Creating Model Retraining and Logging Infrastructure to Grow QUICK-Comments – A Tool To Suggest Scores and Feedback for Open-Ended Questions in ASSISTments

An Interactive Qualifying Project
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
degree of Bachelor of Science

by
Trevor Paley
with Kendall Goto

Date:
3 May 2022

Report Submitted to:

Professor Neil Heffernan
Worcester Polytechnic Institute

ABSTRACT

QUICK-Comments uses machine learning models to give teachers automatic suggestions of scores and feedback for student responses to open-ended questions on ASSISTments, an online education platform. However, up until now, the data being passed through the QUICK-Comments API has not been logged, limiting the ability to research it. Additionally, the machine learning models being used have needed to be retrained manually when new data is available. In this paper, we design and implement a solution to both of these problems to prepare for the launch of an extended research platform built on QUICK-Comments.

TABLE OF CONTENTS

Introduction and Background

Many existing services use natural language processing (NLP) to automate sentiment analysis, text summaries, text filtration, and more (Nadkarni et al., 2011). These NLP-based services rely on the continual development of underlying machine learning models to offer feedback. These models are ideally trained as a backing dataset increases in size, though often the process of regenerating the models using new data needs to be initiated manually. Further, if the used models are expected to improve, their outputs must be documented and archived to allow for efficacy testing and general instrumentation. Without these surrounding systems in place, it is difficult for NLP services to grow as they are used, limiting their ability to better serve users.

This paper will discuss new systems designed to complete the automatic model retraining and runtime documentation of one particular service using NLP. This work includes a design proposal and implementation of a retraining system that can automatically generate new instances of NLP models based on a growing data set. It also includes an implementation of a logging system that can directly document how the associated NLP models are used for analytical analysis and growth of the models.

The implementations in this paper take the form of two distinct pieces. The retraining system is a discrete program that can be directly accessed by privileged system users or automatically scheduled to retrain models regularly. The logging architecture is created as an augmentation that operates on an existing REST API which is utilized as an interface to the associated NLP models. This logging architecture acts as middleware between the end-user and the REST API system, to document the input and output behavior of the NLP service.

These systems were produced to further the development of QUICK-Comments, a web-based system that provides grades and comments for student responses to open-ended questions on ASSISTments, an online educational tool.

