DEVELOPING INFRASTRUCTURE FOR REINFORCEMENT LEARNING IN ASSISTMENTS

by

Peyton Grant

Aditya Kumar

Date: 3/18/21

Abstract

This project is an extension of the Reinforcement Learning Service (RLS) designed for the ASSISTments learning platform. ASSISTments is an organization that is committed to improving student learning using online software. The RLS is a service that is called whenever a student requests additional instruction or tutoring. The RLS takes different features from the student and the problem then recommends a related video or hint based on those features. Based on if the student successfully answered the problem correctly afterward, the RLS then learns from the results and refines its recommendation algorithm. For our project, we are mainly concerned with what happens when the RLS receives a request and how it communicates with the rest of ASSISTments. We wrote code that will use the information from a student or problem, provide a recommendation with a given model, and will write a detailed log that can be analyzed later.

Acknowledgements

We want to thank multiple National Science Foundation grants (e.g., 1917808, 1931523, 1940236, 1917713, 1903304, 1822830, 1759229, 1724889, 1636782, 1535428, 1440753, 1316736, 1252297, 1109483, & DRL-1031398), as well as the US Department of Education for three different funding lines; a) the Institute for Education Sciences (e.g., IES R305A170137, R305A170243, R305A180401, R305A120125, R305A180401, & R305C100024), b) the Graduate Assistance in Areas of National Need program (e.g., P200A180088 & P200A150306), and c) the EIR. We also thank the Office of Naval Research (N00014-18-1-2768) and finally Schmidt Futures as well as a second anonymous philanthropy. Without funding from these parties, our project wouldn't be possible.

We would like to thank our mentors Ethan Prihar and Zekun Dai. These two individuals put a lot of time and effort into helping us achieve our goals and ensuring that we had gained valuable knowledge and experience throughout this project. We are thankful for their aid along with other various members of the ASSISTments staff who have offered suggestions and feedback. We want to thank the head software engineers Chris Donnelly and David Magid for providing constructive criticism towards the structure of the code. Joseph St. Pierre is another software engineer we want to mention as he wrote a function that connected problem identifiers to skill identifiers. We also would like to thank Thanaporn "March" Patikorn, who helped integrate our code into TeacherASSIST. Lastly, special thanks to Neil Heffernan for advising us and allowing us to work on this project at ASSISTments.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 The ASSISTments Platform

ASSISTments is an online platform that focuses on mathematical instruction. The primary goals of ASSISTments are to empower teachers, tutor students, and perform research. Through the platform, teachers can assign homework to their students as well as get quick feedback to understand what areas their class is struggling in. While the students work on their assigned problems, ASSISTments offers tutoring strategies to help students learn from difficult problems they encounter. While making mathematical instruction more convenient, ASSISTments also collects data and uses that data to perform research to improve the quality of its mathematical instructions to help students in their learning endeavors (Heffernan, Neil T. and Cristina Lindquist Heffernan).

1.2 Personalized Learning and RLS

Recently, ASSISTments has been looking into personalized learning to aid the students on the platform. According to Peggy Grant and Dale Basye, personalized learning is a method of instruction that considers the strengths, needs, and interests of an individual student and tailors the instruction to take advantage of those qualities (Grant, P. and D. Basye). While students work on problems via ASSISTments, they seek help using an online search engine if they're confused about a particular problem. During this search, a student will find information and videos are abundant online. This may prove difficult for finding the appropriate resource to help clear that initial confusion. Instead of students struggling to seek instruction, ASSISTments wants to incorporate personalized learning by providing additional instruction tailored for the student to clear up any confusion they might encounter while problem-solving.

To accomplish this, ASSISTments began the development of the Reinforcement Learning Service (RLS). The RLS will recommend a video related to the problem the student is working on. With this system, the RLS can personalize a student's learning experience by recommending videos with specific characteristics that aid the student in the learning process. For example, one

student may learn more from videos where the speaker has a passionate tone, while another may learn more from videos where the speaker has a calm tone.

1.3 Request Manager

The main focus of our project is what happens when the ASSISTments tutor makes a request for the RLS to provide personalized learning. This portion of the RLS is what we called the Request Manager. Our responsibilities included designing the class diagram, building the backend code in Java, and integrating the project with the ASSISTments tutor.

The goal of the project was to have an implementation ready to be used and incorporated into the RLS. If the Request Manager works as intended, the RLS will gain more knowledge that will help improve the personalization of the service. The achievement of this goal would mean that ASSISTments would be one step closer to providing personalized additional instruction for students on the platform.

2. Background

2.1 ASSISTments

ASSISTments is an online learning platform designed for tutoring and helping students in grade school. As stated earlier, the primary goals of ASSISTments are to empower teachers, tutor students, and perform research. Teachers can take advantage of the ASSISTments platform to easily track their student's performance and learn what areas they are struggling in. With the ability to assign problems and automatic grading, teachers are empowered by this platform (Heffernan, Neil T. and Cristina Lindquist Heffernan). For students using ASSISTments, they can work on assigned problems and gain additional aid through the ASSISTments tutor (Figure 1). The ASSISTments tutor can handle requests for tutoring strategies on math problems by asking TeacherASSIST (more detail in Section 2.3) for tutoring content such as hints and explanations (Patikorn, Thanaporn and Neil T. Heffernan).

Figure 1. Screenshot of ASSISTments Tutor

2.2 Personalized Learning

Typical methods of instruction in K-12 classrooms involve each student following a cookie-cutter approach to learning new material. Personalized learning separates itself from the standard cookie-cutter approaches to learning by providing instruction tailored to the individual student's strengths (Bulger, Monica). There are many studies regarding personalized learning and its effectiveness with varying results.

An example of a study that successfully supports personalized learning was performed by a group of researchers working at ASSISTments. In this study, four randomly selected classes composed of eighth-grade students participated in completing a pre-test and problem sets. While completing these problem sets, students in a class had access to one of two learning methods. One of these methods was a detailed step-by-step breakdown of a problem that leads to the solution, while the other method showed only the solution. After finishing the problem sets, the students of each class were then assigned to complete a post-test which can be compared to the pre-test score to measure learning effectiveness. As the researchers of this experiment performed their analysis, they ended up dividing the students into two groups based on their scores on a

standardized math test: high proficiency students and low proficiency students. The results of this study showed that highly proficient students learned much more effectively through being shown the solution compared to being given the step-by-step breakdown. In contrast, low proficiency students learned much more effectively through the step-by-step breakdowns compared to being shown the solution (Razzaq, Leena and Neil Heffernan). This study demonstrates that students can learn more effectively by providing the appropriate learning method which can be achieved through personalized learning.

While some studies support personalized learning, some studies conclude personalized learning to be not as effective. A study performed by two researchers, Massa and Maya, highlights this. In this study, Massa and Maya provided computer-based electronics lessons where the students were categorized to be visual learners or verbal learners. Massa and Maya provided additional help during these lessons that came in either text or illustrations. Note that both these methods cater to the learning styles of one group of students. Students from both groups were exposed to these methods of additional help during instruction. Their results showed no significant difference in performance even when the help was tailored to the student's personal learning preference. In the case of Massa and Maya's study, there was no proper evidence to provide separate personalized instruction for visualizers and verbalizers (Pashler, Harold et al).

