Master's thesis

NTNU
Norwegian University of Science and Technology

Jonas Eilertsen Hægdahl

Exploration of Automatic Quality Assurance of Code Reviews

Master's thesis in MACS
Supervisor: Rune Hjelsvold
Co-supervisor: Mariusz Nowostawski and Christopher Frantz
June 2023

NTNU
Norwegian University of
Science and Technology

Jonas Eilertsen Hægdahl

Exploration of Automatic Quality Assurance of Code Reviews

Master’s thesis in MACS
Supervisor: Rune Hjelsvold
Co-supervisor: Mariusz Nowostawski and Christopher Frantz
June 2023

Norwegian University of Science and Technology

Abstract

This thesis will delve into the research of quality assurance of code reviews in higher education in order to create an efficient way for teachers and teaching assistants to know the quality of the code reviews that students are writing. With a lack of research into the area, I go in without much research to reference, and in order to get the difference of how well experts compare to Artificial Intelligence when it comes to qualitatively ranking reviews, I have to make a ranking program that the AI can be integrated into. With the help of an expert and ChatGPT, I am able to get two sets of qualitatively ranked code reviews which I use to train a Text Classification Machine Learning model. Using this model, I am able to get predictions on how well the expert and ChatGPT are able to consistently rank reviews based on quality. But before I look into the results, I also create some random data predictions to have as some extra comparison. The result of this data shows that ChatGPT is more unreliable than the expert, and the random data shows that the expert and ChatGPT are less consistent in accuracy than the random data.

Sammendrag

Denne masteroppgaven vil gå inn i forskningen omgående kvalitetssikringen av kode anmeldelser for å lage en effektiv måte for lærere og lærerassistenter til å vite kvaliteten av kode anmeldelser som studenter skriver. Med en mangel på forskning i området går jeg inn uten mye forkning å refere til, og for å finne forskjellen mellom hvor godt en ekspert og kunstig intelligens sammenlignes når det kommer til kvalitativt rangering av kode anmeldelser, må jeg lage et rangeringsprogram som AIen kan integreres i. Med hjelp av en ekspert og ChatGPT klarer jeg å skaffe meg to sett med kvalitativt rangerte kode anmeldelser som jeg kan bruke for å trene en tekstklassifisering maskinlæringsmodell. Med bruk av denne modellen klarer jeg å få tak i forutsigelser på hvor godt eksperten og ChatGPT klarer å konsekvent rangere kode anmelderlser basert på kvalitet. Men før jeg ser på resultatene lager jeg også noen tilfeldige forutsigelser for å ha som ekstra sammenligning. Resultaten av denne dataen viser at ChatGPT er mer uåplitelig enn eksperten, og de tilfeldige forutsigelsene viser at eksperten og ChatGPT er begge mindre konsistent enn den tilfeldige dataen.

Acknowledgement

I would like to thank Rune Hjelsvold, Mariusz Nowostawski, and Christopher Frantz for helping me as supervisors. I would also like to thank Nipuna Hiranya Weeratunge for ranking the reviews so that I could complete my thesis.

Contents

Figures

Tables

Code Listings

Acronyms

AI Artificial Intelligence. ix, 1, 2, 4, 18, 22, 29, 30

API Application Programming Interface. 4, 6

JSON JavaScript Object Notation. 11

LINQ Language Integrated Query. 11, 12

ML Machine Learning. 2, 4, 7, 28, 29

Chapter 1

Introduction

1.1 Background and Motivation

The focus of this thesis is the quality assurance of code reviews performed by students for other students. A code review is the methodical assessment of code another person has written to find bugs, improve the quality of the code and help developers learn[1]. However, that is for a normal code review where the goal is to ensure that the product created is of sufficient quality. The code reviews that are the focus of this thesis are the reviews students write to each other to learn how to review code, give feedback and receive feedback.

The thesis will be operating on code reviews made through the use of a Computer Science Assignment Management System made and taken in use at NTNU [2]. This is a system where students are gets coding assignments which they are to complete before a deadline by posting a link to a repository of their work in the system. The students are then tasked to review the work the other students have done based on a set list of questions that they have to answer to give feedback. Questions like if the student has completed some set goal, how well the code works, how good the test coverage is or just giving feedback on what is good or could be improved.

There are however no methods available to know how good the quality of the review the students write are. Teachers that want to present examples of how to write a code review will have to read through the reviews to find a review they want to present to the class, and since each student can review more than one other student, the amount of reviews quickly becomes unmanageable for the teacher and the teaching assistants to go through.

1.2 Goals and Research questions

This thesis will be focusing on creating and investigating how well an AI can be trained to predict the quality of code reviews that are written by students and to learn if having an expert or a Generic Language based AI model set the quality of the code reviews. To accomplish this, I will have to get qualitative data on code reviews based on how good the quality of the review is which requires a method of getting the data, which will be the creation of a ranking process that can be used to rank code reviews based on quality. Alongside this, I will also research whether an expert or an AI can consistently provide the qualitative data required for quality assurance. To accomplish these goals, I will be focusing on these research questions:

- How consistent are the rankings provided by a human expert compared to a Generic Language based AI model with qualitative code review rankings?
- Can Machine Learning (ML) accurately predict the quality of a code review based on code qualitatively ranked code reviews?

As it is a new topic, there is not a lot of research into the use of ChatGPT with quality assurance of code reviews. This also means that there is not a lot of research made looking into the consistency of an expert compared to a generic language based AI model. There have been people looking into how ChatGPT[3, 4] and other AI[5] can be used to help write code reviews and how it can be used to proof-check essays written by students or be a tutor who reviews the students' writing[6], but how well can it analyze the quality of code reviews, and how does it compare to an expert?

1.3 Contributions

With this master's thesis, I will be providing the following contributions;

- A ranking process that allows both an Expert and a Generic Language based AI model to rank reviews based on the quality of the review.
- An evaluation for whether an expert or a Generic Language based AI model is more consistent with their qualitative ranking.
- Proposals for how to what can be done in the future regarding quality assurance of code reviews.

1.4 Thesis Structure

Chapter 2 contains the requirements for how the qualitative ranking and ML. In Chapter 3, the designs used in the thesis are presented with the implementation in Chapter 4. Chapter 5 provides how the evaluation was done. Chapter 6 continues by discussing the evaluation and other aspects of the master, finishing with the conclusion and future work in Chapter 7.

Chapter 2

Requirements

The end goal of this project is a system that will work in the background for the students where they can get feedback on the review they are currently writing. But before this can be developed, several steps are required to be done before feedback can be given.

2.1 Data gathering

Whichever reviews I get access to, there is a need to manage the data in a usable and simplified format that can be worked with, whether it be removing reviews due to circumstances or changing the database format for how they are stored. Having duplicate reviews is not necessary, so they need to be merged and in the case there are several languages available in the reviews, they need to be separated and dealt with depending on how many reviews there are of each language. Reviews also need to be separated by subject such that they all focus on the same area within programming.

There is no need for a subject to have several reviews which are the same. This means that reviews, like "it works well", "It works well" and "it Works well!" will all be merged into one review. This will prevent the rankings, explained in the next section, from having several different ranks filled up by the same review written in different forms.

2.2 Review Ranking

To know how good a review is compared to other reviews, there is a need to baseline the reviews. The baseline for this thesis will be a ranking of the reviews in order based on their quality. Reviews will be matched up against each other in a pair-wise ranking, with the final goal being a total ranking ordered based on the quality of the review. Each review will have a relation with every other review in the form of the other review being better, equal, or worse.