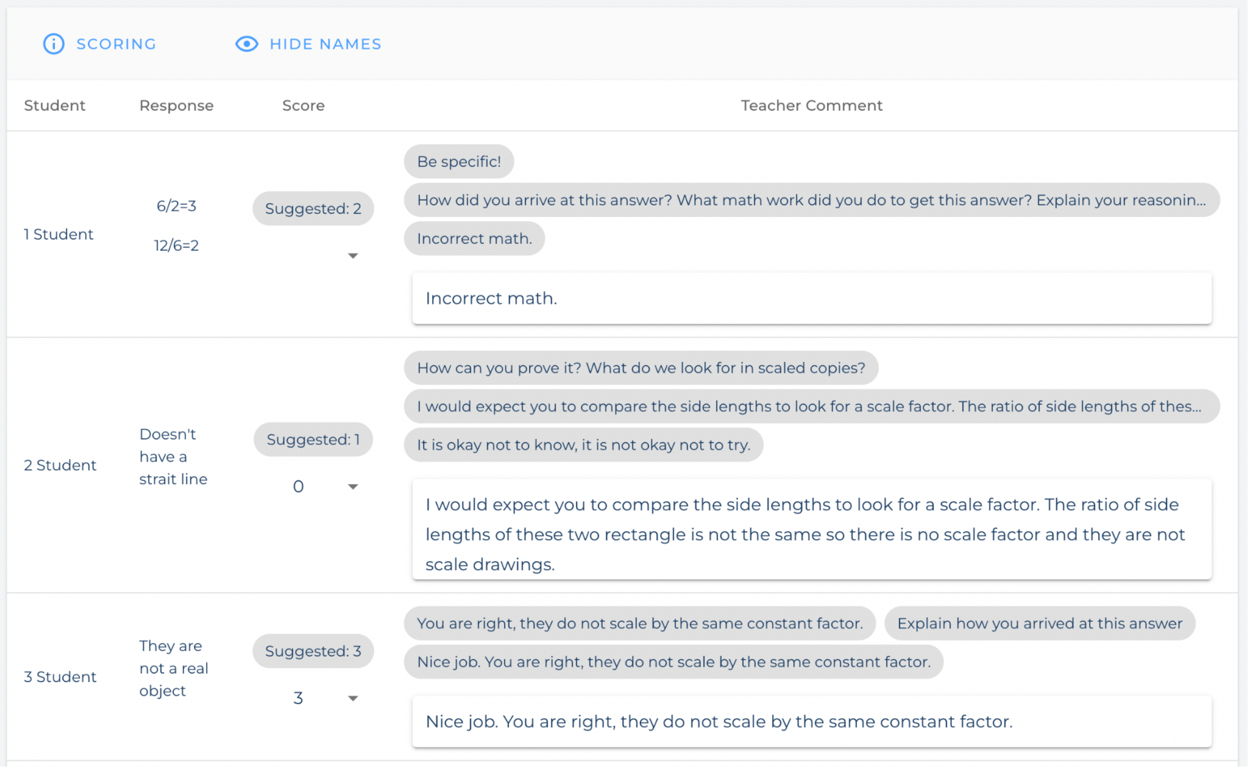
ASSISTments

ASSISTments is a free online education resource aimed to assist schools and teachers' math curricula. A core part of ASSISTment's mission is to continually perform research and innovate to deploy cutting-edge technology onto the platform. As such, The ASSISTments Foundation and Worcester Polytechnic Institute support numerous experimental projects that enhance, test, or grow ASSISTments.

ASSISTments is used by 20,000 teachers and 500,000 students to improve math education by providing digital problem sets for students to answer with continuous feedback (ASSISTments Foundation, 2019). The data from these users is used by researchers to further education, by providing educators with insights and guiding effective curriculum development. Of the many ongoing ASSISTments projects, QUICK-Comments is an actively deployed project designed to assist the teacher-student feedback loop.

QUICK-Comments

Figure 1: The ASSISTments Interface for QUICK-Comments



Student	Response	Score	Teacher Comment
1 Student	6/2=3 12/6=2	Suggested: 2	Be specific! How did you arrive at this answer? What math work did you do to get this answer? Explain your reasonin... Incorrect math. Incorrect math.
2 Student	Doesn't have a strait line	Suggested: 1 0	How can you prove it? What do we look for in scaled copies? I would expect you to compare the side lengths to look for a scale factor. The ratio of side lengths of thes... It is okay not to know, it is not okay not to try. I would expect you to compare the side lengths to look for a scale factor. The ratio of side lengths of these two rectangle is not the same so there is no scale factor and they are not scale drawings.
3 Student	They are not a real object	Suggested: 3 3	You are right, they do not scale by the same constant factor. Explain how you arrived at this answer Nice job. You are right, they do not scale by the same constant factor. Nice job. You are right, they do not scale by the same constant factor.

Problem sets often ask students to explain how they solved a particular problem, but giving feedback to every student for each one of these short response problems can be extremely time-consuming. This issue is often great enough that the teacher does not have time to provide detailed, or in some cases any, feedback to students. To alleviate this issue, ASSISTments has a feature called QUICK-Comments which uses machine learning techniques to automatically generate grades and feedback (comments) for short answer responses, which teachers can take as is, modify, or ignore (Benachamardi, 2021).

QUICK-Comments operates by running student responses through machine learning models which are often tuned to particular problems. However, in the past, these models had to be manually generated and copied into the production environment. As part of an overall shift of ASSISTments to use Amazon Web Services, there was work to move these QUICK-Comments models to the cloud as well, but models would still have to be trained and uploaded to an S3 bucket manually. Automating that retraining and uploading process was one of the goals of this project.

Like ASSISTments as a whole, a major goal of QUICK-Comments is to be able to run randomized controlled trials on how users interact with the software. However, before our project, there was no mechanism for automatically logging requests and responses to the QUICK-Comments

