adjusted R^2- value	P-Value
Fixations per second	0.022	<0.001	0.086
Average fixation duration (ms)	<0.001	<0.001	0.071
Saccades per second	-0.020	0.009	**<0.001**
Average Saccade Duration (ms)	<0.001	>-0.001	0.572
Regression Rate	-0.025	<0.001	0.094

Table 11. Correlations Between Each Eye Metric And Perceived Facial Attentiveness

Based on the results provided in Tables 10 and 11, the correlations w.r.t research goals are presented:

- Fixations per second, the rate at which the participant fixates per second, has a positive correlation with perceived facial attentiveness. The correlation is weak with an adjusted R^2-value of <0.001. It is not statistically significant with a p-value of 0.086. This result is similar to the results from experiment 1, but differs in that the coefficient is smaller, the adjusted R^2-value is much smaller, and the p-value is smaller. With this information, the interpretations from experiment 1 should still hold, but to a lesser extent. This positive correlation can be interpreted as higher values meaning more visual effort being exerted, or potentially a higher efficiency in finding relevant information. The dataset has an average number of fixations per second of 0.65 (S.D of 0.30). It could be possible to assess whether some students in a classroom need more assistance in learning the information or not if their fixations per second are much higher or lower than the average of the class.
- Average fixation duration, which is the average duration of a participant's fixation, also has a positive correlation with perceived facial attentiveness. The correlation is very weak with an adjusted R^2-value of below 0.001 and is not statistically significant (p-value of

0.071). This positive correlation can be interpreted as higher values indicating more effort being used to find and interpret the Java code faults. An average length of fixation duration of 676.50 ms (S.D of 333.24 ms) can indicate that the dataset has a much higher average fixation duration than normal, as normal fixations are generally between 200-300 milliseconds (Dewan et al., 2019). Because this experiment focused on Java source code faults, a higher average fixation duration would be expected as they will be spending more time focusing on understanding the code.

- Saccades per second, the rate at which the participants' eyes rapidly move between two fixation points, has a negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.009) and statistically significant (p-value of below 0.001). This negative correlation can be explained by a lower value indicating more focused searching of relevant information. The dataset has an average number of saccades per second of 1.19 (S.D of 0.94). Given more data, it could be determined if one set of students was more attentive while searching for information than another set if their average number of saccades were significantly different from one another.
- Average saccade duration, which is the average duration of a participant's saccade, has a slight positive correlation with perceived facial attentiveness. The correlation is negligible (R^2-value of >-0.001), and is not statistically significant (p-value of 0.572). This result differs from experiment 1, where average saccade duration was a statistically significant negative correlation with attentiveness. This can be interpreted as average saccade duration being explanatory while participants are watching videos but not actively searching for code faults.
- Regression rate, which is the percentage of time spent reading backward, and perceived facial attentiveness did not have a statistically significant correlation with a p-value of

0.094. The negative correlation is very weak with an R^2-value of <0.001. This weak negative correlation means that as the regression rate increased, perceived facial attentiveness slightly decreased. An average regression rate of 0.25 (S.D. of 0.25) could be used with different artifacts to determine which one was more difficult to understand. This result slightly differs from experiment 1 where the regression rate had a very weak positive correlation with perceived facial attentiveness. In both cases, however, the correlations were not statistically significant.

5.2.3 EXPERIMENT 3

In Table 12, an overview of the third experiment's dataset is provided(independent and dependent variables) and the average values calculated after the experiment run is presented.

Metric	Feature	Mean	S.D.
D1	Perceived Facial Attentiveness (CNN Output)	0.61	0.28
IV1	Fixations per second	0.67	0.30
IV2	Average fixation duration (ms)	661.80	355.72
IV3	Saccades per second	1.15	0.92
IV4	Average Saccade Duration (ms)	107.29	179.46
IV5	Regression Rate	0.25	0.24

Table 12. Overview Of Experiment 3 Dataset

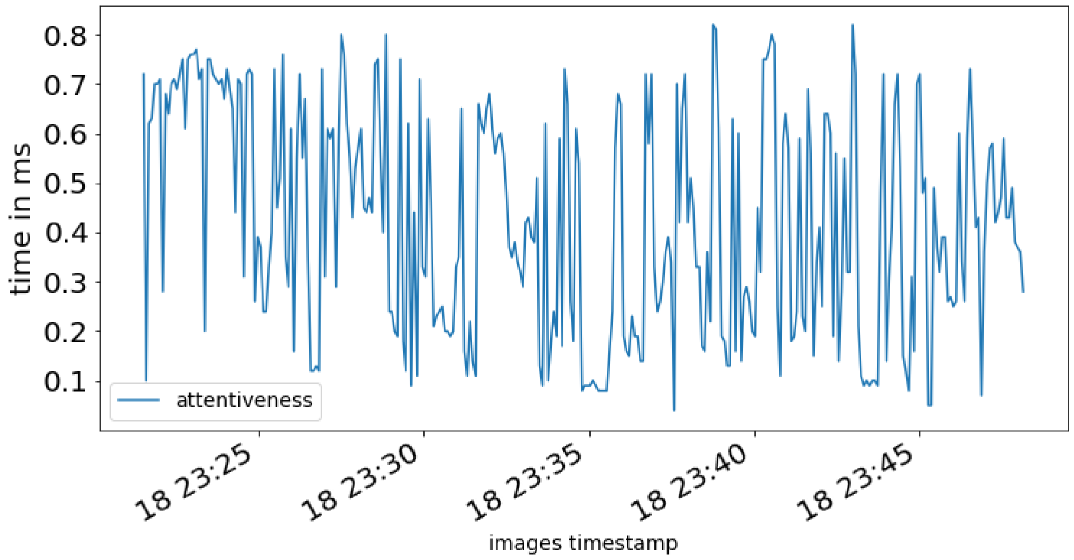


Figure 18. Example Of A Single Participant's Attentiveness Throughout Experiment 3