The rankings will be decided by two different kinds of parties. An expert that has deep knowledge of the course the student's reviews are from will use his expertise to rank the reviews. The features that make a good quality review will differ from expert to expert, so getting more than one expert to rank all the reviews will help mediate the issue. At the same time, an AI will also rank the same reviews. Many experts think that the reviewer's experience is really important, while not there is some disagreement on if the length of the review is important for the quality[7, 8], but what the expert and AI decides are the most important aspects of quality will not be known, however, it can be assumed that giving proper explanations of what is good and what is bad will be quite important as the code reviews are made for learning.

The expert and the AI has some different requirement for how the ranking can be presented to them. The expert can make use of both an application and a website, with the website being more universally available on different kinds of computers. Most applications are not able to run on every kind of computer, and there is a great chance that the different experts will be working with different operating systems which would require a new application for each operating system, but websites still run just as well on all operating systems. Therefore, a website will be developed for the experts to rank the reviews on. The AI however will not be easily integrated into a website, so an application will work better for it.

2.3 Artificial Intelligence

The AI used in this thesis for ranking cannot just be any standard AI. It needs to be trained on a large language dataset and be easily available. A great option for this is ChatGPT[9] which uses a generic language model. The ranking AI, ChatGPT, will be integrated into the application through the use of an API[10] to automate the process.

Another type of AI in the form of ML will be used for training and prediction of the results. ML is a practice within AI that focuses on using data to teach an algorithm how to imitate the way humans learn [11], which is why ML was chosen for generating the predictions. The data will be trained on the reviews in text format alongside the percentile rank of the review. As the review at rank 1 is the best review, the percentile ranking will therefore be set to have 0 as the best review and 1 as the worst review.

Chapter 3

Technological Design

3.1 Data Clean-up

The data used in this project will be taken from the Computer Science Assignment Management System at NTNU i Gjøvik using reviews from "Data Modelling and Database Systems Spring 2021", "Cloud Technologies Spring 2021" and "Advanced Programming Spring 2021". All the data has been cleaned for identifying information. The reviews have two options for the students to write a review in text format. The teacher can ask for the review to be written in text only or they can allow the student to write a comment that goes with a value. Reviews and comments are stored in different columns on the database. Both the reviews and comments can be considered as a type of review since the comments will explain the reason behind the value, so they need to be filtered.

In order to accommodate this change, a new database has been created which contains only the reviews and the comments. The values are not part of this new database as they do not have any influence on the quality of the review. The reviews are stored in the "Review" column on the "TextReviews" table. Student-review relations will also be removed as much as possible in this new database, except for when the review contains a link to a repository containing a student's code.

Due to time limitations, there is a limit to how large a course can be without being too overwhelming for ranking, so I am setting a limit to the number of reviews a course can contain to 800, a number which is arbitrarily chosen due to the sizes of the largest courses as it would allow them to be split into smaller courses of between 700 and 800 reviews. All the courses are represented in the database in the "Course" column on the "TextReviews" table.

Each course in the database will then be split up into smaller groups which are defined in the "Grouping" column on the "TextReviews" table. These groups are for use in the ranking process, which will be explained later in this chapter where they will be referred to as different groups in the courses.

The database can be seen in figure 3.1

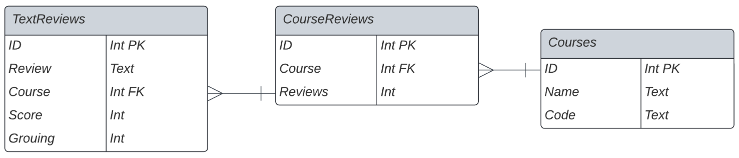


Figure 3.1: The database used for ranking the reviews.

3.2 Ranking

To implement ChatGPT with the ranking method, I decided to create a new ranking program. This allows me to seamlessly integrate the ChatGPT API into the application with the ranking program while also having the ranking program be available to the experts through the website.

3.2.1 Process

The architecture for the ranking process is quite simple with the user interacting with either the website or the application which is connected to a database and local storage. The process for how they rank the reviews is more complex and consists of three steps. In the first step named "Rank By Group", the reviews within a single group in a course will all be compared to each other until they are all compared. In the second step named "Rank Between Groups", some selected reviews from each group will be compared with each other before in the third and final step named "Finish Ranking", the ranking will be completed for the course. The reviews selected in the 2nd step are the best, worst, and median reviews in their respective group.

This method has been set up to try and minimize the number of comparisons the user has to do before they are finished. The big goal comes in the 2nd step where a select few reviews from each group are compared to each other. Should the worst review in one group, A, be compared to the best review in another group, B, with A being better than B, then every review in the first group are better than every review in the 2nd group. While it might not be likely, it is possible and can save a lot of time for the experts ranking the reviews.

For how the course and groups in the reviews work, some rules have been set:

- Reviews with different "Course" are never compared.
- Reviews with the same "Course" but different "Grouping" will be compared in the "Rank Between Groups" and "Finish Ranking" steps.
- Reviews with the same "Course" and "Grouping" will be compared fully in the "Rank By Group" step.

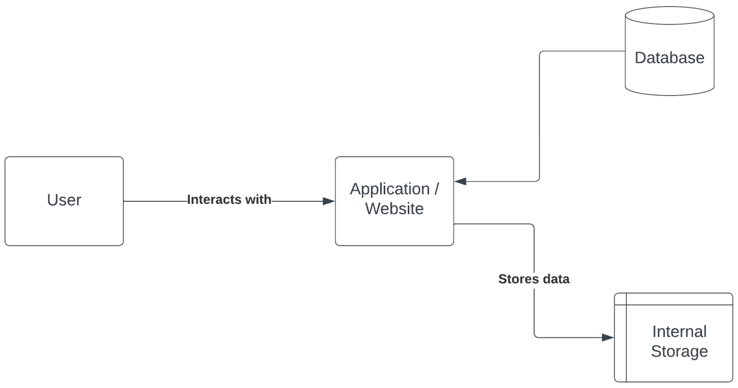


Figure 3.2: A diagram that shows how the ranking process works.

3.2.2 Comparison

When the reviews are being compared, the reviews that are being compared, alongside reviews they have previously been compared to, are matched with each other. When one review is better than the other review, the better review will be set as Review 1 which will have the reviews that are better than it and equal to it be compared to the other review alongside the reviews that are equal to it and worse. Here, all the reviews in Review 1 that are being compared will be set as "better" than all the reviews in Review 2 that are being compared. Figure 3.4 shows with an image which reviews are compared to which.

When the reviews are equal, the comparisons will be a bit different. The reviews that are better than Review 1 will be set as better than Review 2 and the reviews that are equal to or worse than it. The reviews that are worse than Review 1 will be set as worse than the reviews that are equal to or better than Review 2. Review 1 and the reviews that are equal to it will be set as worse than the reviews that are better than Review 2, better than the reviews that are worse than Review 2, and equal to Review 2 and the reviews that are equal to it.

3.3 AI

The data for the ML will be using the data from the database made during the cleanup 3.1. The only relevant columns in the database are the Review and Score columns which contain useful data for training and testing the AI. Data from the prediction depends on how the data structure the ML model used uses, which will be a type of Text Classification[12]. Text Classification is a model that takes a label column of a single element, no arrays, as the key for what it is trying to predict, and one or more sentence columns of text to train on[13].