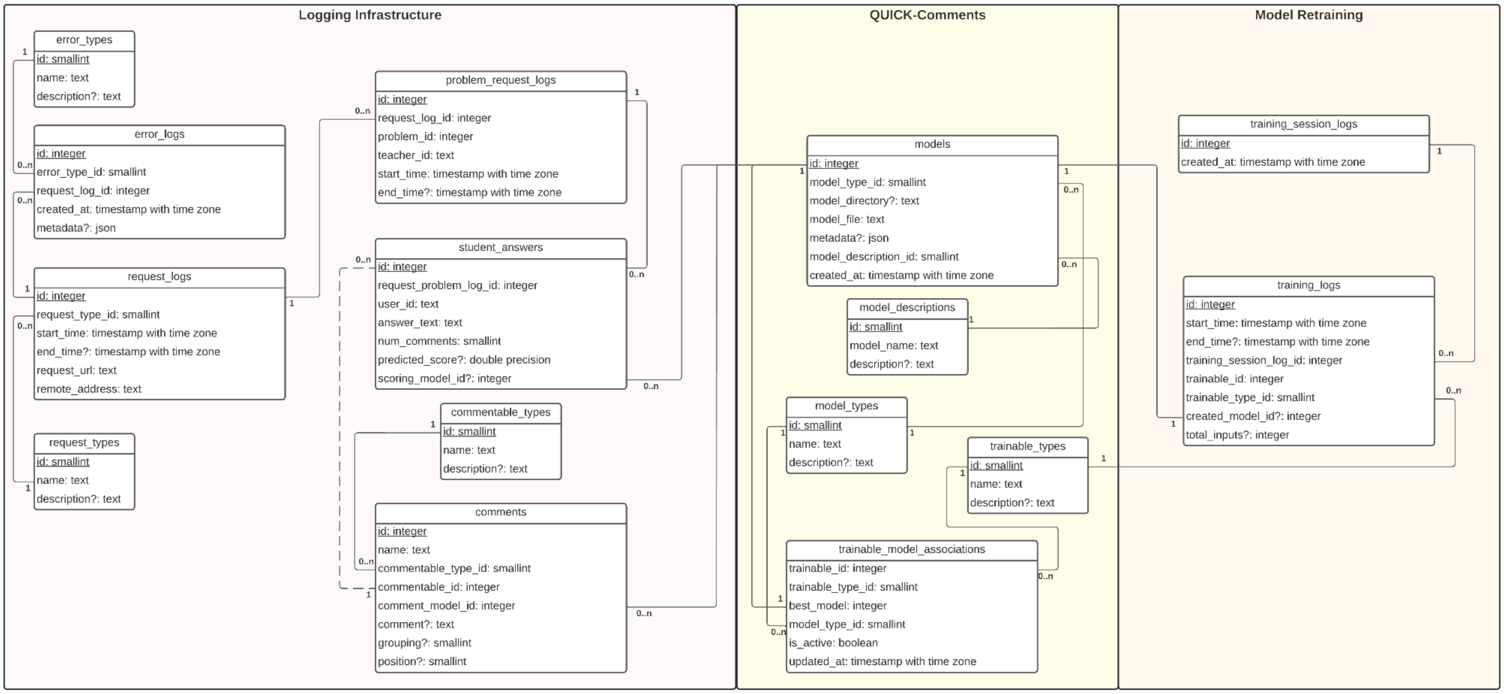
API, which was necessary before RCTs could be implemented. Fortunately, a prior group of students completed some of the design work in an unpublished report, which included an initial design for the QUICK-Comments logging infrastructure. We tweaked that design and used it as a base for our work on implementation.

Implementation

Our work takes three long strides to improve the overall QUICK-Comments system and prepare it for continued growth and development. First, a database design was developed and implemented to better solidify the QUICK-Comments process and provide storage for logged data. Next, automatic model retraining was developed as a step to further improve model accuracy with continuous growth as new data becomes available. Finally, a high-detail logging infrastructure was designed to provide researchers on QUICK-Comments with more information to develop innovative ways to produce generated feedback.

Database Design

Figure 2: QUICK-Comments Entity-Relationship Diagram



The database design we have implemented builds upon prior work on a database design for QUICK-Comments Randomized Controlled Trials (RCT). While we did not implement nor review in depth the RCT infrastructure in this project, we did implement the core QUICK-Comments database infrastructure underlying it, as well as logging and retraining (Figure 2). The latter two components will be discussed in their own sections, but we want to discuss the core QUICK-Comments part first, as it is important for understanding how everything else fits together.

All models, such as the grading and comment models for each problem, have a row in the `models` table. This includes models which are no longer used, as we may want to compare old models with new ones, either by directly inspecting them or through RCT in the future. This also creates accountability if there is an issue with generated models, as we can track precisely what caused the issue and when it first appeared. Each model has associated with it a model type and a model description. The model type defines what the model is used for (e.g. is it a comment model or a scoring model) while the model description defines what algorithm was used to train the model. These values are designed to be extensible so that QUICK-Comments can include currently unrealized model types and algorithms in the future while maintaining the same structure.

To select which model to use, we have another table for model associations, which keeps track of the "best" (currently defined as the most recent) model of a given model type for a particular "trainable," and whether that association is active (i.e. should we use it when looking for a model to run). A trainable is a combination of a trainable type, a type of thing that can be trained (e.g. a problem), and a trainable id, the particular trainable of that type to train (e.g. a problem ID). This allows us to train new types of objects in the future. A downside of this structure is that it is impossible to have a foreign key to the table containing a particular trainable, but doing that could have been tricky anyways due to the QUICK-Comments infrastructure's isolation from the rest of ASSISTments.

Automatic Retraining

Until now, training models for QUICK-Comments has been a manual process. Now, with our migration to Amazon Web Services, we wanted the ability to regularly run the training process to update our models as new data came in without the need for human intervention. In this section, we will discuss how the automatic training process works, as well as the recovery flow we have implemented in case there is a failure during the training process.