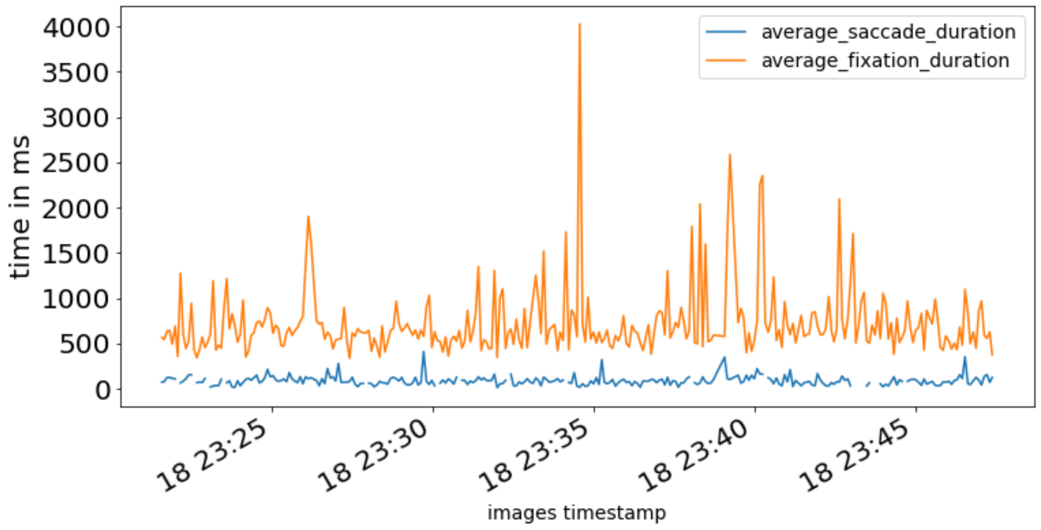


Figure 19. Example Of A Single Participant's Average Fixation Duration And Average Saccade Duration Throughout Experiment 3

Eye Metric	Coefficient	Adjusted R^2- value	P-Value
Fixations per second	0.138	0.030	**<0.001**
Average fixation duration (ms)	<0.001	0.003	**0.010**
Saccades per second	-0.040	0.022	**<0.001**
Average Saccade Duration (ms)	>-0.001	0.009	**<0.001**
Regression Rate	0.052	0.002	**0.029**

Table 13. Correlations between each eye metric and perceived facial attentiveness

Based on the results provided in Tables 12 and 13, the correlations w.r.t research goals are presented:

- Fixations per second, the rate at which the participant fixates per second, has a positive correlation with perceived facial attentiRegression rate, which is the percentage of time spent reading backward, and perceived facial attentiveness. The correlation is weak with an adjusted R^2-value of 0.030. It is statistically significant with a p-value of <0.001. This result is similar to the results from experiment 1 and 2, but differs in that the coefficient is much larger, the adjusted R^2-value is larger, and the p-value is smaller. With this information, the interpretations from experiment 1 and 2 should still hold. This positive correlation can be interpreted as higher values meaning more visual effort being exerted, or potentially a higher efficiency in finding relevant information. The dataset has an average number of fixations per second of 0.67 (S.D of 0.28). It could be possible to assess whether some students in a classroom need more assistance in learning the information or not if their fixations per second are much higher or lower than the average of the class, or if a student is more or less efficient at finding code faults..
- Average fixation duration, which is the average duration of a participant's fixation, also has a positive correlation with perceived facial attentiveness. The correlation is weak with

an adjusted R^2-value of 0.003 and is statistically significant (p-value of 0.010). This positive correlation can be interpreted as higher values indicating more effort being used to find and interpret the Java code faults. An average length of fixation duration of 676.50 ms (S.D of 333.24 ms) can indicate that the dataset has a much higher average fixation duration than normal, as normal fixations are generally between 200-300 milliseconds (Dewan et al., 2019). Because this experiment focused on Java source code faults, a higher average fixation duration would be expected as they will be spending more time focusing on understanding the code.

- Saccades per second, the rate at which the participants' eyes rapidly move between two fixation points, has a negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.022) and statistically significant (p-value of below 0.001). This negative correlation can be explained by a lower value indicating more focused searching of relevant information. The dataset has an average number of saccades per second of 1.15 (S.D of 0.92). Given more data, it could be determined if one set of students was more attentive while searching for code faults than another set if their average number of saccades were significantly different from one another.
- Average saccade duration, which is the average duration of a participant's saccade, has a slight negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.009), but is statistically significant (p-value of <0.001). This result differs from experiment 2, where average saccade duration was a non-statistically significant positive correlation with attentiveness. It can be interpreted from these results that students that have a longer average saccade duration are generally perceived as having less attentiveness as they are paying less attention to the material. If a class's average

saccade duration increases, a professor can use different activities in an attempt to increase the attentiveness of the students.

- Regression rate, which is the percentage of time spent reading backward, and perceived facial attentiveness have a statistically significant correlation with a p-value of 0.029. The correlation is positive and very weak with an R^2-value of 0.002. This weak positive correlation means that as the regression rate increased, perceived facial attentiveness slightly increased. An average regression rate of 0.25 (S.D. of 0.24) could be used with different artifacts to determine which one was more difficult to understand or caused the participant to become less attentive. This result is similar to experiment 1 where regression rate had a very weak positive correlation with perceived facial attentiveness but differs from experiment 2 where regression rate had a non-statistically significant weak negative correlation with perceived facial attentiveness. Different from both, however, is that regression rate in experiment 3 is statistically significant. This can be interpreted as a student being more attentive if they spend more time reading backward to find relevant information such as code faults.

5.2.4 EXPERIMENT 4

In Table 14, an overview of the fourth experiment's dataset is provided(independent and dependent variables) and the average values calculated after the experiment run is presented.

Metric	Feature	Mean	S.D.
D1	Perceived Facial Attentiveness (CNN Output)	0.47	0.27
IV1	Fixations per second	0.60	0.28
IV2	Average fixation duration (ms)	700.29	402.24
IV3	Saccades per second	1.24	1.00
IV4	Average Saccade Duration (ms)	124.59	181.06
IV5	Regression Rate	0.25	0.25