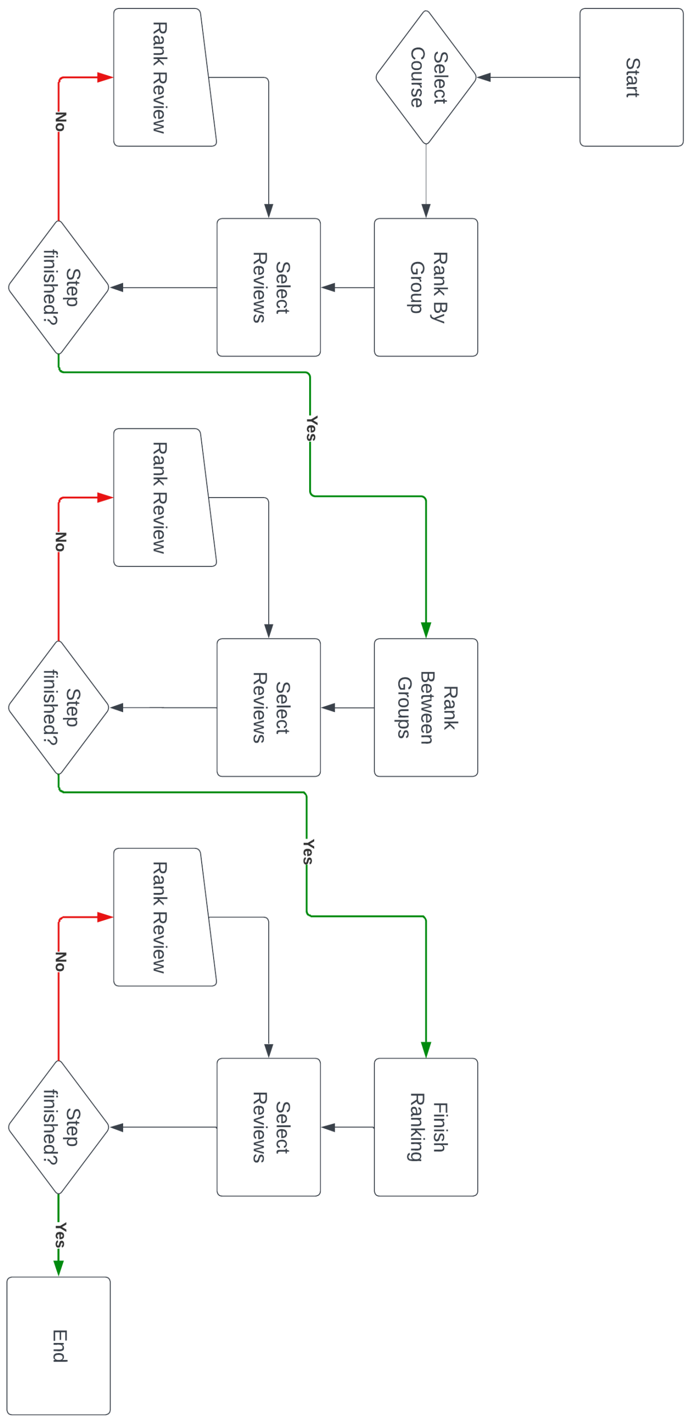


Figure 3.3: The process for how ranking works

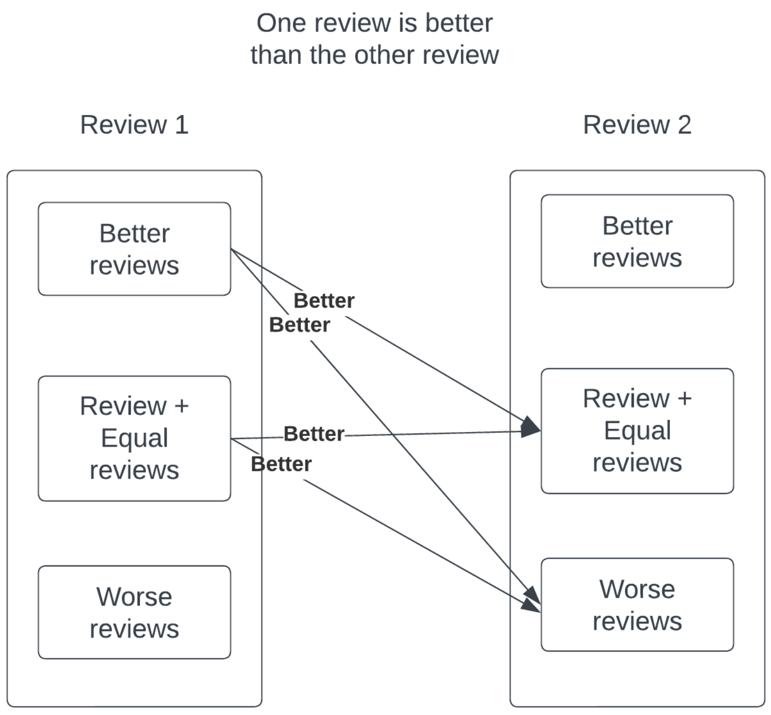


Figure 3.4: How reviews are compared when one review is better than the other review.

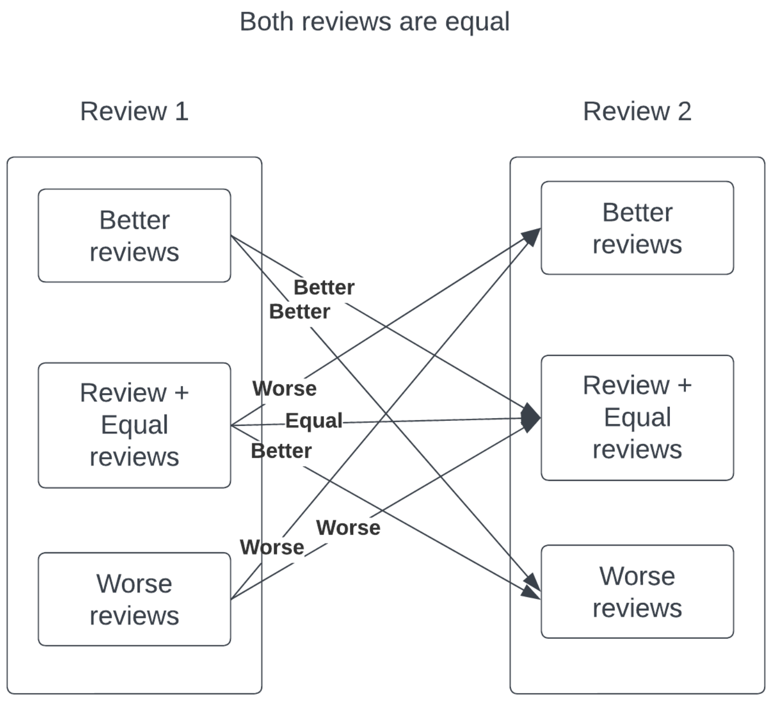


Figure 3.5: How reviews are compared when the reviews are equal.