Training Flow

Before training, we first fetch all relevant data from our database and save it to disk, including the problem IDs which we want to train on and the set of responses for each problem with their associated instructor-assigned comment and grade. We then pass this data into our trainer module, which has been abstracted for all training methods (utilizing the `model_description` table as well as an internal `Model` interface). At the moment, however, we only build models using SBERT-Canberra (Baral et al., 2021). With minor tweaking, the trainer module would also be able to support training for areas other model types than just response grades and comments.

For each problem that we need to generate grading and comment models for, the trainer module loads the appropriate data and then creates a model object using the appropriate training method. This model object is then saved to disk as a pickle file, and we create an entry for the model in the database. The model is then uploaded to an S3 bucket on AWS, though the particular upload function we use is available as a parameter to our trainer, so if more fine-grained control is desired, it can be exercised without changing the internals of the trainer module. Lastly, the database is updated to mark the new model as the best model for the particular trainable via the model associations table we previously mentioned.

Error Recovery

Training can take a long time to complete, and there is a possibility of an error occurring during training. To mitigate the impact of this, we built a recovery flow in which training progress can be saved, and the trainer only needs to train models which were not previously trained. Possibly counter-intuitively, our recovery flow works by *not* doing something which occurs during the normal training flow. Normally, when training starts, all models and cached data from the database are cleared from the local machine, providing a blank slate for retraining. In the recovery flow, we do not clear any of this data, so when we detect the presence of a model from a previous training cycle, we can skip training that model again since it was already trained.

Currently running the recovery flow requires manual intervention to run, though in the future there could be a mechanism to automatically schedule training to resume later if an error is detected.

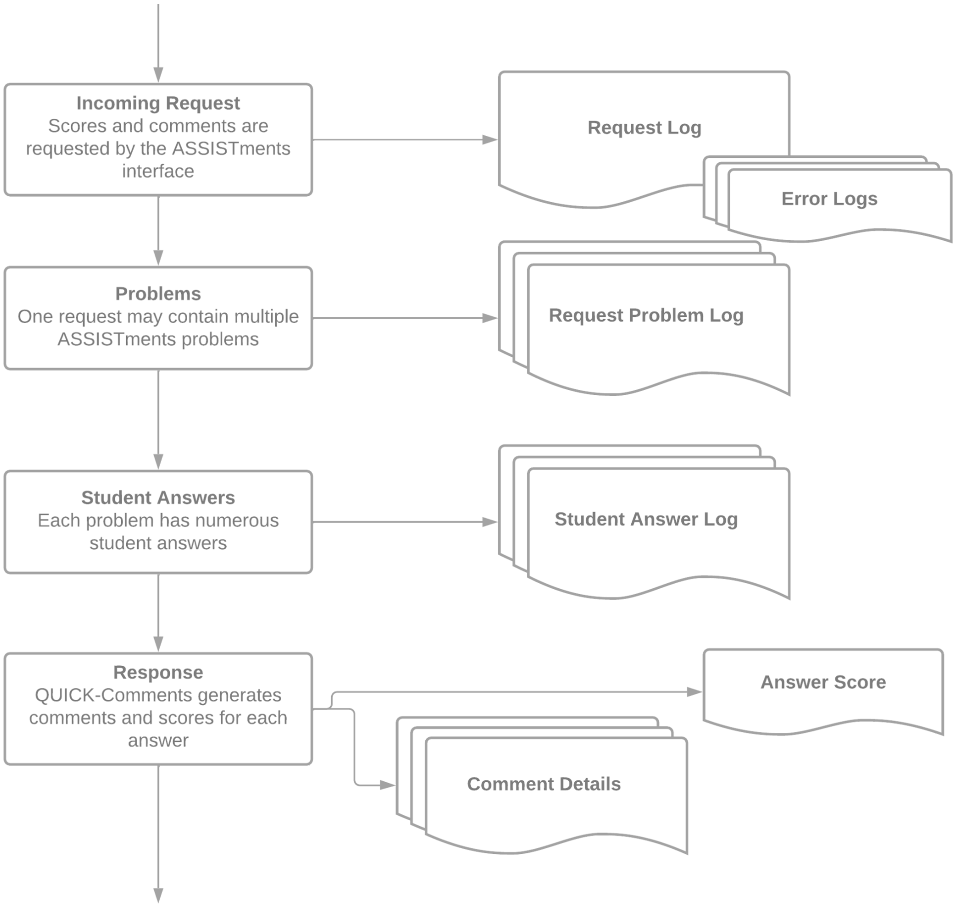
Logging Infrastructure

Before this work, the results produced by QUICK-Comments were largely undocumented, providing no context to researchers looking to improve the efficacy of the platform. When the ASSISTments platform sends a request for comments and scores from QUICK-Comments, QUICK-comments provides an estimated score and numerous comments for the end-user to consider. However, a proper logging infrastructure was required to document the generated results and provide researchers with evidence of their model's efficacy.