While these studies show different results for the effectiveness of personalized learning, it is an area of research that can be further explored. Online learning platforms like ASSISTments have the potential to incorporate personalized learning and design studies to test its effectiveness. By taking advantage of student profiles and machine-learning algorithms, it is possible to analyze an individual's strengths and respond with personalized instruction. Each student on the platform is unique from their age, gender, ethnicity, academic performance, and learning preferences. Taking these characteristics into consideration with the Request Manager, we can provide personalized instruction for students on the ASSISTments platform. Researchers can also design experiments around the personalized instruction to assess its effectiveness. Even if

we find personalized instruction is ineffective, the Request Manager will still provide a form of instruction that is the most helpful for the student.

2.3 TeacherASSIST

TeacherASSIST is a feature recently implemented inside of the ASSISTments platform. It is a crowdsourcing system designed for students to get on-demand assistance created from teachers outside of their class. This on-demand assistance is tied to individual problems and it will be referred to as "tutor strategies" throughout this paper. Teachers using TeacherASSIST have the freedom to create tutor strategies for the math problems they assigned to their students. As for the crowdsourcing aspect of TeacherASSIST, teachers who regularly create tutor strategies and correct their mistakes for their students can re-distribute their tutor strategies to other students through TeacherASSIST given their permission (Patikorn, Thanaporn and Neil T. Heffernan).

TeacherASSIST currently supports three different types of tutor strategies: hints, scaffoldings, and explanations. Hints are a series of helpful messages that provide students with some additional information to aid in solving the problem. Scaffoldings are the original problem decomposed into smaller steps to guide students towards the answer. Explanations are detailed breakdowns on how to solve the problem and reveal the answer. While all three tutor strategies are supported, teachers can only create hints and explanations through TeacherASSIST. The complexity and time for creating a scaffolding went against the convenience aspect of TeacherASSIST (Patikorn, Thanaporn and Neil T. Heffernan).

While students can request one of many tutor strategies for individual problems through TeacherASSIST, the selection of the tutor strategy is random in the system's current state. To incorporate personalized learning, we needed to implement the Request Manager into TeacherASSIST. Upon request for a tutor strategy, the Request Manager will select the tutor strategy by considering all the available tutor strategies from TeacherASSIST. The Request Manager then looks at an individual student's traits to make a personalized decision. Once the decision is made, the Request Manager will inform TeacherASSIST which strategy to provide to

the student. We will later discuss how we implemented the Request Manager into TeacherASSIST in Section 3.3.

2.4 Reinforcement Learning

Reinforcement learning is one of the main approaches in machine learning. The goal of reinforcement learning is for the model to learn how to make informed decisions on what action to take in a complex environment to maximize a possible reward. Consider the situation where a mouse must navigate through a maze. The faster he can finish the maze, the more food he will receive as a reward. The mouse wants to receive the highest reward he can, so he will learn from past maze runs to get through the maze as fast as possible. Typically, in the beginning, the model will start by making seemingly random decisions. The purpose of this is to gather more information about the environment. Each time that the model makes a decision, it will receive a reward or penalty. With the intention of trying to maximize the reward, the model will look at what factors had previously resulted or did not result with the reward and will consider those factors when it makes its next decision.

The typical reinforcement learning framework can be seen in Figure 2. The process begins when the model observes the current state of the environment. Afterward, the model will decide on an appropriate action that affects the environment. The model will then observe if the action produced the desired reward and the current state of the environment. Based on those observations, the model will learn and then use them to decide the next action. This cycle will then repeat.

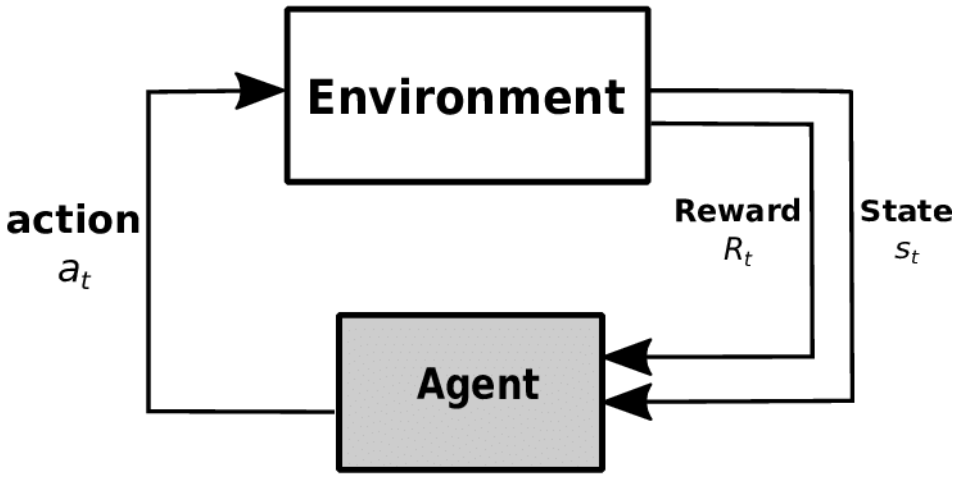


Figure 2. General Reinforcement Learning Model

A theoretical scenario where a reinforcement learning strategy could be applied would be the multi-armed bandit problem. In this problem, there is a gambler at a row of slot machines (one-armed bandits), each with different unknown probabilities of winning. To maximize his winnings, he must choose which machine to play each time to balance getting information and profiting from the machines. How a reinforcement learning strategy might try to handle this problem would be to first explore information about the machines using a certain strategy or just by random trial and error. Then, when it will use the known information to try to exploit the machines to maximize the reward. However, different reinforcement learning strategies may have different ideologies for the exploration vs. exploitation tradeoff.

When a student asks for additional instruction for a problem, the RLS in ASSISTments will provide a personalized video. At first, the model for RLS will recommend videos and strategies randomly. Then, the student will answer the next problem. If the student answered the problem correctly, the model will receive a reward. If the student didn't answer it correctly, it will not. When analyzing which recommended videos resulted in a reward and which do not, the model will observe the features of the student, the problem, and the video. From those features, the model will learn which videos are most relevant when recommending a video.

2.5 Ideas Lab Grant

When observing the number of STEM professionals, there is a clear underrepresentation of certain populations. Concerning this, if a student from these underrepresented populations were to watch an instructional video by searching Google, it is likely that the video that they manage to find may not resonate with them as it would with other students. To help solve this issue, ASSISTments proposed Ideas Lab. The Ideas Lab proposal focused on a process called Precision Learning to provide personalized learning to students. As seen in Figure 3, the features from the students and videos would be taken, and using those features, the Precision Learning process would provide personalized recommendations. The recommended videos would be labeled as "additional instruction".