Table 14. Overview Of Experiment 4 Dataset

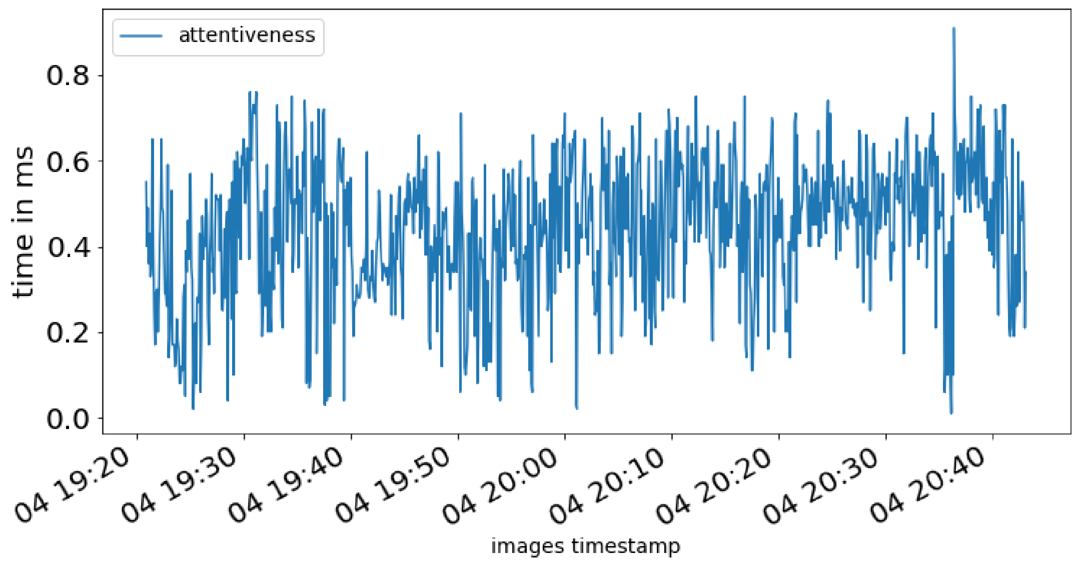


Figure 20. Example Of A Single Participant's Cnn-Outputted Attentiveness Throughout

Experiment 4

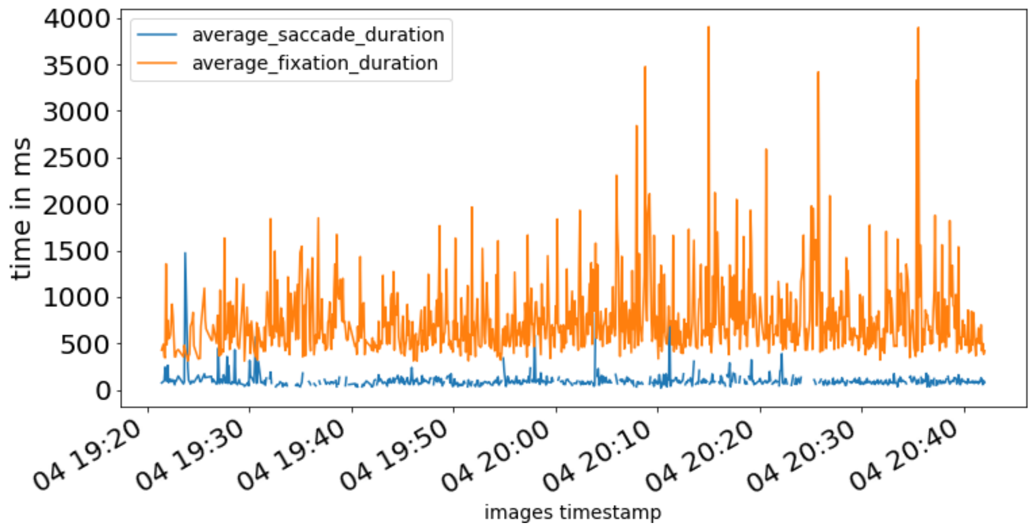


Figure 21. Example Of A Single Participant's Average Fixation Duration And Average Saccade Duration Throughout Experiment 4

Eye Metric	Coefficient	Adjusted R^2- value	P-Value
Fixations per second	0.145	0.021	**<0.001**
Average fixation duration (ms)	<0.001	0.005	**<0.001**
Saccades per second	-0.020	0.005	**<0.001**
Average Saccade Duration (ms)	>-0.001	0.013	**<0.001**
Regression Rate	-0.056	0.002	**<0.001**

Table 15. Correlations Between Each Eye Metric And Perceived Facial Attentiveness

For experiment 4, each eye metric's result was statistically significant. This may be due to the size of the dataset. Experiment 4's experiment collected nearly three times as much data as experiment 2. Based on the results provided in Tables 14 and 15, the correlations w.r.t research goals are presented:

- Fixations per second, the rate at which the participant fixates per second, has a positive correlation with perceived facial attentiveness. The correlation is somewhat weak with an adjusted R^2-value of 0.0212. It is statistically significant with a p-value of <0.001. This result is similar to the results from experiments 1, 2, and 3, but differs in that the coefficient is larger, the adjusted R^2-value is on the larger size, and the p-value is smaller. With this information, the interpretations from experiment 1 should still hold. This positive correlation can be interpreted as higher values meaning more visual effort being exerted, which would be reflected in their perceived facial attentiveness. The dataset has an average number of fixations per second of 0.60 (S.D of 0.28). It could be possible to assess whether some students in a classroom need more assistance in learning the information or not if their fixations per second are much higher or lower than the average of the class.

- Average fixation duration, which is the average duration of a participant's fixation, also has a positive correlation with perceived facial attentiveness. The correlation is weak with an adjusted R^2-value of 0.005 and is statistically significant (p-value of <0.001). This positive correlation can be interpreted as higher values indicating more effort being used to find and interpret relevant information. An average length of fixation duration of 700.29 ms (S.D of 402.24 ms) can indicate that the dataset has a much higher average fixation duration than normal, as normal fixations are generally between 200-300 milliseconds (Dewan et al., 2019). Because this experiment focused on a CS1 exam, a higher average fixation duration would be expected as they will be spending more time focusing on understanding the questions presented to them.
- Saccades per second, the rate at which the participants' eyes rapidly move between two fixation points, has a negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.005) and statistically significant (p-value of below 0.001). This negative correlation can be explained by a lower value indicating more

focused searching of relevant information. The dataset has an average number of saccades per second of 1.24 (S.D of 1.00). It can be interpreted that as saccades per second increase, a student is less attentive towards the material as they are sporadically searching for information instead of being more focused in their search.

- Average saccade duration, which is the average duration of a participant's saccade, has a slight negative correlation with perceived facial attentiveness. The correlation is weak (R^2-value of 0.013), and is statistically significant (p-value of <0.001). This result is similar to experiments 1 and 3, wherein both cases average saccade duration was a statistically significant negative correlation with attentiveness. This result differs from experiment 2, where the average saccade duration was non-statistically significant and had a positive correlation. This can be interpreted as students being less attentive if their average saccade durations are longer because they are less focused on finding information.
- Regression rate, which is the percentage of time spent reading backward, and perceived facial attentiveness did not have a statistically significant correlation with a p-value of <0.001. The negative correlation is very weak with an R^2-value of 0.002. This weak negative correlation means that as the regression rate increased, perceived facial attentiveness slightly decreased. An average regression rate of 0.25 (S.D. of 0.25) could be used with different artifacts to determine which one was more difficult to understand. This result slightly differs from experiment 1 where regression rate had a very weak positive correlation with perceived facial attentiveness. In both cases, however, the correlations were not statistically significant.