Logging Requests

Utilizing the tables included within the "Logging Infrastructure" section of the Database Design (Figure 2), we are now able to log QUICK-Comments action on a request-by-request basis.

Figure 3: Request Processing Flow


Incoming Request
Scores and comments are requested by the ASSISTments interface
Request Log
Error Logs
Problems
One request may contain multiple ASSISTments problems
Request Problem Log
Student Answers
Each problem has numerous student answers
Student Answer Log
Response
QUICK-Comments generates comments and scores for each answer
Answer Score
Comment Details

In Figure 3, we can see the augmented QUICK-Comments flow with logging attached. For each QUICK-Comment action on the left, a corresponding log is recorded in the database on the right, fully documenting each request, its problems, those problems' answers, and each answer's feedback. If QUICK-Comments encounters an error at any point when handling a request, error logs may be attached to the request log as well, to fully document it. This enables developers to backtrack through the logs and see the specific issue to help them debug and fix it.

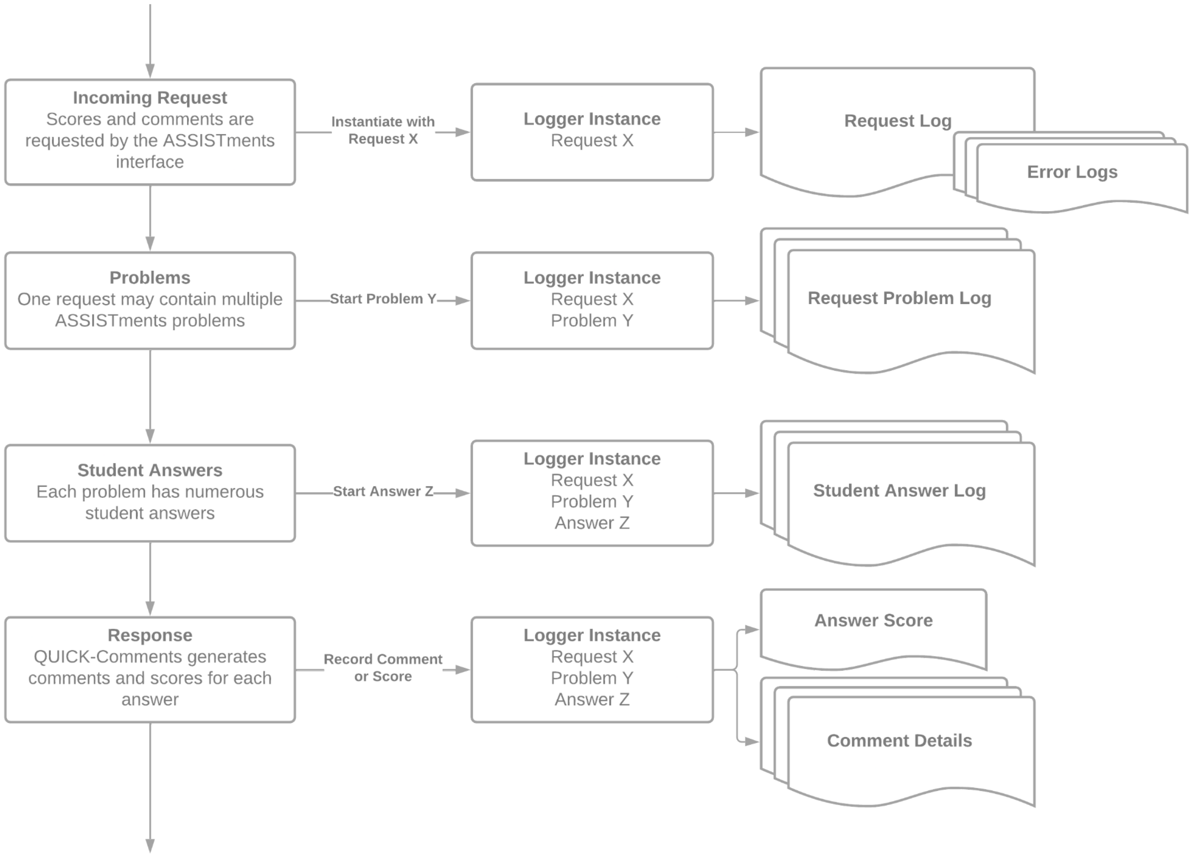
The logged entities are reflected logically in the database design seen in Figure 3. Every request maps to a `request_logs` entry, each problem maps to a `problem_request_logs` entry, each answer to a `student_answers` entry, and each comment to a `comments` entry. Generated errors map to an `error_logs` entry, if necessary. Additional "types" tables exist as well, to classify main table entries into typed groups. For instance, an `error_logs` entry may have an associated type of "Runtime Error" or "Connection Timeout Error" defined by a static `error_types` entry. At the moment these tables are largely unused but are included to further improve logging in the future with discrete type definitions.