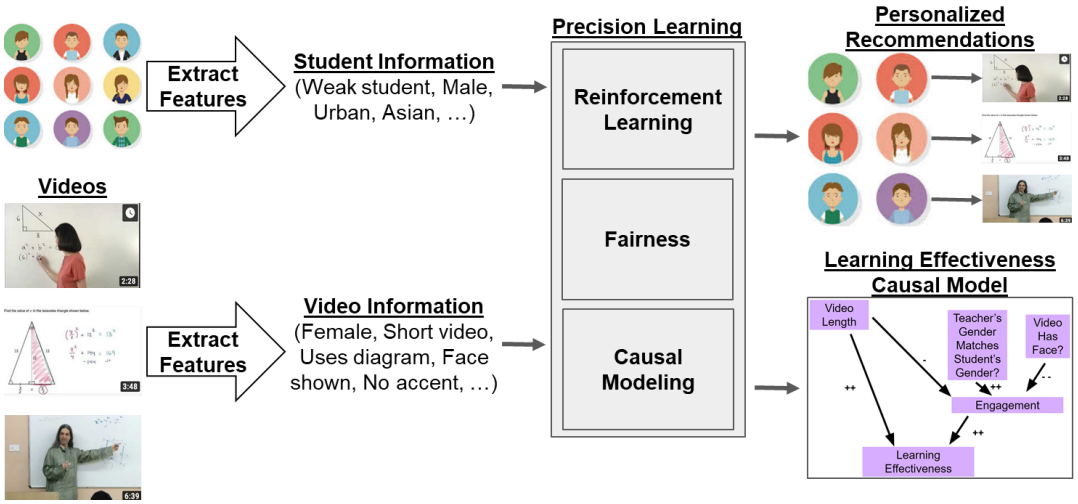


Figure 3. Ideas Lab Process Overview

The first phase of the Ideas Lab project involved collecting a portion of educational videos (approx. 1,300) and developing video recommendation algorithms for ranking educational value and for fairly recommending based on student features. The second phase of the project will involve collecting many more educational videos (approx. 100,000) and starting experimenting with real students to improve the algorithms. The third phase will involve using the improvements of the algorithm to implement the service in ASSISTments. Then, based on the effectiveness of the implementation, the model will be constantly refined.

A visual mock-up of what the additional instruction button will look like when implemented can be seen in Figure 4. When the student clicks on the button, a personalized video related to the skill in the problem will be provided to the student.

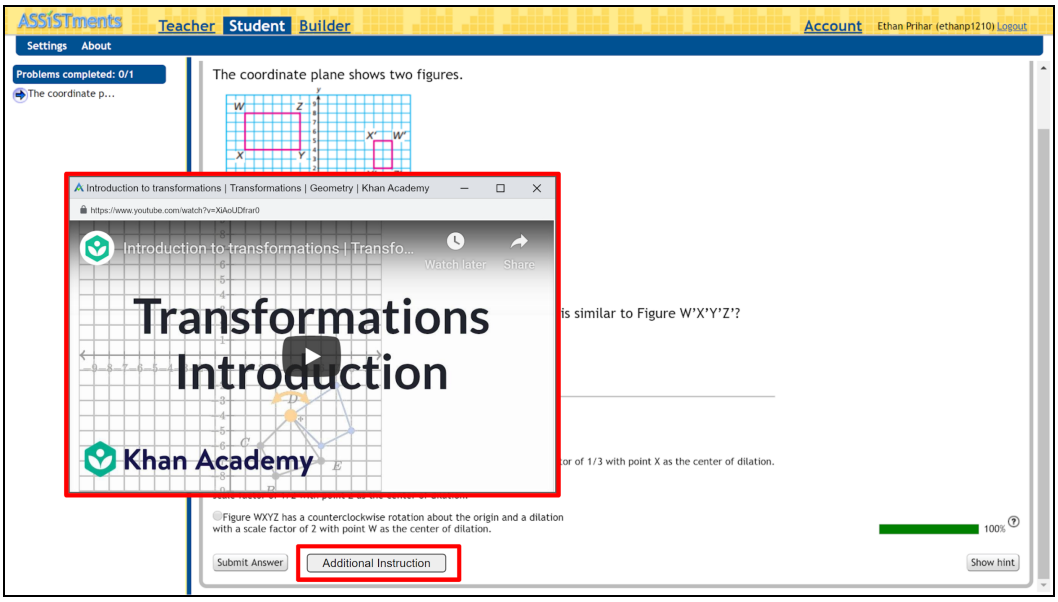


Figure 4. Mock-Up of Student Perspective when Implemented

3. Methods

3.1 Designing and Coding the Request Manager

We started the design process for the Request Manager by attempting to define and understand the purpose of our project and where it fits into the RLS. The Request Manager receives a "request" which contains a list of tutor strategies as well as student and problem IDs and then returns the tutor strategy that is predicted to be the most helpful to the student. This process involves taking the features of the student and problem, choosing an RLS model, and returning the tutor strategy or additional instruction.

Student features include their gender, ethnicity, and average scores. Meanwhile, problem features include the type of problem and the frequency of the correct answer is given. In the context of the RLS, a model means the algorithm being used to calculate a score for each tutor strategy which takes in any relevant features it may need. As stated previously, tutor strategies are on-demand assistance associated with each problem. There are multiple types of tutor strategies including hints, explanations, etc. In this stage of development, we considered an "additional instruction" a special type of tutor strategy that provided a video on the assigned topic and is related to a Common Core skill code. The features for each student, problem, tutor strategy, and additional instruction are stored in databases that can be accessed through their respective data access object (DAO). These databases are updated nightly in the RLS to get the most recent information and DAOs can be implemented in Java programming to easily obtain information from a database.

Once we understood what our task was, we created a class diagram. We decided that the best way to structure the Request Manager would be to use different sub-managers for the different tasks in the process. We created three sub-managers: the Context Manager, the Model Manager, and the Log Manager. The main purpose of the Context Manager is to obtain relevant features and then form context objects which are sent back to the Request Manager. The main purpose of the Model Manager is to use the given context to select one of the available RLS models, then use that model to give the predicted score for a tutor strategy. The Request Manager then compares the predictions of each tutor strategy and then picks the one with the highest score. Finally, the main purpose of the Log Manager is to write a log entry to the RLS logs which details all of the relevant information that can be used to revise the RLS models in the future.

3.1.1 Request Manager

Once the core functionality of the Request Manager was designed, revisions were made to fit expectations. We realized that the Request Manager obtained different types of context from the Context Manager and needed a convenient object to store all that context. Thus, an object called the RequestInfo was created with the purpose of holding all the relevant context that is retrieved from the Context Manager. The class diagram of the Request Manger and RequestInfo object can be seen in Figures 5 and 6. The Request Manager builds a RequestInfo object by gathering the student, problem, and tutor strategy context objects provided by the Context Manager. Another revision was made after we were informed that the Request Manager should be able to make multiple predictions for different tutor strategy types. To achieve this, we changed the input list of tutor strategies to be a hashmap containing the tutor strategy type as the key and the list of tutor strategies as the value. The returned tutor strategy was also changed to be a hashmap that contains a chosen strategy for each tutor strategy type.