5.3 COMMONALITY OF RESULTS ACROSS EXPERIMENTS

Across all experiments, there were a number of commonalities.

- Fixations per second was always a positive correlation with perceived facial attentiveness, but was only statistically significant in experiments 1, 3, and 4. The strength of the relationship was slightly inconsistent. In experiments 1 and 2, the adjusted R^2-value was below 0.01, whereas in experiments 3 and 4, the adjusted R^2-value was above 0.01. When the adjusted R^2-value is higher, the value better explains the variance of another value. In the results where fixations per second have a higher R^2-value, the fixations per second values are better explanatory of the variance of attentiveness. When considering the statistical significance of each experiment's fixations per second, it can be interpreted that in activities where the participants are passively observing, such as watching a pre-recorded lecture, fixations per second are less indicative of attentiveness than in activities where participants are actively engaging, like in a CS1 exam.
- Average fixation duration had a weak positive correlation in all four experiments but was only statistically significant in experiments 3 and 4.
- Saccades per second had a weak negative statistically significant correlation in all four experiments and was the only correlation to be statically significant in all four experiments. In all cases, it seems that saccades per second as an eye metric can be used to increase the accuracy of a predictive model than if it wasn't included.
- Average saccade duration had a weak negative statistically significant correlation in three of the four experiments. Average saccade duration in experiment two had a weak positive correlation but was not statistically significant at a p-value of 0.572. It had the strongest relationship in experiment 1 where the participants watched a pre-recorded lecture. This can be interpreted that in activities that have the participants passively engaging with the

material, the average saccade duration is more indicative of attentiveness than in activities where the participants are actively engaging with the material.

- Regression rate had inconsistent results between the experiments. In experiments 1 and 3, regression rate had a weak positive correlation, while in experiments 2 and 4, it had a weak negative correlation. It was statistically significant in experiments 3 and 4, which had a negative correlation and positive correlation, respectively.

Based on these results, it could be possible to combine the CNN model output with each of the stronger correlations to make a more accurate predictive model. Fixations per second, average fixation duration, saccades per second, and average saccade duration all have consistent enough results to be considered for use in creating a better predictive model. Regression rate, based on the results from all four experiments, seems to not be consistent enough for consideration.

CHAPTER 6

CONCLUSION

This thesis presented the investigation, design, analysis, and results of exploratory work to identify and automate significant indicators of students' attentiveness and the relationship between attentiveness and eye metrics. The experimental designs collected students' eye data and facial images as input for analysis.

The novel contribution of this thesis is the creation of the NiCATS data collection system and the establishment of the link between eye metrics and perceived facial attentiveness. This research adds to the body of knowledge regarding attention tracking in a classroom setting and seeked to validate previous research in the topic. This research could prove invaluable in shaping how classroom lectures and materials are structured and improving upon the feedback that institutions and professors receive regarding their teaching methods.

Based on the findings from the research, a link between perceived facial attentiveness and the amount per second and duration of both fixations and saccades was identified. It suggests that utilizing eye tracking with facial expressions could be combined when measuring attentiveness to create a more accurate predictive machine learning model than a machine learning model without this information. There appears to be no definitive type of attentiveness-predicting eye metric when referring to "per second" metrics and "average duration" metrics. Notably, saccades per second are the most reliable eye metric in terms of statistical significance and effect on attentiveness. Additional research is needed to more confidently state the potential of using eye tracking in the measurement of attentiveness, along with which eye metrics are most relevant in the prediction of attentiveness.

CHAPTER 7

DISSEMINATION OF RESULTS

The results presented in this thesis have been published and submitted in the following refereed conferences:

Accepted and Published:

1. Sanders, A., Allen, A. A., Walia, G., & Boswell, B. (2021). Non-Intrusive Classroom Attention Tracking System (NiCATS). Proceedings of the 2021 Frontiers in Education Conference. Presented at Lincoln, Nebraska, USA.

Under Review:

1. Sanders, A., Allen, A. A., Walia, G., & Boswell, B. (2022). Development of a Real-time Non-intrusive Classroom Attention Tracking System for Tracking Attention and Knowledge-Building Skills. 2022 Frontiers in Education Conference.

REFERENCES

A. Sanders, B. Boswell, G. S. Walia, and A. Allen, “Non-intrusive classroom attention tracking system (nicats),” *2021 IEEE Frontiers in Education Conference (FIE)*, 2021.

B. Anderson, “Stop paying attention to ‘attention,’” *WIREs Cognitive Science*, 2021.

B. T. Carter and S. G. Luke, “Best practices in eye tracking research,” *International Journal of Psychophysiology*, vol. 155, pp. 49–62, 2020.

D. Rosengrant, D. Hearrington, K. Alvarado, D. Keeble, N. S. Rebello, P. V. Engelhardt, and C. Singh, “Following student gaze patterns in physical science lectures,” *AIP Conference Proceedings*, 2012.

E. Lane and S. Harris, “Research and teaching: A new tool for measuring student behavioral engagement in large university classes,” *Journal of College Science Teaching*, vol. 044, no. 06, 2015.

J. A. Fredricks, P. C. Blumenfeld, and A. H. Paris, “School engagement: Potential of the concept, state of the evidence,” *Review of Educational Research*, vol. 74, no. 1, pp. 59–109, 2004.

J. Zaletelj and A. Košir, “Predicting students’ attention in the classroom from Kinect facial and body features,” *EURASIP Journal on Image and Video Processing*, vol. 2017, no. 1, 2017.

M. A. Dewan, M. Murshed, and F. Lin, “Engagement detection in online learning: A Review,” *Smart Learning Environments*, vol. 6, no. 1, 2019.

M. S. Young, S. Robinson, and P. Alberts, “Students pay attention!,” *Active Learning in Higher Education*, vol. 10, no. 1, pp. 41–55, 2009.

N. Veliyath, P. De, A. A. Allen, C. B. Hodges, and A. Mitra, “Modeling students' attention in the classroom using Eyetrackers,” *Proceedings of the 2019 ACM Southeast Conference*, 2019.

T. Tabassum, A. A. Allen, and P. De, “Non-intrusive identification of student attentiveness and finding their correlation with detectable facial emotions,” *Proceedings of the 2020 ACM Southeast Conference*, 2020.

W. James, “The principles of psychology,” 1910.

W.-H. Yun, D. Lee, C. Park, J. Kim, and J. Kim, “Automatic recognition of children engagement from facial video using Convolutional Neural Networks,” *IEEE Transactions on Affective Computing*, vol. 11, no. 4, pp. 696–707, 2020.