Chapter 4

Implementation

The entire project was written in C# .NET 6 with Visual Studio as the IDE which was selected due to my knowledge of using LINQ[14] which I figured would be useful for manipulating data with ease. I also knew how to create both an API and an application with C# and .NET. The programs that connect to a database use Microsoft.EntityFrameworkCore[15] for the connection. The APIs made make use of the Newtonsoft.Json[16] for managing the JSON[17] data as the standard JSON package with C# does not function well with arrays and lists.

4.1 Data Clean-up

While the reviews in the CSAMS database are separated into two different columns, they are never in the same row. The value in which the review is written has a type attached to it, one of which is "textarea" which contains all the reviews, while other types contain the values and comments. Using this, I am able to select the review by looking for reviews where either the type is a textarea and is filled in (not all are) or where the comment is filled in.

I can then add the LanguageDetection[18] library to the project and add a variable in the database containing all the reviews for which language it is written in. Having looked through the reviews, I found there to be two languages used, Norwegian and English. Using the filter created to find the reviews with the language detector using the code shown in listing 4.1, I could fill in the added variable with the language the review was written in. Once the languages had been detected, I went through the reviews detected as Norwegian due to having less than 300 and double-checked that they were indeed all in Norwegian, changing any review written in English to English. I did not double-check the reviews written in English due to the total amount of reviews being around 9000.

Code listing 4.1: Code for detecting language

```
var doable = context.UserReviews.Where(r => (r.Type == "textarea" && r.Answer !=
    null) || r.Comment != null).ToArray();
var langDetec = new LanguageDetector();
langDetec.AddLanguages("en", "no");
```

```
foreach (var review in doable)
{
    string rev;
    if (review.Comment != null)
        rev = review.Comment;
    else
        rev = review.Answer;
    review.Language = langDetec.Detect(rev);
}

context.SaveChanges();
```

Once all the reviews had a set language, I filtered for the reviews written in English and grouped the reviews using LINQ by the course they were in. Once I had gotten the reviews grouped by course, I grouped them again with LINQ through a filter to find duplicate reviews that were the same even if they had different upper and lower cases or if one review ended with punctuation while another one did not.

Code listing 4.2: Code for removing duplicates

```
var removeDupes = reviews.GroupBy(n => Filter(n)).ToList();
```

Code listing 4.3: Code for filtering duplicates

```
private static string Filter(string text)
    {
        Regex rgx = new Regex("[^a-zA-Z0-9␣-]");
        var t = rgx.Replace(text, "");
        t = t.ToLower();
        return t;
    }
```

With duplicate reviews and languages filtered, the courses with over 800 reviews are split into smaller courses. This is done by selecting how much a course will be split up into and selecting a new ID for the "Course" column in the database, randomizing the order of the reviews, and separating them into equally sized groups with one function 4.4.

Code listing 4.4: Code for splitting a course into smaller courses

```
private static void SplitCourse(TextReviews[] reviews, int[] split)
{
    var amount = reviews.Length / split.Length;
    if (reviews.Length % split.Length != 0)
        amount++;

    for (int i = 0; i < split.Length; i++)
    {
        reviews.Skip(i * amount).Take(amount).ToList().ForEach(t => t.Course =
             split[i]);
    }
}
```

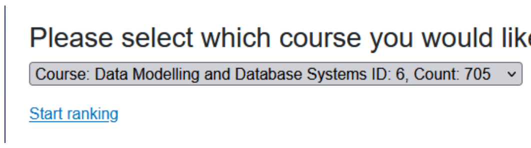
With each course split up into smaller courses for manageability, I get the number of words each review contain and use them to split each course group into small groups for the ranking process. The courses are split up into groups of around 40-60 reviews each by either splitting reviews with a certain amount of words into several groups or merging several reviews with different amounts of words in them into one group through the use of two functions 4.5

Code listing 4.5: Code for getting courses into groups

```
private void CombineIntoOneGroup(TextReviews[] reviews, int[] counts, int group)
{
    reviews.Where(n => counts.Contains(Utility.WordCount(n.Review))).ToList().
        ForEach(n => n.Grouping = group);
}

private void SplitIntoGroups(TextReviews[] reviews, int count, int[] groups)
{
    var revs = reviews.Where(n => Utility.WordCount(n.Review) == count).ToArray();
    revs = revs.OrderBy(n => rng.Next()).ToArray();
    var amount = revs.Length / groups.Length;
    if (revs.Length % groups.Length != 0)
        amount++;

    for (int i = 0; i < groups.Length; i++)
    {
        revs.Skip(i * amount).Take(amount).ToList().ForEach(n => n.Grouping =
            groups[i]);
    }
}
```

4.2 Ranking

Before the user can begin ranking the reviews within a course, they need to select which course they want to rank. The user is presented with a dropdown that gives information about the course name which is the "Name" on the "Courses" table, the ID of the course which is the number set in the "Course" column on the TextReview table, as well as how many reviews are in the course. The tables are found in figure 3.1.


Please select which course you would like to rank for

Course: Data Modelling and Database Systems ID: 6, Count: 705

Start ranking

Figure 4.1: Showcase of how users select which course they will rank.

In addition to the data shown in chapter 3.1, the "TextReviews" has three additional non-mapped lists of numbers that are used to store the IDs of reviews that are better, equal, and worse than the review. The IDs are stored in the "ID" column on the "TextReviews" table shown in figure 3.1.

4.2.1 Ranking process

The three steps of the ranking process all use the same logic to select a review from within the reviews the step has access to with only some minute differences. Of the reviews that are available in the process, it will select one of the reviews with the least amount of comparisons made as the first review. The second review is then selected by first looking at the other reviews with an equal amount of comparisons made as the first one, checking if they are already compared. If there are no reviews that are not already compared with the first one, it will look through all the available reviews and select a random review that is not compared to the first review.

Code listing 4.6: Code for selecting reviews to be compared

```
var min = reviews.Min(n => n.Comparisons());
var mins = reviews.Where(n => n.Comparisons() == min).ToArray();
var r = rng.Next(mins.Length);
var left = mins[r];

mins = mins.Where(n => n.Id != left.Id && n.IsCompared(left.Id) == false).ToArray()
     ;
TextReview right;
if (mins.Length > 0)
{
    r = rng.Next(mins.Length);
    right = mins[r];
}
else
{
    var remaining = reviews.Where(n => n.Id != left.Id && n.IsCompared(left.Id) ==
         false).ToArray();
    r = rng.Next(remaining.Length);
    right = remaining[r];
}
```

Rank By Group

The "Rank By Group" step works by going through all the groups in the course one by one comparing the reviews within the group with each other until all the reviews have been compared. The checking uses a single support function to check that the reviews are all compared. The reviews that are available for this step are all the reviews with the current "Grouping" within the "Course" that is being ranked.

Code listing 4.7: Code for checking if all reviews in a group are compared

```
private bool CheckGroupRankingComplete(IGrouping<int, TextReview> group, int count)
{
    foreach (var item in group)
    {
        if (item.Comparisons() < count - 1)
        {
            return false;
        }
    }
    return true;
}
```

Rank Between Groups

Once all the groups have had their respective reviews compared internally, three reviews are selected from each group. The reviews selected are the best, worst, and median reviews in each group and are done through one function 4.8. The reviews gotten from the function are the only reviews this step has access to. The program will continue to loop until all the reviews are compared with each other using another function 4.9 that gets the reviews with their compared count within the ranking step for the purpose of selecting reviews. This function will only count the reviews that the step has access to when counting up the number of reviews a review has been compared to.