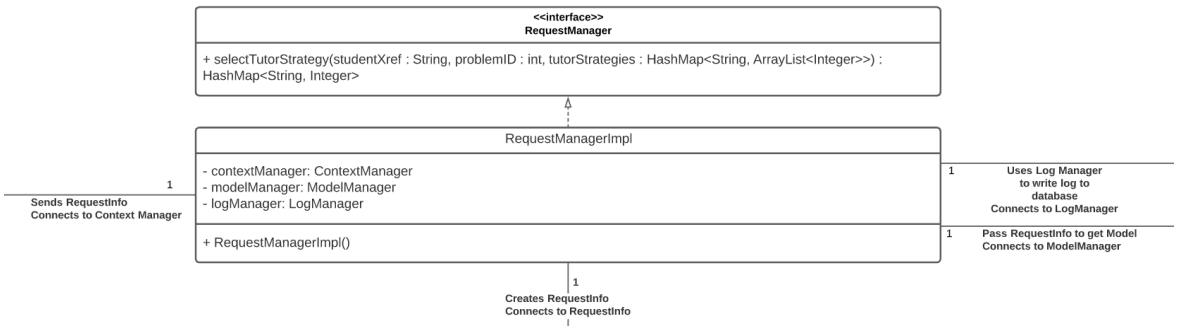


Figure 5. Class Diagram for the RequestManager

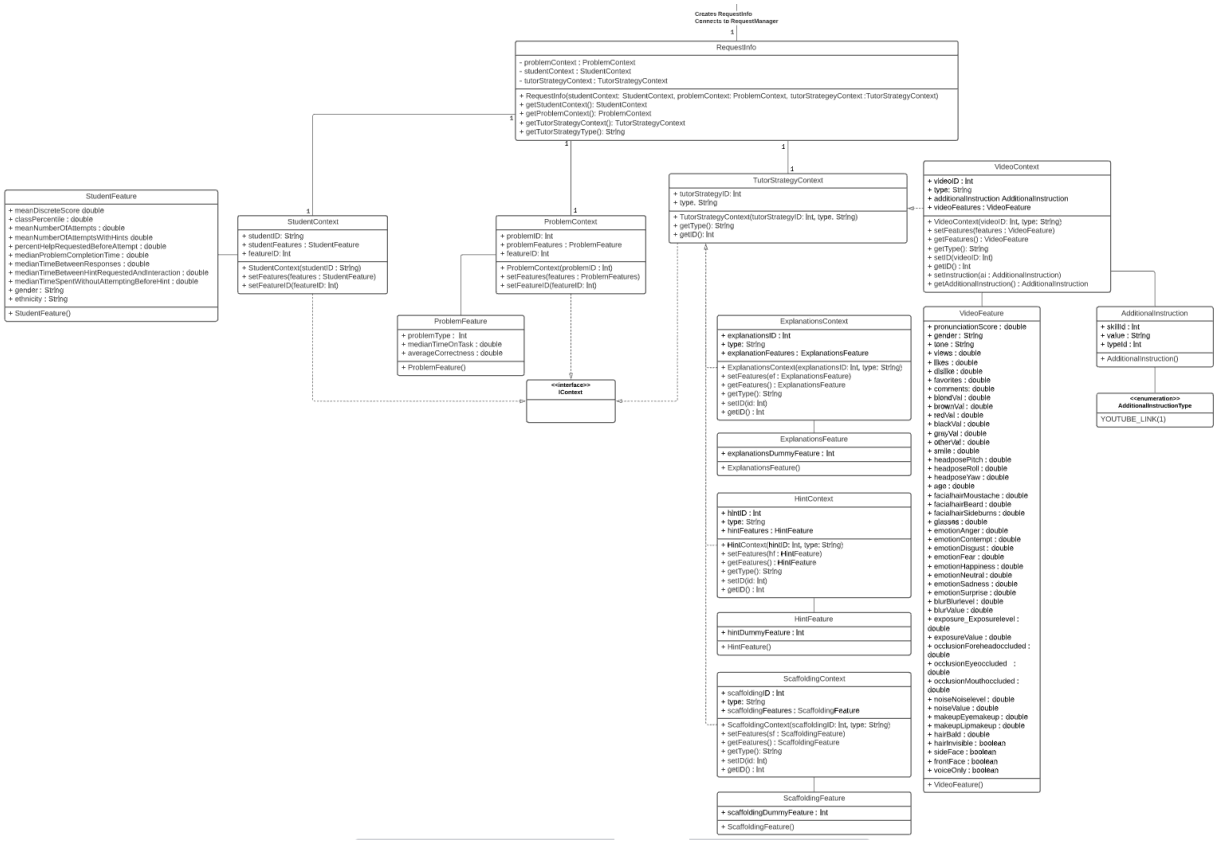


Figure 6. Class Diagrams for RequestInfo object

Once the initial design process had been approved, we moved on to the task of writing code for the Request Manager. We wrote the `selectTutorStrategy` function which is the function that is called every time by TeacherASSIST. `selectTutorStrategy` begins by creating the hashmap to be returned. Then, using the student ID and problem ID, the student context and problem context objects are made for later use. For each key in the hashmap, there is a list of tutor strategies. Each tutor strategy in the list has an associated tutor strategy context object and then a RequestInfo object. The RequestInfo object is then used to get a prediction from the model. The model prediction is then compared to the previous highest prediction, and if it's higher, the selected tutor strategy will be replaced. The process continues until there are no more tutor strategies in the list. When the tutor strategy with the highest prediction is found, that tutor strategy is put into the recommended tutor strategies hashmap with its corresponding type. In

addition, the RequestInfo relating to the tutor strategy and the model are sent to the Log Manager to write a log.

There were also major revisions that came up with design later during the project during the coding phase of the project. The decision was made that the TutorStrategyType enumeration we had been using up until that point would be changed to a string. We then refactored our code to accommodate this change. Additional instruction was originally intended to be grouped in the TutorStrategyType enumeration. Later, it was decided that additional instruction and TutorStrategyType should be separate. The Additional Instruction DAO and Additional Instruction enumeration were also created which we also had to refactor the code for. Despite this, the `selectTutorStrategy` function is still used for selecting an additional instruction.

3.1.2 Context Manager

The next part we wrote code for was the Context Manager. The initial step we took was auto-wiring the DAOs to the Context Manager. For the `getStudentContext` and `getProblemContext` functions, a query is built to find the matching student or problem reference ID in the DAO with the most recent timestamp to ensure that the most recent features are being used. Once the features have been found, they are used to build the student or problem context object which are returned to the Request Manager. For the `getTutorStrategyContext` function, we used a switch statement to check which tutor strategy type is being used. If the type matches one of the cases, it will query the corresponding DAO using the given tutor strategy ID and return the appropriate tutor strategy context object. If the type is an additional instruction, it will query the Additional Instruction DAO using the provided ID. With the resulting AdditionalInstruction object, it will check the Additional Instruction type ID, find the matching case, and return the appropriate tutor strategy context. The class diagram for the Context Manager can be seen in Figure 7 below. The only notable revision that was made when we made dummy DAOs for hints, explanations, and scaffolding. We made it so the switch statement in `getTutorStrategyContext` could communicate with the dummy DAOs and later be replaced by the DAOs for their respective tutor strategies.