Stateful Logger

To accomplish the logging goals of QUICK-Comments, a stateful logger was proposed. This logger is instantiated for each request, and maintains an internal state, referring to the active problem and answer that the QUICK-Comments codebase is currently processing. With the logging system maintaining its own state, the QUICK-Comments system is able to fulfill its duties with very little code overhead to perform logging actions. This design paradigm is invaluable for future development of the QUICK-Comments codebase, to ensure further researchers can grow QUICK-Comments without worrying about maintaining log-related code.

In practice, the QUICK-Comments flow is modified to instantiate a logger at the start of a request and pass key state transitions and data back to the logger (such as beginning a new problem, or the generated comments for the current problem). The logger then, in a self-contained manner, handles creating the logs and inserting them into the database, with appropriate connections between log lines.

Figure 4: Stateful Logging System



In Figure 4, the QUICK-Comments system is shown interacting with the stateful logger instance. In this model, the logger instance handles the logging, while the main suggestion engine uses a simplified interface to interact with the logger. Internally, the logger instance tracks database-related values, like a Request ID, and links stored data accordingly. This provides the core QUICK-Comments code with a very simple interface to continually document its actions, without the weight of tracking database-level events and tasks.

Training Logs

The model retraining process is also logged, though using a separate system from the request logging infrastructure, since model retraining is segregated from the public-facing API.

Each time the trainer begins training, a “training session” is created to associate all of the information about what occurred during that execution of the trainer. Even when training is resumed with the recovery flow, a new training session is created. Associated with each session are training logs, one for each model that is trained, detailing information about what data was used to train the model, the ID of the resulting model that was trained, and how long it took to train the model. These logs are linked to trainables, as we discussed in the section on database design.

Results

Thoroughly testing the effectiveness of our additions to QUICK-Comments is vital to the continued success and growth of the platform. The work produced in this paper does not constitute front-end changes towards the QUICK-Comments platform, and only includes features accessible by the developers of QUICK-Comments. This means that user surveys or audience usability tests could not be performed. As such, other means of demonstrating the effectiveness of our tools were established.

Automatic Retraining

Automatic retraining has been manually tested, both from a clean slate and from the recovery flow. One of the main issues we encountered with it was inconsistent and slow database access during training, which led us to cache all of the data we needed for retraining on the local disk ahead of time rather than fetching per-problem data on demand. However, when database access was lost during training and caused training to fail, we successfully ran the recovery flow without data loss. Losing database access after a model was generated but before it was fully processed could be an issue, though due to the order of operations, the worst that should be able to happen is that for one training cycle a single model does not get retrained. This is a relatively minor cost, as the old model should still be valid, its performance may just be slightly worse.

Performance of the training process is a concern, though no more of a concern than it was previously. With CUDA hardware acceleration on a medium-performance consumer graphics card, training all of the problems with SBERT-Canberra can be done on the scale of around a day, which is acceptable, even if not ideal. Without CUDA however, training can take several days. These issues will only be exacerbated if future training algorithms require more computational power, so avenues for speeding up this process should be pursued.

Logging Infrastructure

As a core function of the logging infrastructure, the basics of performing logging tasks were tested. This includes appropriately recording data for each request or retraining session, as well as ensuring the data integrity, validity, and value of the logs recorded. Additionally, in a production system, the logging infrastructure needed to be tested at scale, to ensure that neither large quantities of data nor a large number of requests hindered performance.

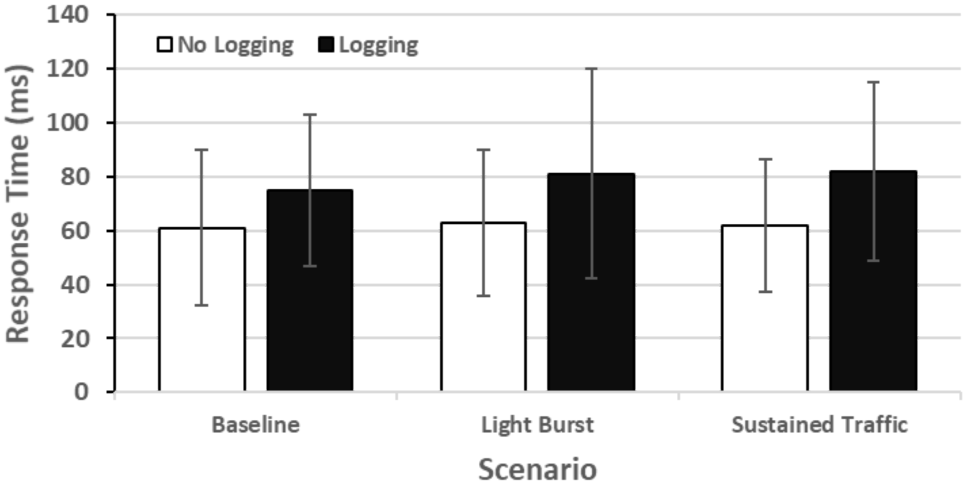
Basic Testing

General data validity of the logging infrastructure was predominately tested by hand. The database schema, including the tables and relations used, was thoroughly reviewed in regards to current functionality and future extensibility. After implementation, data was populated by the logging system into the database and monitored for general accuracy and functional value.