Code listing 4.8: Code for getting reviews for the Rank Between Groups step

```
public static List<TextReview> GetSignificantReviews(IGrouping<int, TextReview>?
     group)
{
    List<TextReview> reviews = new();
    var g = group.OrderBy(n => n.BetterReviews.Count()).ToArray();
    reviews.Add(g[0]);
    reviews.Add(g[^1]);
    reviews.Add(g[g.Length / 2]);
    return reviews;
}
```

Code listing 4.9: Code for getting reviews with their compared count

```
public static (TextReview review, int Count)[] CheckSignificantReviewComplete(List<
     TextReview> reviews)
{
    return reviews.Select(n => (n, reviews.Where(c => n.IsCompared(c.Id)).Count()))
         .ToArray();
}
```

Finish Ranking

Once the ranking between some reviews from each group is done, it is time to finish the ranking process. This final step has access to all the reviews in the course and will continue the process of comparing reviews until every review has been compared with all the other reviews in the course.

4.2.2 Comparing reviews

Once the two reviews for comparison are selected in any of the steps, the user, an expert in this case, will get the two reviews up on the screen with three buttons as shown in 4.2. The three buttons are for selecting which review is better with the leftmost button for the review on the left, the same for the rightmost button, and the center button for when the reviews are about equal.

When a choice for which review is better has been made, the process for comparing the reviews alongside the reviews they are already compared to starts. The reviews are first added to each other's list, which is explained at the end of chapter 4.2, based on which review is better or worse. New lists are then generated based on the model in chapter 3.2.2, from which it loops through the lists and compares the reviews.

Code listing 4.10: Code for review comparison when a review is better than the other

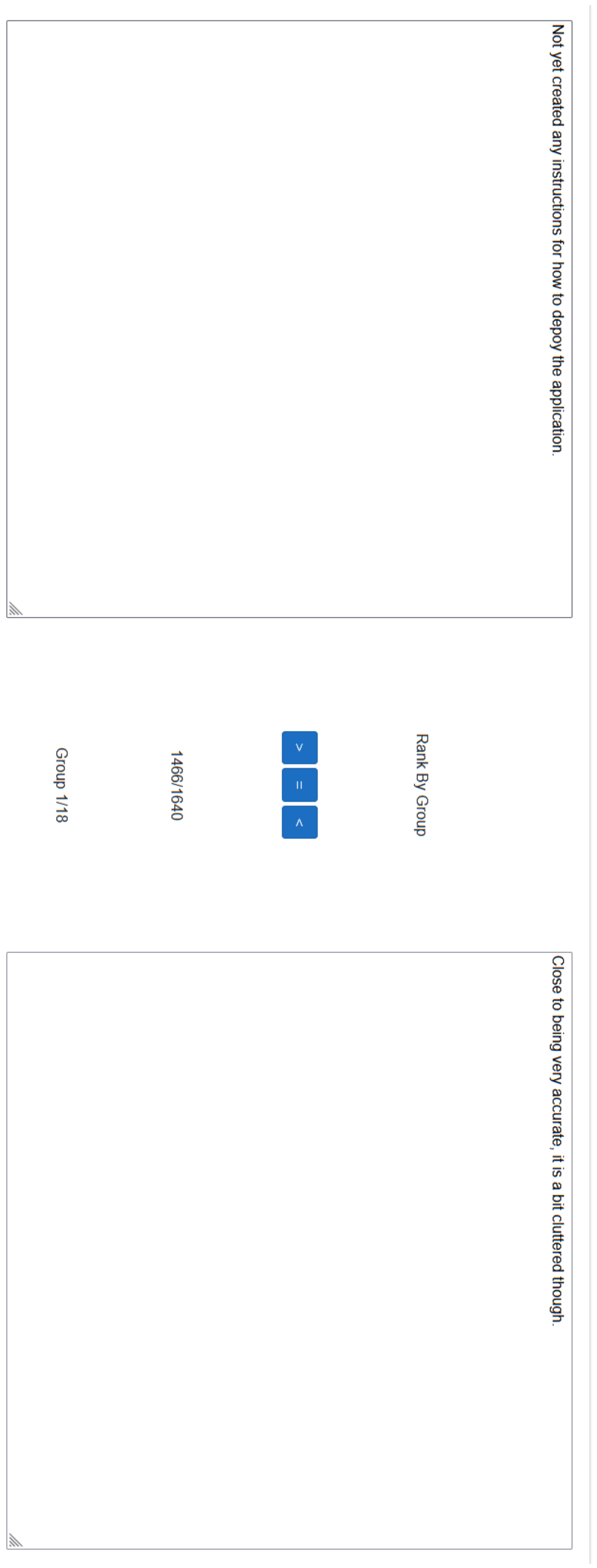
```
public static void DifferentComparison(TextReview better, TextReview worse,
    TextReview[] reviews)
{
    better.WorseReviews.Add(worse.Id);
    worse.BetterReviews.Add(better.Id);

    var betterReviews = better.EqualReviews.ToList();
    betterReviews.AddRange(better.BetterReviews);
    betterReviews.Add(better.Id);
    var worseReviews = worse.EqualReviews.ToList();
    worseReviews.AddRange(worse.WorseReviews);
    worseReviews.Add(worse.Id);

    foreach (var review in betterReviews)
    {
        var betRev = reviews.Where(n => n.Id == review).First();
        foreach (var worseReview in worseReviews)
        {
            if (betRev.IsCompared(worseReview) || review == worseReview)
                continue;
            var rev = reviews.Where(n => n.Id == worseReview).First();
            betRev.WorseReviews.Add(rev.Id);
            rev.BetterReviews.Add(betRev.Id);
        }
    }
}
```

4.3 ChatGPT ranking

The process of ranking reviews with ChatGPT is made with the usage of the OpenAI[19] library. It allows for asking it questions through prompts and selecting a model. An answer is then received and just needs to be parsed before going on to the next comparison. The prompt used for having the AI rank the reviews is: "Compare the following 2 code reviews for which review is better, or if they are equal, separated by an | after the colon and answer with only "First", "Second" or "Equal":First Review|Second Review"

Code listing 4.11: Code for asking AI to compare reviews

```
var key = File.ReadAllText(folder + "/key.txt");
var openai = new OpenAIAPI(key);
```


Not yet created any instructions for how to depoy the application.

Close to being very accurate, it is a bit cluttered though.