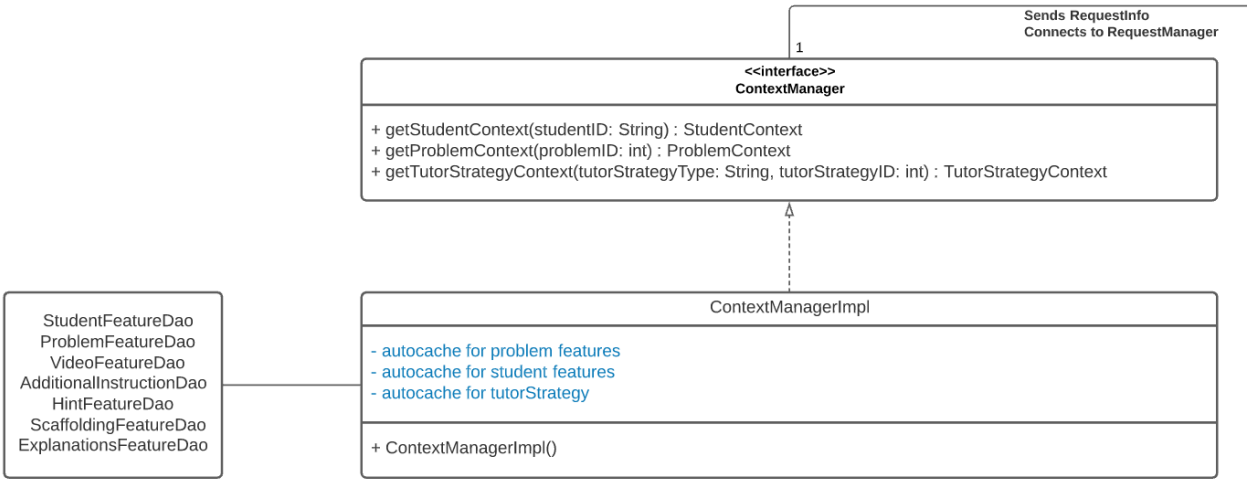


Figure 7. Class Diagram for Context Manager

3.1.3 Model Manager

Next, we wrote code for the Model Manager. The main function of this manager is `getPrediction` which returns a ModelInfo object containing the model ID and the reward prediction. We decided to add two helper functions to help with `getPrediction`. One of them was the `loadModel` function, whose purpose is to take the context from RequestInfo and decide which model to choose. The other is `formContextArray` function, which can form an array of doubles that the chosen model will understand and use to make a predicted reward. When the model is loaded and the context array is formed, the ID of the model and the predicted reward from the model are put in the ModelInfo object which is returned to the Request Manager. The class diagram for the Context Manager can be seen in Figure 8 below.

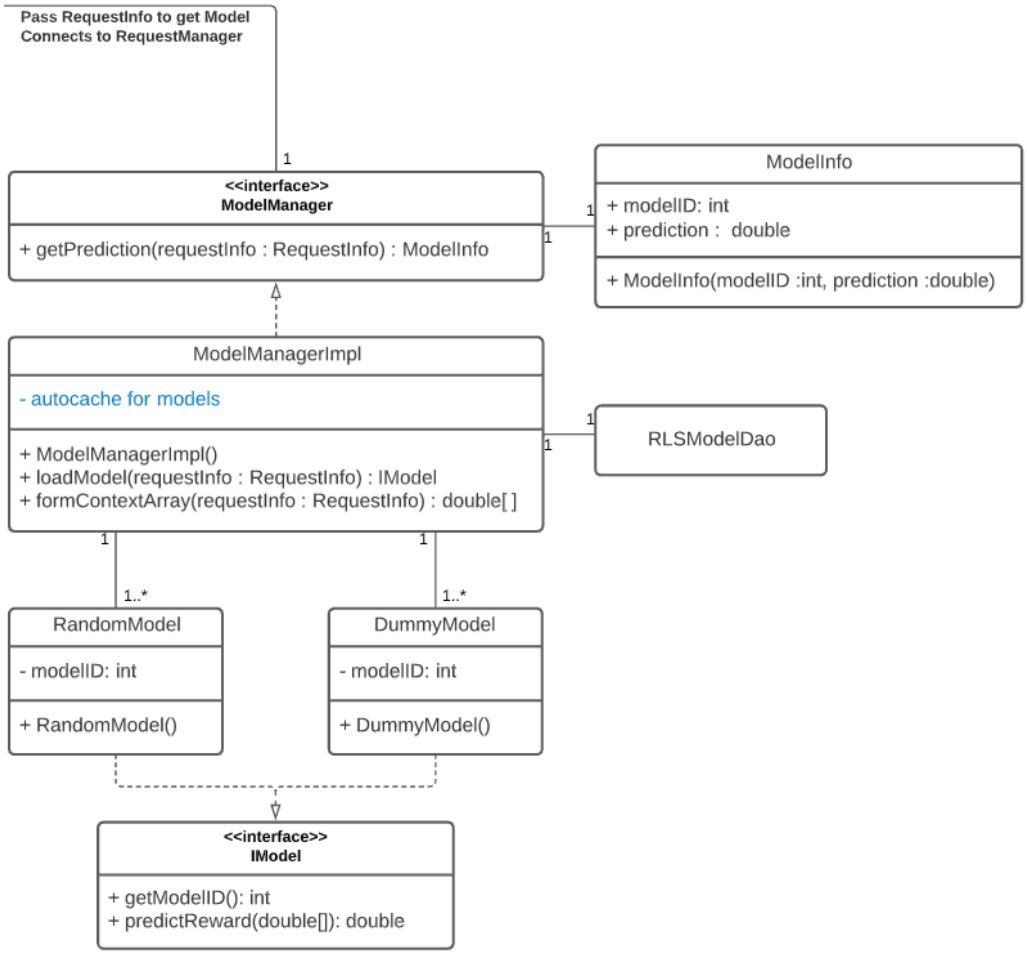


Figure 8. Class Diagram for Model Manager

The first model we made was called RandomModel and returned a randomly generated prediction score for each tutor strategy. The second model, DummyModel, was made alongside the dummy DAOs we mentioned previously. For each tutor strategy type, we made a DummyModel which returned a placeholder value in the tutor strategy context object. Originally, `loadModel` loaded the RandomModel if the tutor strategy type was additional instruction and the DummyModel if the type was a hint, explanation, or scaffolding. For `getPrediction`, if the RandomModel was being loaded, the context array would be empty. If it was the DummyModel, the context array consisted of the placeholder value. Later, we were tasked with making the ThompsonSamplingModel which would replace the DummyModel in `loadModel` and `getPrediction`. The ThompsonSamplingModel holds the mean and standard deviation of rewards

and the number of times picked for each tutor strategy in a string. To find the predicted reward for a tutor strategy, we used the formula $(R \times \sigma) + \mu$ where R is a random Gaussian number, σ is the standard deviation, and μ is the mean. After the ThompsonSamplingModel was implemented, we then constructed the BinaryThompsonModel. This model uses the same method for predicting the reward but contains different information in the model value string. The string now contains the number of times each 1 and 0 have been returned as the reward. With this information, the mean and standard deviation can be derived. The mean is found with the equation $\mu = n_1 / (n_1 + n_0)$ while standard deviation can be found with $\sigma = \sqrt{((n_0 \times n_1) / (n_1 + n_0)^3)}$ where n_0 is the number of times 0 has been returned and n_1 is the number of times 1 has returned.

3.1.4 Log Manager

After the Model Manager, we wrote the code for the Log Manager. The `writeLog` function uses the RequestInfo and ModelInfo objects of the tutor strategy with the highest predicted reward. An RLS log entry is created. The contents of the log entry can be seen in Table 1. For `nextProblemCorrectness` and `nextProblemCorrectnessTimestamp`, the required information will only be available after the student has answered the next problem, so they are not handled by the Log Manager when creating a log entry. When the available information is provided, the Log Manager sends the log entry through the RLS logs DAO. The class diagram for the Log Manager can be seen in Figure 9 below.