Z. Sharafi, T. Shaffer, B. Sharif, and Y.-G. Gueheneuc, “Eye-tracking metrics in software engineering,” *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 2015.

Z. Zhu, S. Ober, and R. Jafari, “Modeling and detecting student attention and interest level using wearable computers,” *2017 IEEE 14th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2017.

APPENDIX

NICATS STUDENT CLIENT TECHNICAL BREAKDOWN

The NiCATS Student Client is a Windows C++ program that is designed to collect data from a monitor-mounted eye tracker and webcam and send that data, along with screenshots of the screen, to a remote server. It includes a number of third-party libraries to help with functionality, as listed below:

1. OpenCV2, for the handling of image data, image encoding, video capture, and general image processing and manipulation. It was also used originally for image classification using cascade classifiers but was changed to facedetection library.
2. Libfacedetection, for the detection of faces in an image and for the bounding box of the faces.
3. Curl, for all HTTP and HTTPS data transfers to-and-from the server.
4. Tobii Stream Engine, for all communication with the Tobii Eye Tracker.
5. JSON for Modern C++, for the structuring of data so it may be sent to the server in a form for easy handling.

The NiCATS Student Client has a header file that declares the functions of the NiCATS Student Client cpp file, along with defining structs that are useful for structuring data. There is also a NiCATEnums header file that defines a few useful enumerations that are used as error codes, HTTP request methods, and indicators for Tobii within the main program.

The NiCATSEnums.h file defines the following enums:

1. NiCATSErrorType, which is a collection of possible errors the program might run into that is handled with other internal program logic. This includes (each of these has NiCATS_E_ prepended) the following:
    a. NO_ERROR, for when there is no error.
    b. LOAD_LIBRARY, for when the Win32 function Load_Library has an error.
    c. KEYBOARD_HOOK, for when the Windows Low Level Keyboard Hook fails.
    d. MOUSE_HOOK, for when the Windows Low Level Mouse Hook fails.
    e. GET_MESSAGE, for when Windows Message Loop receives a -1.
    f. THREAD_ERROR, for when an exception happens when creating any of the five threads.
    g. OPTED_OUT, for when a student has opted out of the NiCATS data collection.
    h. TERMINATE, for when a global flag has been set to terminate the threads.
    i. WRONG_HTTP_METHOD, for when an incorrect HTTP method has been set when sending an HTTP request.
    j. SERVER_IS_DOWN, for when the client cannot access the server due to it being offline.
    k. HEARTBEAT, for if an exception occurs when trying to periodically reach out to the server for information.
    l. NO_TOBII_EYETRACKER, for when no eye tracker is plugged into the computer.
    m. MALLOC, for when malloc fails.
2. NiCATSHttpRequestMethods, which is the collection of HTTP requests that the student client should be using. It consists of NiCATS_HTTP_GET and NiCATS_HTTP_POST. They are used depending on if the client is asking for data or sending data.
3. NiCATSTobiiIndicator, which consists of indicators to insert along with the Tobii eye gaze data as logical breaks within the eye gaze and screenshot data. These serve to assist the researcher in

indicating which data should be considered valid, invalid, or potentially invalid. The indicator enums are as follows:

a. NiCATS_Tobii_VALID, for when a student's input has been inactive for .5 seconds following input from the student, or if the time threshold of 15 seconds has been reached and if the face detection thread has detected a face within the last 15 seconds.
b. NiCATS_Tobii_INVALID, for when a student has inputted within the last .5 seconds, or if the time threshold of 15 seconds has been reached and if the face detection thread has not detected a face within the last 15 seconds.
c. NiCATS_Tobii_TIME_THRESHOLD_REACHED, for when the time threshold of 15 seconds has been reached.

The NiCATSStudentClient.h header file contains the following:

1. #include preprocessor statements. These include code that is used by the rest of the NiCATS student client program.
2. #define preprocessor statements. UNICODE is defined for the Win32 functions, and DETECT_BUFFER_SIZE 0x20000 for LibFaceDetection. For debugging purposes, either _DEBUG or NDEBUG is included, which either defines main or winmain in the main program. The purpose of that is to have a console window or not.
3. Structs. There are three structs defined: httpJobData, ScreenshotStruct, and EyeData.
    a. httpJobData has three members: a NiCATSHttpRequestMethods type httpRequestMethod member, std::string type httpRequestData member, and a std::string type endPoint member. The purpose of this struct is to be used by an HTTP job thread pool that takes the httpJobData structs from a queue and performs the actions specified.

   b. ScreenshotStruct has three members: Screenshot, epochTimestamp, and ForegroundWindowTitle. Screenshot is of type cv::Mat. epochTimestamp is of type double. ForegroundWindowTitle is of type std::string. The purpose of this struct is for easier handling of screenshot data so that the correct screenshot is sent to the server whenever the student inputs or if the 15 second time threshold is reached. This is explained in more detail in the ScreenGrabberAndEyeDataLoop function mentioned in NiCATSStudentClient.cpp.
   c. EyeData has three members: double timestamp, double currentEyeX, double currentEyeY. This allows for easier grouping of relevant eye data.
4. Function Declarations. The functions will be discussed in the NiCATSStudentClient.cpp section.

The NiCATSStudentClient.cpp #includes the NiCATSStudentClient header file and contains the primary functions and variables for the student client program. The following subsections will cover the variables and the functions:

1. Variables: The program contains a large number of global variables, grouped by purpose.
   a. Low Level Hook variables: There are two Windows HHOOK variables that hold the handle to the low level hooks. There is the keyboardHook and the mouseHook, each responsible for that specific input type. The purpose of the low level hooks is to split the eye gaze and screenshot data by periods of input so that the screenshot data is relevant to the eye gaze data. If a student were to click or page down on a webpage, for example, then the screenshot data they're associated with would be irrelevant.
   b. Endpoint variables: There are a number of std::string variables that hold the http/https endpoint that the program needs to request from. They are updated at runtime in the GetCorrectEndpoint and UpdateEndpoints functions. The purpose for this is to

dynamically find the correct endpoint to use based on if the server is running on the same machine as the student client. Since this is a common occurrence when developing the program, this is useful for quickly finding where to send data without having to hardcode the endpoints every time the program is updated.

c. Keylist variable: The KEY_LIST std::vector holds a predefined list of common “page-changing” keys that the Low Level Keyboard Hook will search on every keypress. Specifically, it is the arrow keys, page up/down, home/end, escape and return keys. This is used with the screenshot and eye gaze data for splitting up the data.