Figure 4.2: Showcase of how users select which review is better.

```
CompletionRequest request = new CompletionRequest();
request.Prompt = $"Compare␣the␣following␣2␣code␣reviews␣for␣which␣review␣is␣better,
    ␣or␣if␣they␣are␣equal,␣separated␣by␣an␣|␣after␣the␣colon␣and␣answer␣with␣only␣
    \"First\",␣\"Second\"␣or␣\"Equal\":{LeftReview.Review}|{RightReview.Review}";
request.Model = OpenAI_API.Models.Model.DavinciText;
request.MaxTokens = 100;

var completion = await openai.Completions.CreateCompletionsAsync(request);
```

4.4 Data manipulation for AI predicting

Before the Text Classification AI can begin training on the data, the comparisons need to be made into useful data for the predictions. This is done by counting up how many reviews are better than it and dividing that number by the number of reviews there are in the course of the review for a value between 0 and 1, and the closer the number is to 0, the better it is using the code in listing 4.13

Code listing 4.12: Code for getting useful values before Text Classification

```
public static void SetReviewScore(APIDbContext context, string folder)
{
    var revs = context.TextReviews.GroupBy(n => n.Course).ToArray();
    foreach (var reviews in revs)
    {
        foreach (var review in reviews)
        {
            if (File.Exists(folder + @$"\{review.Id}.txt"))
            {
                var data = File.ReadAllLines(folder + @$"\{review.Id}.txt");
                if (data.Length != 0)
                {
                    review.Score = (float)Math.Round(data.Where(n => n.StartsWith("
                        Worse")).Count() / (float)reviews.Count(), 2);
                }
            }
        }
    }
    context.SaveChanges();
}
```

4.5 AI prediction

Text Classification is done with the use of Microsoft.ML, Microsoft.ML.Torchsharp [19] and Torchsharp [20]. The predefined Textclassification model is used for the training with the review text being trained with the score placement of each review. Training data is saved so that retraining is not required unless it is for an update. The predictions come in an array with lengths above 70, however, it is the 2nd element that is the most consistent with its predictions being within the 0-1 range. Most other elements are mostly either in the negative range or above 1.

Code listing 4.13: Code for training the AI

```
public static void CreateModel(AIModel[] reviews, string file)
{
    var ctx = new MLContext()
    {
        GpuDeviceId = 0,
        FallbackToCpu = true
    };

    IDataView trainingData = ctx.Data.LoadFromEnumerable<AIModel>(reviews);
    var pipeline = ctx.Transforms.Conversion.MapValueToKey(outputColumnName: "Label
         ", inputColumnName: nameof(AIModel.Score))
    .Append(ctx.MulticlassClassification.Trainers.TextClassification(
         labelColumnName: "Label", sentence1ColumnName: "Review"))
    .Append(ctx.Transforms.Conversion.MapKeyToValue("PredictedLabel"))
    .AppendCacheCheckpoint(ctx);
    ITransformer trainedModel = pipeline.Fit(trainingData);
    ctx.Model.Save(trainedModel, null, file);
}
```

Code listing 4.14: Code for AI predictions

```
public static AIModel[] PredictScores(AIModel[] reviews, string trainedData)
{
    MLContext mlContext = new MLContext();

    DataViewSchema predictionPipelineSchema;
    ITransformer predictionPipeline = mlContext.Model.Load(trainedData, out
         predictionPipelineSchema);

    IDataView data = mlContext.Data.LoadFromEnumerable(reviews);
    PredictionEngine<AIModel, AIOutput> predictionEngine = mlContext.Model.
         CreatePredictionEngine<AIModel, AIOutput>(predictionPipeline);
    var predictions = reviews.Select(n => predictionEngine.Predict(n)).ToArray();
    var transformedData = predictionPipeline.Transform(data);

    for (int i = 0; i < predictions.Count(); i++)
    {
        reviews[i].Score = predictions[i].PredictedScore[1];
    }
    return reviews;
}
```

Chapter 5

Evaluation

Through the help of an expert, I was able to get an expert's evaluation on two of the courses, both of which originated from the course Data Modelling and Database Systems Spring 2021. To evaluate the rankings from ChatGPT at the same level as the expert, I used ChatGPT to rank the same two courses. With the two courses, I set up one of them to be training data while the other is used as test data. I also checked how the trained data compared to completely random data and how close the random data would be to the trained expectations.

All the values should in the end be between 0 and 1, with 0 being the best prediction while 1 is a horrible prediction. The training data also trains on values between 0 and 1, with closer to 0 meaning the review is better while closer to 1 means the review is worse.

Both the expert and ChatGPT use the same course for training their respective Text Classifications. They also use the same course for their test data so that all the data is the same except for how they have ranked the courses. The evaluation for how well their predictions are is gotten by running the test data through the function shown in listing 4.14. One part of the function that is a bit misleading is that the actual data for the test data is also sent in with the reviews, however, what they are does not matter and they have just changed to the prediction afterward. This misleading part can be changed to find the difference between the actual value and the predicted value by changing the for-loop to the one shown in listing 5.1.

Code listing 5.1: Code for the difference between prediction and actual value

```
for (int i = 0; i < predictions.Count(); i++)
{
    reviews[i].Score = Math.Abs(reviews[i].Score - predictions[i].PredictedScore
        [1]);
}
```

As stated in chapter 4.5, the values are mostly within the range of 0-1 when using the 2nd element in the list, however, it does not mean that all the values are in the range. This can be seen in how at least one prediction is off by 1.4 of the actual value. This is quite a large difference, but since the Text Classification does

Table 5.1: Prediction Results

Ranker	Min	Median	Max	Mean	Q1	Q3
Expert	0.00	0.46	0.99	0.44	0.31	0.59
AI	0.00	0.57	1.40	0.59	0.38	0.78
Random	0.00	0.29	0.96	0.33	0.14	0.50
Random Expert	0.00	0.24	0.78	0.27	0.12	0.40
Random AI	0.00	0.23	0.80	0.26	0.12	0.39

not consistently predict the rank of the review to be between 0 and 1, there is not a lot to do. Selecting different elements from the prediction for each prediction so that they all would fall in the 0-1 range would not make for a true comparison as it would be cherry-picked data.

I am also checking how the trained data performs in evaluation compared to how accurate it would be to just randomly select a value. This is being done in three different ways. For the first two, I randomly generate random numbers between 0 and 1 and subtract them from the actual scores using the code in listing 5.2. This gives me two sets of comparisons where I have random values to predict the actual values which can be used as a guideline for how well the Text Classification is working.

For the last way, I also replaced the score in listing 5.2 with another random number between 0 and 1 to get a difference between two random numbers between 0 and 1. This can help see if anything interesting is happening with the Text Classification values or the random values.

The differences of all are then put into a box plot through the use of the website `https://www.statskingdom.com/boxplot-maker.html` for values and `https://www.imathas.com/stattools/boxplot.html` for making the plots readable.

Code listing 5.2: Code for comparing random values to actual values

```
string t = "";

for (int i = 0; i <scores.Count(); i++)
{
    t += Math.Abs(scores[i].Score - Random.Shared.NextSingle()) + "\n";
    t += scores[i].Score + "\n";
}
```

The data shows that randomly generating a ranking for the reviews is more consistent than using a trained AI to predict the ranking as all the values are consistently lower than the predicted values from the expert. Using random data compared to the actual values of both the expert and ChatGPT data sets shows that they are more consistent with full-on random data coming in afterward. There is also an interesting difference in how close the data for random values differing from the actual values are closer to 0 than the completely random data.