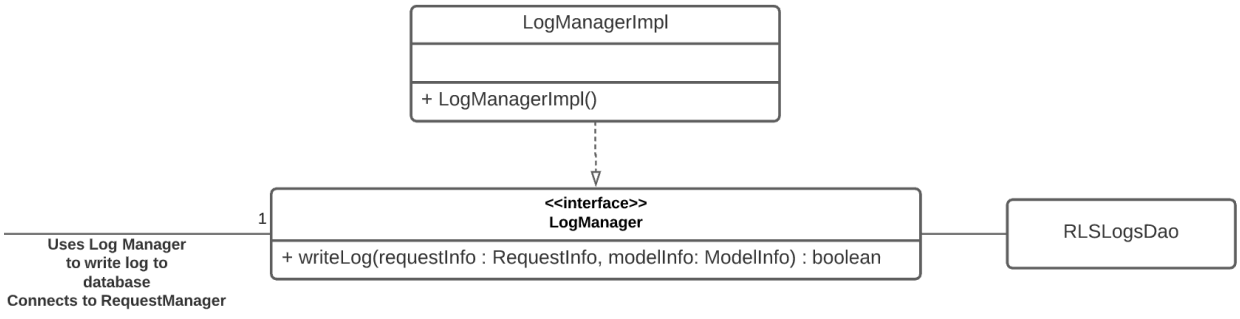


Figure 9. Class Diagram for Log Manager

Logged Item	Description
studentFeatureID	ID for the student's features in the student features DAO. Obtained from RequestInfo.
problemFeatureID	ID for the problem's features in the problem features DAO. Obtained from RequestInfo.
tutorStrategyTypeID	Type of tutor strategy being recommended. Obtained from RequestInfo.
tutorStrategyFeatureID	ID for the selected tutor strategy's features in the tutor strategy features DAO. Obtained from RequestInfo.
tutorStrategyPredictedReward	Predicted value of the selected tutor strategy. Obtained from ModelInfo.
modelId	ID for the selected model in the RLS model DAO. Obtained from ModelInfo.
recommendationTimestamp	Time of when the tutor strategy was recommended. Obtained in Log Manager.
nextProblemCorrectness	Boolean for if the student answered the next problem correctly. Obtained when the student answers the next problem.
nextProblemCorrectnessTimestamp	Time of when the student answers the next problem. Obtained when the student answers the next problem.

Table 1. RLS Log Entry

3.2 Implementing Request Manager in TeacherASSIST

Once the Request Manager implementation was completed, the next step taken was to integrate it into TeacherASSIST. As we established previously, TeacherASSIST is responsible for providing a tutoring strategy to students upon request. These tutoring strategies are related to the problem the student is working on and randomly selected. By integrating the Request Manager into TeacherASSIST, we set the stage for tutor strategies to be personally selected for each student.

The first step to integrating the Request Manager was to combine the RLS project with the TeacherASSIST project. This step was relatively straightforward as the RLS project is a Maven project type. Maven projects can convert the Java files within into .jar files and upload them through a process called a Maven build. After performing the Maven build, we can access the uploaded .jar files by editing the Project Object Model (POM) file of the TeacherASSIST project. The POM file of a Maven Project is responsible for configuring the project and by editing it, we tell the TeacherASSIST project to import the .jar files we previously uploaded. Through this process, we were able to access the implementation of the Request Manager and created instances of the manager inside of the TeacherASSIST project.

Once the RLS and TeacherASSIST projects were combined, we had to figure out where exactly we should add the Request Manager. We ended up having to meet with Thanaporn "March" Patikorn. March is a PhD student in the ASSISTments lab and the head developer behind TeacherASSIST. During this meeting, March suggested that we create a new Java class called RLSHelper. The main reasoning for creating an entirely new class was to keep the code organized and allow future editors of TeacherASSIST to have an easier time finding where RLS was integrated. Inside of the RLSHelper, we created functions that take specific parameters provided by TeacherASSIST and use the Request Manager to select the proper tutoring strategy or video for the student. Once we understood where we could add the Request Manager, we needed to know where to create an instance of the RLSHelper in TeacherASSIST.

TeacherASSIST is mainly structured around the TeacherAssistManager. In the implementation of TeacherAssistManager, there is a function called `selectTutorStrategyFromMany` where a decision is made to select a tutor strategy. This was the place where we were able to insert our instance of RLSHelper. The parameters for this function can be seen in Table 2.

Parameter	Description
tsList	a list of potential tutor strategies TeacherASSIST can provide to the student
creatorType	an enum to indicate how the strategy was selected
rlsEnabled	a boolean to help form a conditional for using RLS
studentXref	a string that uniquely identifies the student requesting a tutoring strategy
problemId	an integer that uniquely identifies the problem the student requires a tutoring strategy

Table 2. Parameters for `selectTutorStrategyFromMany` function

The most problematic part about using these parameters is the tsList since it is a list of TutorStrategy objects. This structure of the data caused a problem since the Request Manager is not aware of this object which leads to compatibility issues with TeacherASSIST. In other words, something needed to be done to handle the list of TutorStrategy objects, or we cannot use the Request Manager to select tutor strategies. Recall that the Request Manager takes in a hashmap where the key is a string and the value is an array of integers. Also, recall that the string is an indicator for the tutor strategy type and the array holds integers that help identify the tutor strategy. To sort out this compatibility issue, we created a function inside of RLSHelper that iterated through the list of TutorStrategy objects and processed it into a hashmap that is compatible with the Request Manager. This worked out since the TutorStrategy object itself contains the unique identifier for the tutor strategy and the type of tutor strategy needed to construct the hashmap for Request Manager. After the hashmap was created, we were successfully able to pass it to the Request Manager to select a tutor strategy as we intended. By selecting a strategy through Request Manager, we can add personalization and log the instances of a tutor strategy requested by a specific student.

3.3 Implementing Additional Instruction

While we had already integrated Request Manager to select tutor strategies for TeacherASSIST, we still needed to implement video recommendations into TeacherASSIST. This process was tricky since there were multiple approaches to its implementation. At first, we were going to treat videos as a tutor strategy called “additional instruction”. However, after many meetings, a decision was made to handle additional instruction differently from tutor strategies. The justification for this decision was because tutor strategies are linked to specific problems while additional instructions are linked to the skills used to solve the problem. This will also allow for more types of additional instructions to be created in the future.

To adapt to the change of separating additional instruction from tutor strategies, adjustments needed to be made for the Request Manager and RLS Helper. The first of these adjustments involved creating an AdditionalInstruction object. The characteristics of the object and they represent can be seen in Table 3.

Characteristic	**Description**
skillId	a unique ID that corresponds to each of the Common Core Standard codes. Ex. a value of “7” may correspond to the 4.G.A.1 skill
value	a string that holds the contents of the additional instruction such a video link Ex. http://www.youtube.com/watch?v=9O7D-YiSxOo
typeId	an integer that identifies the type of additional instruction Ex. a value of “1” corresponds to a YouTube Link

Table 3. Characteristics of the AdditionalInstruction object

With the AdditionalInstruction object created and implemented in both Request Manager and RLS Helper, the next step was to find a method to decide which video to recommend upon an additional instruction request. The video must be related to the skills being assessed in the problem as with any form of additional instruction. Each problem hosted on the ASSISTments platform is already tagged with the appropriate skill codes defined by the Common Core Standards for Mathematics. To ensure quality and correctness during the skill tagging process with videos, ASSISTments hired a dozen teachers to decide if the videos that were found with algorithmic YouTube searches were relevant to the skills being assessed.