Error conditions, such as partial or invalid requests, were also tested to ensure they did not hamper the functionality of the system as a whole.

Further, response times were directly compared between QUICK-Comments before and after logging was enabled. It is expected that the added logging infrastructure would have some overhead given the high detail recorded by each request. However, it is important to document the added infrastructure's exact overhead to better understand the performance of the system at a production level.

Figure 5: Comparing Logging vs No Logging Performance for Typical Traffic



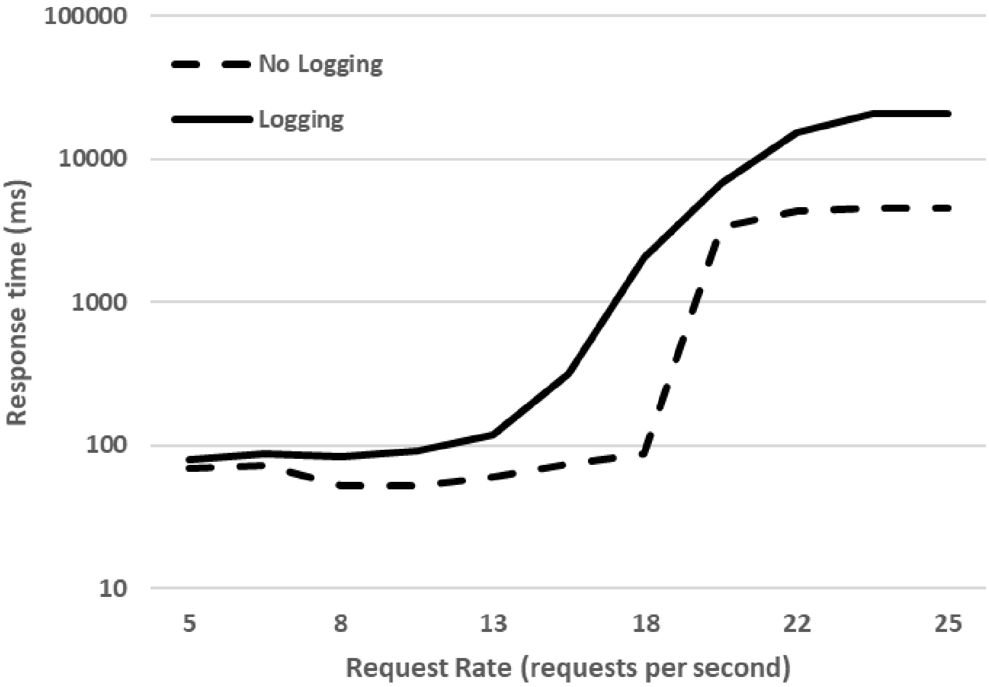
In Figure 5, the measured runtime performance is seen before and after the logging infrastructure was implemented. Initially, a baseline measurement is taken, to measure the basic performance of one-off requests. Then, two realistic scenarios are tested. In the "Light Burst" scenario, QUICK-Comments must be able to appropriately handle sudden bursts of traffic without generating errors or dropping incoming connections, at a rate of 3.5 requests per second. In the "Sustained Traffic" scenario, QUICK-Comments must be able to handle continuous but slow traffic without getting behind and delivering slow responses, at a rate of 2.5 requests per second. We see that the baseline performs closely to the two scenarios, indicating that the service is performing as expected across the tests.

In these scenarios, we see an overall change in latency by 19ms on average or a 26.5% increase. Knowing this, it can be deduced that the logging infrastructure accounts for 23% of the overall latency involved in a QUICK-Comments request. We have determined that this change in performance is acceptable in exchange for the value and detail of the data provided for further research and development of QUICK-Comments. However, further commentary on the ability to improve this value and make the logging task more lean is discussed in the conclusion.

Pre-Production Testing

To further explore the logging infrastructure's capability in a production-grade system, high-volume tests were performed to establish the limitations of the service. Benchmarking is hard and there are many considerations when stress testing, but we focused on optimizing the overall throughput threshold, measured in requests per second. As the rate of requests increases, it can be expected that the server will eventually meet a breaking point where it cannot serve requests faster than they are received, and thus they begin to backlog. In the case of QUICK-Comments, this can lead to errors and slow response speeds noticed by the end-user.