Between the expert training set and the ChatGPT training set, the expert is more consistent than ChatGPT. ChatGPT's predictions often end up far from the

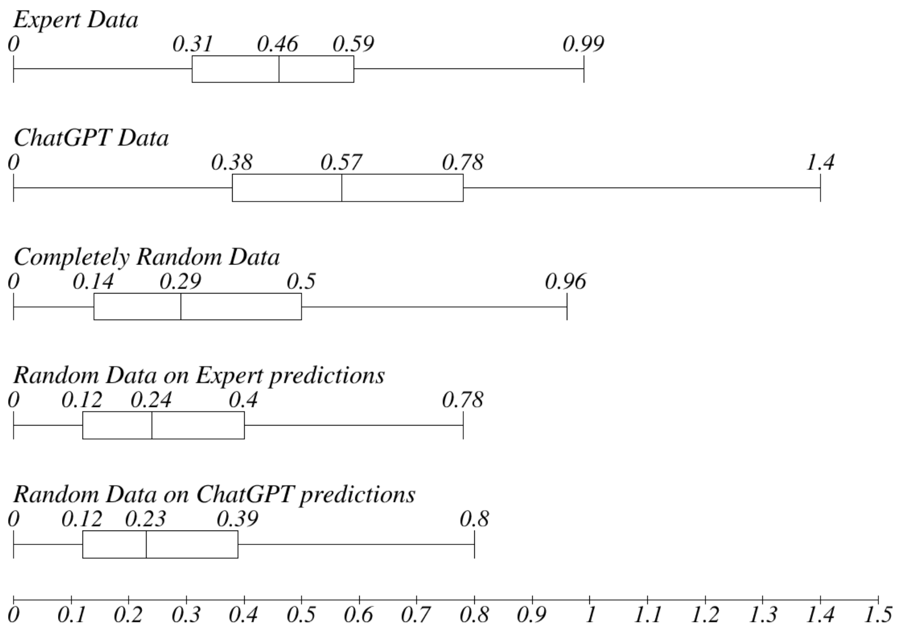


Figure 5.1: The box plots of the data.

actual value, with over half of them being more than 0.5 off. The expert's predictions, however, have more than half of them with the predicted value being off by less than 0.5 of the actual value.

Chapter 6

Discussion

A major part of the thesis went into programming the ranking application/website. Getting it to work did not take a lot of time, only about one month, but it was only working in theory. One of the courses had 337 reviews in it, so I was using it as a testing course for how effective the program was, which it was not. With the fastest time, without saving anything, it took 14 seconds for the program to run through the choices and comparisons when every review was selected as equal. If I took a random choice each time, it took almost 15 minutes to rank all the reviews when a choice was made instantly. This was just not feasible to work with, so I had to reduce the time it took as the larger courses would take days in computer time to finish at the pace it was working.

After quite a bit of time working to get the time down over several issues, and completely rewriting large parts of the code for the project, it ended with the small course taking only 12.6 seconds when the UI was being updated between each comparison with a random choice. Without updating the UI, the computation time for it went down to less than half a second. With this, I was at a time to test with the largest course, which at that time was at over 5600 reviews. This behemoth took between 40 minutes and 1 hour of calculation time which was acceptable enough for the project, but it did also require 14 GB of RAM.

The ranking using 14 GB of RAM was also something that required to be fixed to make it feasible as most modern computers are bought with only 8 GB, so even more time was spent on fixing this. I originally had the reviews keep lists of other reviews, but this took a lot of RAM, so it was changed to a hash set of the review IDs, which is the final improvement made to make the project feasible putting the RAM usage at around 1 GB which all modern computer can run. All the code is remaining in the project, which is available in the appendix, as new functions were created over editing the old ones.

Due to this, I learned the hard way how important keeping optimization in mind while coding was made quite apparent. Had I been coding with even some optimization in mind, I could have spared up to a month of progress which could have been used to rank reviews for more data. The ranking started quite late due to these issues which caused me to not get as much data as I could have gotten.

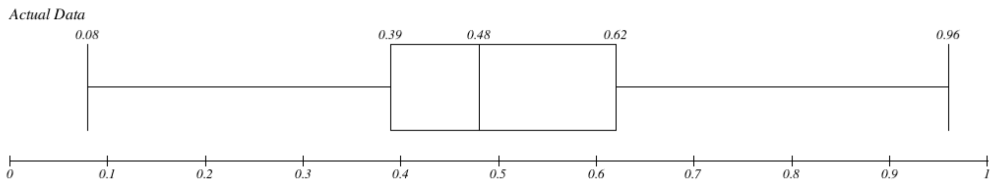


Figure 6.1: Box plot of expert ranking data

6.1 Results

It took the expert more than 1 week to rank a single course with 705 reviews, so if I had the extra time to get reviews ranked, the results would have been based on even more data and would likely be quite a bit different. With just 2 or 3 more courses at 705, I would have been able to train the AI for predicting scores a lot more, and the results themselves would definitively have changed. How I cannot say, but I believe that both the expert ranking and AI ranking would have been more accurate with their predictions.

As for the results themselves, randomly giving a rating is more accurate than using both the expert and ChatGPT data. It is interesting to note that when comparing the random data to the trained data, they are more accurate than when random prediction is compared to random values. One possibility for why this is the case could be that a lot of the actual values ended up at around the 0.4-0.6 range which makes the random values more likely to be closer to them. I reason this is the most likely case since if the actual values are indeed bunched between 0.4 and 0.6, then the highest difference between an actual value and the random value would be 0.6 at worst, with a much shorter difference being much more likely.

Having checked the actual values for one of them by creating another box plot 6.1 using the actual data from the expert data, this does appear to be the case, and after inspecting the code, I did notice a bug where reviews compared each other as worse instead of worse and better, that the tests did not catch which was the cause of the numbers clumping up in the 0.4-0.6 range, but it is too late to fix this. But even with the bug, the way the two different courses are ranked stays consistent, so while the data is warped, it is warped in a semi-predictable pattern which does not invalidate the predictions the Text Classification model has made.

Knowing that most values are around 0.4-0.6, it seems that the ChatGPT predictions tend to rank new reviews mostly closer to 0 or 1. This can be seen in how the Q1 difference is at 0.38, putting most of the predictions in between Q1 and the median at 0.57 in the values between 0.8 to 1.2 and -0.2 to 0.2 ranges. There is also the fact that some of the predictions are more than 0.8 away. Even going as far as being 1.4 off, which is well outside the range for values no matter what the prediction was. This shows that the ChatGPT ranking is extremely inconsistent, and is quite prone to going outside the range of the actual value range.

The expert prediction however has the Q3 at 0.59, which is 0.02 points above the median of ChatGPT predictions, with the expert's median value falling between the Q1 and the median value of the ChatGPT predictions. The Q1 also has a lower score which shows that overall, since the closer a value is to 0, the better the value is, and the expert predictions are more consistent than ChatGPT's predictions.

6.2 ChatGPT ranking

We also learn that even with the bug causing some bad data, the expert ranking is more consistent than the ChatGPT ranking. My theory behind why this is the case is that the AI ranking was mostly choosing values at random. While working on setting the AI up for the ranking, and testing some predictions with ChatGPT, I noticed that ChatGPT was prone to completely going off-topic, starting to react to the reviews or thinking they were feedback, or starting giving the same result every single time.

The first time I gave the prompt, it would always give a reasonable answer, but it required the answer to be read. Due to this, I reasoned that I had to give the AI the prompt at the same time as I gave it the reviews. I also required the answer from the AI to be parseable, so I had to ask it to answer with a restriction at the same time as well. While this mostly worked, it would at times add a random comment about the reviews alongside the answer, or give two answers at the same time. Due to this, I had to add a manual function that allowed me to read the response and type in the answer. It happened to around 2-3% of the answers.

When I received two answers at the same time, like "first|second", I decided that since it answered with both reviews, I would set it as equal. But it did not always just answer with some extra text or two answers at the same time when failing. There were some instances that the AI would simply refuse to answer at all. This mostly occurred when one of the reviews was telling the reviewer that something was missing in a short form. Often, the AI would just reply to these with the review itself, at which point I had to decide to give it to the other review.