Knowing that both the videos and problems were tagged with skill codes, we needed to implement a function to correlate the two. TeacherASSIST already has an identifier for the problem that the student is working on but not its skill codes. To retrieve the skill codes, we got some aid from Joseph St. Pierre, a software developer working at ASSISTments. Joseph wrote a function that will return the skill codes of a problem given the problem identifier. After implementing this function inside of RLSHelper, we were able to take the problem identifier from TeacherASSIST, obtain the skill codes related to the problem, and find potential videos to recommend. We then passed these videos to the Request Manager where a single video is recommended.

After the Request Manager has returned the recommended video, it is now TeacherASSIST's responsibility to deliver that video to the student who initially requested additional instruction. To achieve the implementation shown in the Ideas Lab mock-up in Figure 4, we needed aid from the engineering team of ASSISTments. The engineering team is currently responsible for modifying the manifest that communicates to the ASSISTments tutor and updating the user interface for displays. These steps are necessary to fully implement the additional instruction feature and allow students to watch the videos we recommended.

3.4 Designing a Study for Testing RLS Personalization

Once the implementation for the RLS service is up and running on ASSISTments, a study would be necessary to test that this personalized service is working properly. When it comes to recommending videos, there are many video features to consider for a particular student. Some students at Penn State University managed to compile a list of video features (see Table 4) for the RLS service to identify and personally tailor its recommendations.

Video Feature	**Description**
Pronunciation score	how clear the speaker articles and pronounces words
Gender	the gender of the speaker
Tone	the tone of the speaker used throughout the video
Voice Only	whether or not the speaker's face is shown in the video
Views	the number of views on the YouTube video
Likes	the number of likes on the YouTube video
Dislikes	the number of dislikes on the YouTube video

Table 4. Example Video Features that RLS takes into consideration

Now that we established what features the RLS services will use to make their recommendations, it is appropriate to start talking about how to design a study around it. For any study, it is important to remove bias from the data and that is often done through randomly selecting participants for the study. Conveniently, ASSISTments can easily create randomized control groups for a study such as this one. Students on the ASSISTments platform will be randomly selected and placed into either one of two groups. The first group will be referred to as the control group. Inside the control group, students will have access to a service that will recommend videos randomly upon request. The video will still relate to the skills assessed in the problem in which the service was requested but, there is no personalization involved. The second

group will be referred to as the experimental group. The experimental group will have access to the RLS service where they receive personalized additional instruction upon request. The videos that are recommended by the RLS service are not randomized and will take advantage of a personalized learning algorithm to find the best video to suit the student. A summary of the two groups and what they are exposed to can be seen in Table 5.

Group Name	**Exposure**
Control	A randomized video recommendation service
Experimental	A personalized video recommendation service (RLS)

Table 5. Groups for testing RLS personalization

Now that we established the groups for the study and what they are being exposed to, we need to discuss what type of data we are going to collect and measure. We want to measure student knowledge gain. To accomplish this, we want to observe instances where students of the two groups request additional instruction and get a video recommendation. A student will most likely seek additional instruction after getting a problem wrong. If a student can answer the problem correctly after watching the video, we assume they have learned and gained the knowledge necessary to solve the problem.

We are doing this study to measure how effective personalized learning is. After collecting all this data, we would analyze it and compare the results of the experimental group to the control group. More specifically, we would be looking at the difference in the average knowledge gained for a student from using personalized recommendations and randomized recommendations. If the average knowledge gained in the experimental group is significantly larger than the control group, that would indicate that personalized recommendations are significantly more effective than randomized recommendations. If the difference in the average knowledge gained is about the same, that would indicate that both recommendation methods

were equally effective in aiding students. Overall, the study will indicate the effectiveness of the personalized additional instruction provided by the RLS service.

4. Future Work

4.1 Additional Steps and Ideas

While we did mention needing the assistance of the software engineering team of ASSISTments to implement personalized videos for students, there are still some other features that still need to be done for this project. From a coding perspective, we would still need to establish a link between TeacherASSIST and the manifest files once they are finished by the engineering team. Afterward, the code that we have written will be used and can be ready to be implemented on the ASSISTments platform.

Some other aspects that can be worked on are related to the Model Manager in the Request Manager. To be more specific, the need to implement models that use the additional features in the reward calculation. Recall how we were able to retrieve features regarding the student and the problem inside of the Context Manager. These features have not been considered in any of the models we have already implemented in the Model Manager. These models are more complex and time-consuming to develop but allow tutor strategies to be more tailored for an individual's experience. Some other features that can also be considered such as the teachers of the student. For example, if a teacher often provides visual explanations to their students, we can consider that feature and favor explanations that use visuals for those students in that teacher's class.

It is also important to consider the algorithms that will be deciding which model to use to predict the rewards. In the future, we could use a supervised learning algorithm such as neural nets, decision trees, or logistic regression. In supervised learning, the algorithm analyzes a set of certain data and makes decisions based on the analysis. We would start by providing videos to students randomly. Then when we had a significant amount of data to analyze, we would give it

to the algorithm and it would observe the context and give the student the video with the highest chance of the student getting the next problem correct. We could also implement unsupervised learning algorithms. In unsupervised learning, clusters are formed based on certain attributes and these clusters can be analyzed and an algorithm can be derived. In our case, the clusters can be formed using the provided features from students or problems. With the way the RLS was designed, many possible algorithmic models could be implemented and experimented with.

Once a machine-learning algorithm has been selected and successfully implemented, it would be time to put everything together to get the RLS service up and running. Before one can release a new online service, testing is necessary to ensure a smooth experience for its users. One of the most common ways to test a new service would be through a closed beta. A beta for the RLS service would be beneficial to perform bug fixes and get valuable feedback from users. ASSISTments has done closed betas before such as with ASSISTments 2.0 so a similar treatment for the RLS service would be reasonable. After the service has been thoroughly tested and implemented into the ASSISTments platform, it would be appropriate to perform the RLS personalization study described in Section 3.5.

One last idea that can be done in the continuation of this project is adding more types of additional instruction. While this project only planned to provide videos hosted on YouTube, there is an opportunity to add to more forms of additional instruction given how the code was structured. Some ideas for new types of additional instruction could be links to other learning platforms such as Khan Academy due to its structured format and many resources to help students understand the skills required to solve the problems they were originally assigned. Another form of additional instruction that has potential is links to websites with articles explaining the concepts such as Wikipedia or MathIsFun. These websites often have a variety of example problems and have graphics that can be helpful for students to learn concepts with a different approach.

4.2 Takeaways from the Project

Working on this project has been a commitment in both terms of time and effort. However, the experience we gained was invaluable and something we will carry on into future projects and work environments. One of the immediate benefits of this project was learning proper communication skills and understanding how an organization like ASSISTments operates. There are many distinct teams in ASSISTments working on various projects simultaneously. There were many times we had to reach out to people outside of our team to get feedback and assistance. This includes talking to software engineers and head developers who attended meetings with us and provided constructive criticism towards our coding style and structure. One of the biggest challenges was learning how to write code in a way that makes logical sense to anyone who sees it for the first time. This was an important skill to learn as future workers of this project can easily understand what we wrote and how we approached it. By learning how to redistribute online content on a learning platform, we also learned how to write clean and effective code which will certainly be helpful in our careers.