Figure 6: Comparing Logging vs No Logging Performance for Heavy Traffic



In Figure 6, the breaking point is directly measured with numerous configurations on the before and after versions of QUICK-Comments. As the number of requests increases, response time begins to rise, indicating that the server is becoming overloaded. The data indicates that the pre-logging service can handle a sustained 18.3 requests per second, while the post-logging service is only able to handle 13.2 requests per second, a reduction of 27.9%. This is a significant change in throughput, which should be better examined when scaling the QUICK-Comments into production. Currently, this threshold is an acceptable value based on the traffic currently realized by QUICK-Comments in production.

Conclusion and Recommendations

As ASSISTments grows to better serve its users, QUICK-Comments continues to be improved upon to provide added value to teachers and provide a testbed for researchers to grow automatic grading models. Before this work, QUICK-Comments was missing key features to expand the platform to be accessed by added researchers via an upcoming Randomized Controlled Trials interface. Important functionality, such as automatic model retraining and response logging was not being performed and stalled the growth of QUICK-Comments. Now, we have designed and implemented systems to address both of these concerns, allowing QUICK-Comments to continue the development of its RCT platform. As QUICK-Comments grows, these features will prove invaluable to researchers, making it easier for them to regenerate their models based on new data and document their model's behavior to further develop them.

Future Work

QUICK-Comments is a continually expanding software project. In this section, we expand on the limitations of our work and explore suggestions for future researchers to further improve the platform in the future.

Automatic Retraining

Automatic retraining currently only works on per-problem models, but QUICK-Comments also has some models that apply to all problems. Fitting these into the database design would be as simple as creating a new type of trainable and binding the new models to that, but the process of fetching and processing the data to retrain these models has not yet been implemented.

One of the big improvements that could be pursued for automatic retraining is trying to make it run faster. A couple of solutions have been proposed for this, though neither has been implemented. Currently, when retraining occurs, every problem (with more than a certain number of responses) has a new model trained for it, regardless of any model that was trained for it previously. One simple way to improve performance would be to skip re-training models for problems that have the same or close to the same number of responses as they did when the original model was trained (assuming both models use the same algorithm). We have not assessed how much of an improvement this would bring, but it has the potential to be quite significant and could allow for more frequent but much lighter training cycles.

A more sophisticated approach could be to reuse work from a previously trained model when training a new model so that only new data has to be included. For example, in the SBERT-Canberra algorithm, we could skip generating encodings for responses that have already been encoded in a previous training cycle. Unlike the previous approach, this would require buy-in from the model implementations themselves rather than just the surrounding retraining infrastructure, but it may be viable for some training algorithms. Both approaches for improving performance could also be used together.

Digesting QUICK-Comments Logs

This body of work only included the production of logs and did not include a way to digest them. In the future, basic endpoints should be developed to allow privileged researchers to access logs for their models to learn and further develop them. These endpoints will likely be a facet of the Randomized Controlled Trials interface (which we discuss later in this section). It is also conceivable that, in the future, automated software could digest the logs themselves to summarize and create trends in model output.

Enhancing Logging Performance for Production

As QUICK-Comments grows, developing a performant system is key to ensuring overall reliability when facing production-level traffic. To ensure consistency and completion of logged information, the logging infrastructure implemented in this project runs concurrently with every request, and fully commits data before a response to the end-user is delivered. At its current scale, this is a practical and reliable implementation of logging. However, to sustain future growth, it is likely that servicing user requests should take a higher precedent over logging them.

One approach to this problem is to offload the slower database-related actions into a queue that can be handled asynchronously, pushing log data into the database at a sustained rate slower than incoming requests. While this would be equally troublesome if serving an extremely high volume of requests for a sustained duration, it would allow QUICK-Comments to offload log handling primarily during periods of low activity.

Randomized Controlled Trials

The next large step for QUICK-Comments is to develop a researcher-facing platform, to enable researchers to plug their own models into QUICK-Comments, generating comments and grades with the ASSISTments' platform dataset. In this controlled environment, researchers could run Randomized Controlled Trials to test model efficacy against one another, based on real-world teacher-student data. By bringing researchers into the RCT program, this interface can propel QUICK-Comments to aid ASSISTments' mission to continually innovate education with the latest technology powering teacher-student feedback.

REFERENCES

ASSISTments Foundation. (2019, April 25). *ASSISTments*. ASSISTments | Free Education Tool for Teachers & Students. Retrieved April 26, 2022, from https://new.assistments.org/

Baral, S., Botelho, A. F., Erickson, J. A., Benachamardi, P., & Heffernan, N. T. (2021). Improving Automated Scoring of Student Open Responses in Mathematics. *International Educational Data Mining Society*.

Benachamardi, P. (2021). *Improving Feedback Recommendation in Intelligent Tutoring Systems using NLP* (Master's Thesis, Worcester Polytechnic Institute).

Nadkarni, P., Ohno-Machado, L., Chapman, W. (2011, September). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*.