d. Recording Session variables: There are two integer variables dedicated to keeping track of the current recording session. They are used in each thread to determine if they should still be running, and they are updated whenever the program sends out a heartbeat to the server.

e. Variables related to thread pools, mutexes, and an http job queue: These are all related to concurrent programming within the program. The thread pool is used for non-blocking HTTP requests. The http job queue mutex, thread pool mutex, Tobii subscribe mutex, and eye data mutex are all used to ensure data integrity and prevent race conditions between threads. The http job queue is used with the thread pool for sending the http requests. Since the data collected in this program needs to be time-precise, non-blocking HTTP requests are important for ensuring timestamps are valid and accurate.

f. Tobii API variables: These are used with the Tobii Eye Trackers to handle the internal name of the device and to subscribe to the eye gaze data feed from the device.

g. Screen size variables: For the correct coordinates of eye gaze points relative to the screen, the screen size is stored. Data from the eye trackers are given in a 0-1 range, so they are multiplied by the screen size to get the actual location of the gaze point on the screen.

h. "Previous" timestamp variables: The timestamp of when the last time the face was detected and the timestamp of when the last time an input was detected are both stored in variables. These are used in relation to the VALID, INVALID, and TIME_THRESHOLD_REACHED NiCATSTobiiIndicators discussed in the NiCATSStudentClient header file section.

i. Base64 table: This allows for base64 encoding of images so they can be sent to the server in a more supported format.

j. Eye Gaze data structures variables: There are a few variables dedicated to holding the data outputted from the eye tracker and handling relevant information about it. There are two std::vector<EyeData> variables for handling the raw gaze points and the moving average gaze points. The moving average gaze points are individually averaged with the previous nine gaze points so that a "smoother" gaze point vector can be available for analysis. Since the Tobii Eye Tracker 4c models are somewhat jittery, it can be slightly unreliable to calculate eye metrics using the raw gaze points. There are also variables for the storage of VALID, INVALID, and TIME_THRESHOLD_REACHED Tobii indicators. VALID and INVALID indicators are stored in their own std::vector<double> variables and the TIME_THRESHOLD_REACHED is stored as a boolean variable.

2. Functions: The program has a number of functions that are used to accomplish various tasks. They are listed as follows:

   a. void gaze_point_callback(tobii_gaze_point_t const* gaze_point, void*): This function is called whenever the Tobii Eye Tracker has gaze points that are ready to be processed. It is a callback function that is user implemented and is passed as an argument to the Tobii Stream Engine API's tobii_gaze_point_subscribe function. In NiCATS, it takes the raw gaze point data and pushes it onto the raw gaze point vector. In addition, it pushes the raw gaze point data onto a 10-capacity vector static variable that is used to calculate a moving average gaze point for the moving average gaze point vector. All of the data manipulation in this function occurs while locking the EyeDataMutex mutex so that no race conditions occur while storing the data. Because the Tobii Eye Tracker 4c model collects data at a rate of 90hz, and it was decided to store a max of 15 seconds of data as a threshold, a max of 90*17 eye data points is stored within the vector. The reason for 17 seconds instead of 15 is to give a small buffer window due to hardware inaccuracies when measuring timestamps.

b. void InsertTobiiIndicator(NiCATSTobiiIndicator indicator, double indicator_time): This function is used by the Screengrab/Eye Data portion of the program to insert TobiiIndicators into the Tobii indicator storage variables. The purpose of this is for researchers to grab the relevant eye data from a large sequence of eye data.

c. void url_receiver(char const* url, void* user_data): This function is used by the Tobii API to enumerate through all connected eye trackers and write their url name.

d. void StopTobii(): This function is used whenever the recording session has ended or is paused. It locks the TobiiSubscribeMutex and unsubscribes from the Tobii gaze point subscription, which stops the collection of eye data. This function also clears the currently stored eye data variables so that a clean slate can be used for the next recording session. The counterpart to this function is InitTobii.

e. NiCATSErrorType RunTobiiLoop(): This function is the sole job of the Tobii thread that is created in the main. It is called by the RunTobii function. It locks the TobiiSubscribeMutex and runs the boilerplate code for starting the Tobii Eye Tracker (runs Tobii Stream Engine's tobii_api_create, tobii_enumerate_local_device_urls, and tobii_device_create). Then, it runs a while true loop with the primary focus of processing the Tobii callback functions. It is important to note that it does not process the callbacks if there is no current recording session occurring (recording session ID is -1). If any exceptions occur, it attempts to catch them and continue running the loop. If the program is told to terminate, the Tobii API is called to clean up and delete any jobs related to the eye tracker. If no eye tracker is discovered on the system, the function returns with the NiCATSErrorType NiCATS_E_NO_TOBII_EYETRACKER.

f. NiCATSErrorType RunTobii(): This function is what is run by the Tobii thread that is created in the main. It contains a while true loop that simply calls the RunTobiiLoop function and ends if the RunTobiiLoop returns with anything other than NiCATS_E_NO_TOBII_EYETRACKER. If any exception occurs within RunTobiiLoop, the function attempts to rerun the function after a 100 millisecond delay (to prevent a while true loop that has high CPU usage).

g. void InitTobii(int NumberOfSeconds): This function is used by the heartbeat thread to lock the TobiiSubscribeMutex and subscribe to the gaze point data stream. It calls the subscribe function with the gaze_point_callback function as the callback. The NumberOfSeconds argument is used to indicate the total number of gaze points the gaze point vector should store, but it is set to 15 for now. If no eye tracker is plugged into the computer, the function does not attempt to subscribe to the data stream. The counterpart to this function is StopTobii.

h. json GetEyeDataJson(double screenshotTimestamp, std::string foregroundwindowtitle): This function is used by the Screengrabber thread to get the current eye data so that it may be packaged with the screenshot and sent to the server. This function locks the EyeDataMutex (which is used by the Tobii thread) and creates a json object with all of the current eye data variables. This includes the Tobii eye tracker device url, the VALID

Tobii indicators, the INVALID Tobii indicators, the boolean TIME_THRESHOLD_REACHED indicator, the raw gaze points vector, and the moving average gaze points vector. It also includes the arguments that are coming into the function which are the timestamp of the screenshot and the current foreground window title (both created in the screengrabber thread). Before returning, the function empties all of the eye data variables and sets the TIME_THRESHOLD_REACHED indicator variable back to false. The function then returns the created json after unlocking the mutex.

i. std::string utf8_encode(const std::wstring& wstr): This is a helper function that converts a std::wstring to a std::string using utf8. Because UNICODE is defined, any string returned by a Win32 function will be a wstring, so this function is used to convert it back to a normal string.