Having to perform the manual parsing of some of the answers also allowed me to catch one example where which makes me believe that the AI was choosing an answer at random. When asked which was better between;

> "The model is a bit messy concidering endpoints and relations. Make it easier to implement the entity types that the project case asks for in the first place, and then add the entity and relationships you feel you lack to make the database work. I would recommend to pull out e.g. weight and size as own entity types, because there will be a lot of redundancy in data. You have chosen a solution using "place order" as an decision entity. This is probably a good solution, but I think I would rather go for a solution that avoid this."

and

> "Well done :)"

it decided that they were equal. This single answer shows that either the prompt I was using to get the answers was badly written, confusing the AI with what it had to respond with, or the AI was just giving out answers at random.

6.3 Machine Learning

Due to the programming error and limited data sample size, I cannot for sure say that using ML for predicting the quality of a code review is possible or not. The data used is faulty which in itself means that an accurate judgement cannot be decided. It is possible that it could still allow for a somewhat accurate judgment to be made, but the sample size of roughly 7 reviews per value throws that completely away as the Text Classification model was trained on barely any data at all. It would likely require at least 10 000 ranked reviews to get an accurate judgment for whether or not it is even feasible to use ML for the predictions, something that would take a lot of time to rank, especially considering how ChatGPT was seemingly giving answers at random.

6.4 Limitations

This thesis has several limitations to it. To start with, the Text Classification predictions as the result are only capable of considering whether the reviews are semantically written well or not. There is no available method for it to know if the review is correct in what is being said, or if the reviewer is just writing something well but is not correct in their statements, like a reviewer saying that a feature is missing when it is not.

It also does not help that while I have studied both AI and Deep Learning, I have not worked with Text Classification, so I do not know a lot about how to set it up using existing methods. There are likely models that allow for only one prediction to be made for each review, but it would be a model I do not know about.

There is also the fact that I was barely able to gather any data which would reasonably be able to give a good expression of how well the Text Classification predictions are. There are around 100 possible values that each review can have, but I only trained it on 705 reviews which give about 7 reviews for each value. This is not a lot of data, especially not when parts of the data are wrong due to an unnoticed programming error.

Chapter 7

Conclusion and Future work

7.1 Conclusion

This thesis has seen the creation of a ranking program that allows for experts and Generic Language based AI models to effectively rank the quality of a large set of code reviews. These rankings have then been put through a ML model for Text Classification to predict the quality of any code review that it receives. This data could then be put to use to assess both if ML can accurately predict the quality of code reviews and how consistent the human expert and Generic Language based AI model is when qualitatively ranking code reviews.

While there was a lack of data and a human error that made it difficult to assert whether the ML model could predict the quality of a review, the error does not make it impossible to ascertain the consistency of qualitative ranking between an expert and a Generic Language based AI model. The evaluations show that while both are currently quite inaccurate, caused both by the lack of data and the human error, the expert is far more consistent in qualitatively ranking code reviews compared to the AI.

7.2 Future work

There is a lot that can be done from this point. More accurate data can be gathered, and the Text Classification predictions can be redone with much larger and more accurate datasets which will give a much more accurate view of how feasible it is to create an AI that will help teachers quickly check the quality of the student's reviews. There would also not be any need to create a ranking program that can easily be integrated with a ChatGPT as all the code made in this thesis will be publically available with the bug fixed.

The Text Classification predictions can also, if feasible, be used to help students know if the review they have written is of good quality while they are performing the code review. A function like this was planned for this thesis, but scrapped due to time constraints, and would be an API that would accept the written reviews and score them, before returning them to the student which they could access through the click of a button.

The AI training in this thesis was done using a pre-defined Text Classification model, however, there were some problems with it, like how it gave more than one prediction and frequently went outside the 0-1 range. In the future, a dedicated Text Classification model can be made solely to predict the quality of code reviews.

Bibliography

[1] GitLab. ‘What is a code review?’ (), [Online]. Available: https://about.gitlab.com/topics/version-control/what-is-code-review/. (accessed: May 2023).

[2] J. Aanesen, B. F. Klausen and S. A. Danielsen, ‘Administrasjonssystem for datavitenskapsoppgaver,’ B.S. thesis, NTNU, 2019.

[3] A. Beganovic, M. A. Jaber and A. Abd Almisreb, ‘Methods and applications of chatgpt in software development: A literature review,’ *Southeast Europe Journal of Soft Computing*, vol. 12, no. 1, pp. 08–12, 2023.

[4] M. M. Rahman and Y. Watanobe, ‘Chatgpt for education and research: Opportunities, threats, and strategies,’ *Applied Sciences*, vol. 13, no. 9, p. 5783, 2023.

[5] N. Davila and I. Nunes, ‘A systematic literature review and taxonomy of modern code review,’ *Journal of Systems and Software*, vol. 177, p. 110 951, 2021.

[6] S. Atlas, ‘Chatgpt for higher education and professional development: A guide to conversational ai,’ 2023.

[7] O. Kononenko, O. Baysal and M. W. Godfrey, ‘Code review quality: How developers see it,’ in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 1028–1038.

[8] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao and M. W. Godfrey, ‘Investigating code review quality: Do people and participation matter?’ In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, IEEE, 2015, pp. 111–120.

[9] OpenAI. ‘Introducing chatgpt.’ (), [Online]. Available: https://openai.com/blog/chatgpt.

[10] G. Brockman, M. Murati, P. Welinder and OpenAI. ‘Openai api.’ (), [Online]. Available: https://openai.com/blog/openai-api.

[11] IBM. ‘What is machine learning?’ (), [Online]. Available: https://www.ibm.com/topics/machine-learning. (accessed: May 2023).

[12] TensorFlow. 'Basic text classification.' (), [Online]. Available: `https://www.tensorflow.org/tutorials/keras/text_classification`. (accessed: May 2023).

[13] Microsoft. 'Textclassificationtrainer class.' (), [Online]. Available: `https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.torchsharp.nasbert.textclassificationtrainer?view=ml-dotnet-preview`. (accessed: May 2023).

[14] Microsoft. 'Language integrated query (linq) (c#).' (), [Online]. Available: `https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/`. (accessed: February 2023).

[15] Microsoft. 'Entity framework core.' (), [Online]. Available: `https://learn.microsoft.com/en-us/ef/core/`.

[16] Newtonsoft. 'Json.net.' (), [Online]. Available: `https://www.newtonsoft.com/json`.

[17] 'Introducing json.' (), [Online]. Available: `https://www.json.org/json-en.html`.

[18] pdonald. 'Language detection.' (), [Online]. Available: `https://github.com/pdonald/language-detection`. (accessed: February 2023).

[19] OkGoDolt. 'C#/.net sdk for accessing the openai gpt-3 api, chatgpt, and dall-e 2.' (), [Online]. Available: `https://github.com/OkGoDoIt/OpenAI-API-dotnet`. (accessed: May 2023).

[20] dotnet. 'Torchsharp.' (), [Online]. Available: `https://github.com/dotnet/TorchSharp`. (accessed: May 2023).

Appendix A

Additional Material

All the code written before and during the master is available at `https://github.com/ScarredSceptile/MasterWork`. All code referenced in the thesis was written during the thesis.

While the data has been cleared of data that can be used to identify the students that wrote the reviews, there is the possibility that a link to a repository still remains in the reviews. The reviews are therefore not available in the repository.