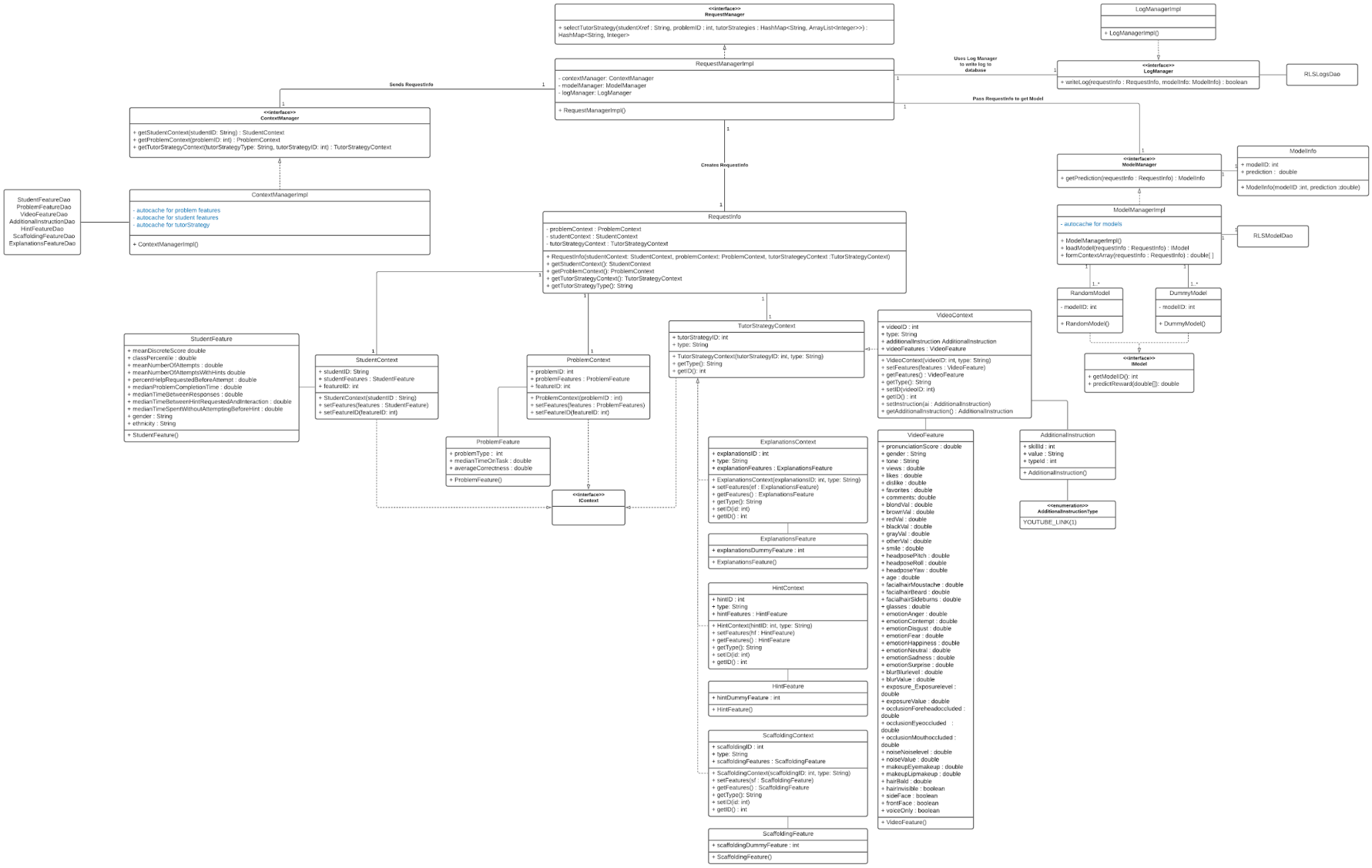
5. Conclusion

The code we have written serves a pivotal role in the integration of the RLS. The Request Manager can take a list of tutor strategies and pick one that is predicted to help the student the most. It does this with the aid of its sub-managers. The Context Manager is able to pull the features of the relevant student, problem, and tutor strategy from the database, so they can be used when predicting a reward. The Model Manager is responsible for choosing the appropriate model to be used, providing it with the context, and returning the predicted reward to the Request Manager. The Log Manager can add a log entry to the RLS log database so that the model chosen can be refined later based on the results. Another crucial step that we performed was integrating the Request Manager with TeacherASSIST. The initial step was to build the .jar file of the RLS and edit the POM file in TeacherASSIST to access it. Next, we made the RLSHelper class in TeacherASSIST, which contained functions to handle communicating with the Request

Manager. For the function that chooses a tutor strategy in TeacherASSIST, `selectTutorStrategyFromMany`, we needed to convert the list of tutor strategies that was provided into a hashmap in order to use `selectTutorStrategy` in the RequestManager. After that step was finished, the groundwork was set for TeacherASSIST to operate with the RLS.

By implementing the Request Manager, the RLS can receive requests from TeacherASSIST and return with a personally selected tutor strategy. In addition, different machine learning models can be provided to choose videos for students using given context. Beginning soon, students using the ASSISTments platform will be able to select additional instruction, then be aided with a video that will be chosen based on what will help them the most. Through this process, we were able to learn key skills in developing a software application as a team as well as apply our skillset to assist in researching teaching strategies that will help students. This project has given us more software engineering experience, which will help us later in our careers. We have also learned how our work can benefit others like students and teachers as well as fellow programmers. We hope that the implementation of the Request Manager will lead students to have a personalized and quality experience in their pursuits of knowledge on the ASSISTments platform.

Appendix: UML Class Diagram



References

Bulger, Monica. "Personalized Learning: The Conversations We're Not Having." Data & Society, 2016.

Grant, P. and D. Basye. *Personalized Learning: A Guide for Engaging Students with Technology.* International Society for Tech in Ed., 2014.

Heffernan, Neil T. and Cristina Lindquist Heffernan. "The Assistments Ecosystem: Building a Platform That Brings Scientists and Teachers Together for Minimally Invasive Research on Human Learning and Teaching." *International Journal of Artificial Intelligence in Education*, vol. 24, no. 4, 2014, pp. 470-497, doi:10.1007/s40593-014-0024-x.

Pashler, Harold et al. "Learning Styles: Concepts and Evidence." *Psychological Science in the Public Interest*, vol. 9, no. 3, 2008, pp. 105-119, doi:10.1111/j.1539-6053.2009.01038.x.

Patikorn, Thanaporn and Neil T. Heffernan. "Effectiveness of Crowd-Sourcing on-Demand Assistance from Teachers in Online Learning Platforms." *Proceedings of the Seventh ACM Conference on Learning @ Scale*, Association for Computing Machinery, 2020, pp. 115–124. doi:10.1145/3386527.3405912.

Razzaq, Leena and Neil Heffernan. *To Tutor or Not to Tutor: That Is the Question.* vol. 200, 2009.