j. std::string base64_encode(const unsigned char* src, size_t len): This function is used to encode the image binary data into base64 so that no data is lost during the transmission to the server. It is used with both the screenshot data and face image data.

k. double getEpoch(): This is a helper function to get the current epoch timestamp. It uses the Win32 function GetSystemTimePreciseAsFileTime to get the current timestamp. It is millisecond accurate.

l. std::string getTimeStr(): This function is used to get a formatted timestamp string that can be interpreted by the server and database as a proper timestamp. It is millisecond accurate.

m. size_t writeFunction(void* ptr, size_t size, size_t nmemb, std::string* data): This is a helper function used by Curl to write data to the data string. Because data is chunked during IP transmission, this function is necessary to write all of the received data to the string.

n. void ImageCleanup(HWND hWnd, HBITMAP hbmScreen, HDC hdcMemDC, HDC hdcWindow): This function is used by the GetScreenshotMat function to clean up Win32 device context handles.

o. cv::Mat GetScreenshotMat(): This function is used to get the current screenshot of the screen. It returns an OpenCV2 n-dimensional dense array that contains the image. It uses Win32 gdi function calls to return the screenshot. It is used by the screengrabber thread.

p. std::string GetCorrectEndpoint(): This function is used to update the endpoint variable. It is primarily used to make development easier as it tries to detect if the server is running on localhost or remotely. It works by attempting to send an HTTP request to the ping endpoint, first on localhost and then to the remote location. If neither endpoint responds, then the function returns “server is down”. If either endpoint responds, the server returns the url of that endpoint.

q. NiCATSErrorType UpdateEndpoints(): This function is the first thing called in the main and it updates the global endpoint variables. It accomplishes this by first calling the GetCorrectEndpoint function to get the correct endpoint and then updates the variables with the correct endpoint as the prefix. For example, after returning from GetCorrectEndpoint, HEARTBEAT_URL_ENDPOINT is set to the correct endpoint concatenated with “/heartbeat”. If the server is down, the function returns with the NiCATSErrorType NiCATS_E_SERVER_IS_DOWN, which is handled by the caller.

r. NiCATSErrorType SendHttpRequest(NiCATSHttpRequestMethods httpRequestMethod, std::string httpRequestData, std::string endPoint, std::string& response_string): This function takes in the arguments and sends out the HTTP request with the passed-in information. It uses the Curl library to accomplish this.

s. void AddHttpJob(NiCATSHttpRequestMethods httpRequestMethod, std::string httpRequestData, std::string endPoint): This function gets called by the screengrabber/eyedata and face image threads to enqueue an HTTP job for the HTTP job thread pool. The purpose of this is to allow the screengrabber/eyedata and face image threads to work without blocking. The function locks the HttpQueueMutex and pushes the passed-in information to the HttpQueue as an httpJobData struct. It then wakes up one of the asleep HTTP job threads to pop the job off the queue and send it to the server.

t. void HttpLoopFunction(): This job is what the HTTP job threads run so they can allow the other threads to send HTTP requests without blocking. When notified that an HTTP request should be sent out, one thread wakes up and pops the job off the HTTP job queue and calls SendHttpRequest with the job's information.

u. void SendScreenshotAndEyeInfo(cv::Mat screenshot, json eyeTrackerData, int rcid, int computer_number): This function gets called by the screenshot and eye data thread to send the eye data and screenshot to the server. It creates a json object with the current recording session ID, the student's account name, the computer number, the current timestamp as a string (with the getTimeStr function), the base64-encoded screenshot, and the eye tracker data that gets created by GetEyeDataJson. It uses AddHttpJob to create the HTTP job request so the HTTP job thread pool can handle sending the request without the thread blocking.

v. void SendFaceImage(cv::Mat webcamimage, cv::Mat faceimage, int rcid, int computer_number): This function gets called by the face grabber thread to send the face image to the server. It creates a json object with the record session ID, the student's account name, the computer number, the current timestamp as a string (with the getTimeStr function), the base64-encoded webcam image, and the base64-encoded face image. The face image is a subset of the webcam image with just the face of the student, or empty if no face was detected. The face gets detected using the LibFaceDetection library as part of the face grabber thread prior to this function being called. After creating the json, it uses AddHttpJob to create the HTTP job request so the HTTP job thread pool can send the data to the server without blocking the current thread.

w. bool CheckOptStatus(): This function is used to check if the current student is opted-in to NiCATS and prompts them to opt-in if they aren't. It sends an HTTP request to the server with the student's account name to check if they are already in the database. The function returns true if the student is opted-in to the database or false if they choose to opt-out.

x. std::string GetForegroundWindowTitle(): This function returns a string of the current foreground window's name. It uses the Win32 function, GetWindowText, to receive a wide string of the name. It then uses the utf8_encode helper function to convert the wide string to a regular string, which is then returned.

y. bool CheckForRecentUserInputActions(): This function returns a boolean that represents if the user has not inputted any action within the last 0.5 seconds. It uses the global "previous" timestamp variable that represents user input and the current timestamp to determine if the user has inputted any actions.

z. __declspec(dllexport) LRESULT CALLBACK LowLevelKeyboardProc(int code, WPARAM wParam, LPARAM lParam): This is the callback function that gets used by the Windows' Low Level Keyboard Hook on each keyboard input. Importantly, it checks if the current key being pressed is within the keylist vector to determine if it should update the "previous" timestamp variable that represents user input. If the input is contained within the keylist vector, the timestamp variable, MouseKeyboardLastInputTimestamp, is updated to the current time.

aa. __declspec(dllexport) LRESULT CALLBACK LowLevelMouseProc(int code, WPARAM wParam, LPARAM lParam): This is the callback function that gets used by the Windows' Low Level Mouse Hook on each mouse input. It checks if the current mouse input is anything other than a mouse move. If it is, then the MouseKeyboardLastInputTimestamp variable is updated to the current time.

bb. NiCATSErrorType MessageLoop(): This is the Windows message loop that is required to process the messages that the current window is receiving. The reason for its inclusion is that it is needed so the low level keyboard and mouse hook messages can be processed.

cc. NiCATSErrorType InputListeners(): This is the function that the input listeners thread is created to run with. It establishes the mouse and keyboard low level Windows hooks and then runs the MessageLoop function so they can be processed. It uses the SetWindowsHookEx functions with the LowLevelKeyboardProc and LowLevelMouseProc functions as arguments. A small note is that the function normally requires a dynamically linked library (dll) to be able to run, but it is "circumvented" by treating the current executable program as a dll so that the low level callback functions can be used.

dd. NiCATSErrorType Heartbeat(): This function is what the heartbeat thread runs. It has a while true loop that attempts to send an HTTP request to the server with the current building number and room number. If there are no issues, the server sends back what the client's recording session ID should be. If the recording session ID is not -1 and is different from what the client already has, then the heartbeat thread runs InitTobii so the gaze point data stream can be subscribed to from the Tobii eye tracker. If the server sends that the record session ID should be -1, or if there are any errors communicating with the server, the record session ID is set to -1 and StopTobii is called so that the Tobii gaze point data streamed is unsubscribed from.

ee. NiCATSErrorType FaceGrabberLoop(): This function is what is used by the face grabber thread for the capturing, cropping, and sending of face data. It uses OpenCV2 to create a video capture and then captures the face every 5 seconds. The reason to capture the face every 5 seconds instead of continuously is that a continuous video stream of the student's face provides a marginal benefit compared to static images when determining attentiveness (Dewan et al., 2019) and it also saves bandwidth, storage space and processing power. It only captures when the recording session is in progress (record session ID is not -1) and if the time difference between when the last face image was captured and now is greater than or equal to 5 seconds. To capture the student's face, the thread grabs the entire webcam image and then attempts to crop out a smaller image of the student's face using the LibFaceDetection library. If no face is detected within the webcam image, the face image is set to null. Regardless of if the face is detected or not, both the face image and the entire webcam image are sent to the server using the SendFaceImage function. If a face is detected, the LastFaceImageTimestamp timestamp variable is updated, which is used with the screengrabber and eye data thread to insert either a VALID or INVALID Tobii indicator with the eye data if the time threshold was reached.

ff. NiCATSErrorType FaceGrabber(): This thread is what is run by the face grabber thread. It is a while true loop that calls the FaceGrabberLoop function. If any exceptions occur within the FaceGrabberLoop function, the FaceGrabber function attempts to run the

FaceGrabberLoop function again after a 100 millisecond delay (to prevent a while true loop that has high CPU usage).

gg. NiCATSErrorType ScreenGrabberAndEyeDataLoop(): This function is used by the ScreenGrabberAndEyeData thread to handle the sending of screenshots and eye data. It contains a while true loop (on a 100 millisecond delay to prevent high CPU usage) that attempts to send the current eye data and screenshot. It sends the eye data and screenshot data together so that the eye data has context on what the student is looking at. To prevent situations where the student changes what they're looking at (which would essentially invalidate their eye data up to that point), the latest iteration screenshot and the previous iteration screenshot are both stored in a vector. Once the logic of the function decides that the screenshot and eye data should be sent, only the oldest of the two screenshots are sent so that the eye data remains contextualized with what the student was looking at. The logic of the function is as follows:

  i. If a threshold of 15 seconds has passed since the previous screenshot and eye data were sent, send a screenshot with the eye data.
  ii. If the previous screenshot was taken and sent more than a small threshold of 0.5 seconds prior, and the user has inputted any new actions (if the previous iteration of the while loop did not have any user input and the current iteration of the while loop did have user input), send a screenshot with the eye data.
  iii. In addition to sending the data, certain Tobii indicators are inserted at various points. If the threshold of 15 seconds has passed since the previous screenshot and eye data were sent, a TIME_THRESHOLD_REACHED Tobii indicator is inserted and also a VALID or INVALID indicator depending on if the LastFaceImageTimestamp variable is recent or not, respectively. If the previous

screenshot and eye data was sent more than 0.5 seconds prior and the user has inputted any new actions, the INVALID Tobii indicator is inserted with the eye data. The reason to insert the INVALID indicator is that if a student is scrolling/clicking/any other action, their eye data is probably not valid since they are still adjusting to the new content on their screen. If the user has not inputted any new actions, but on the previous iteration of the while loop they did input actions, a VALID Tobii indicator is inserted with the eye data, because the student has probably adjusted to the new content on their screen. For any of the Tobii indicator insertions, the InsertTobiiIndicator function is used with the proper NiCATSTobiiIndicator argument. Once the eye data is ready to be collected, the GetEyeDataJson function is called and the return is stored in a json variable. The json variable is used with the screenshot in the SendScreenshotAndEyeInfo function so it can be encoded and sent to the server.

hh. NiCATSErrorType ScreenGrabberAndEyeData(): This function is what is run by the ScreenGrabberAndEyeData thread. It contains a while true loop that calls the ScreenGrabberAndEyeDataLoop function. It also attempts to rerun the function if any exceptions occur in it (with a 100 milliseconds delay to prevent high CPU usage).

ii. void StartupHttpWorkerThreads(): This function gets called to create the HTTP worker threads. They are each added to the HTTP thread pool so that other functions can send HTTP requests without blocking. It creates the same number of threads as the current CPU has support for.

jj. void ShutdownHttpWorkerThreads(): This function wakes up all of the HTTP worker threads and joins them back to the main thread. It then clears the HTTP worker thread pool. It is used for cleanup within the main thread.

kk. int main() or int WINAPI wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ PWSTR lpCmdLine, _In_ int nCmdShow): The entry point of the program initializes every other part of the program. It starts by initializing Curl and starting the main while true loop. The while true loop attempts to update the endpoint variables to the correct values. It calls the UpdateEndpoints function. If the function returns the NiCATSErrorType NiCATS_E_SERVER_IS_DOWN, then the while loop sleeps for 5 seconds and tries to update the endpoints again. If the server is not down, it then checks the opt status of the student by calling CheckOptStatus. If the student is opted-out, then the program ends, since they do not consent to having their data collected. Otherwise, StartupHttpWorkerThreads is called so that the HTTP worker thread pool is created. Each of the five threads are then created on their respective functions. The input listeners thread is created on the InputListeners function, which creates the low level Windows hooks for the mouse and keyboard so that the screenshots and eye data are collected correctly. The heartbeat thread is created on the Heartbeat function, which polls the server every 5 seconds to check what the current recording session ID should be and if the Tobii gaze point data stream should be subscribed to. The face grabber thread is created on the FaceGrabber function, which is responsible for capturing the student's face every 5 seconds and sending it to the server. The screengrabber and eye data thread is created on the ScreenGrabberAndEyeData function, which is responsible for controlling the sending of screenshots and eye data. The Tobii thread is created on the RunTobii function, which is responsible for collecting the gaze points produced by the eye tracker. After creating the five threads, the main attempts to join back all five functions, which causes the main to block. If the program decides to terminate, all threads are joined back to the main and ShutDownHttpWorkerThreads is called to clean up the HTTP worker threads.