ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Automated English Grammar Learning

Master's Thesis

Yoel Zweig

6 September 2021

Advisors: Prof. Dr. Mrinmaya Sachan, Chris Tanner Ph.D.
Department of Computer Science, ETH Zurich

Abstract

Developing good educational applications in general and language learning tools in particular has the potential to help students of all proficiency levels improve their language skills - even without a human teacher.

In this thesis, we propose a novel approach to automatically generate grammatical exercises for different topics like capitalization, singular-plural use, verb conjugation, et cetera, using a special type of grammatical error correction sequence tagger. We evaluate the suitability of different memory models to train students with the generated exercises and validate our approach with an empirical study on Amazon Mechanical Turk.

We show a significant improvement in student English grammar performance after 10 days of short Mechanical Turk study sessions, both overall ($p \leq 0.01$) as well as for each grammar area separately ($p \leq 0.05$).

The code for the exercise generation, the memory model analyses, and the Mechanical Turk survey is publicly available[i].

[i] https://github.com/Y-o-Z/master_thesis

Acknowledgements

I would like to thank my thesis advisors Chris Tanner and Prof. Mrinmaya Sachan for all of their insights and ideas as well as for their encouragement to try new things and explore beyond where I would have gone on my own.

Contents

Chapter 1

Introduction

Nowadays, a vast number of online resources are available for language learning. Machine translation systems like Google Translate and DeepL offer instantaneous high-quality translations from one language to another, dictionaries like Merriam-Webster and LEO provide detailed explanations about the meaning of specific words, and educational language applications like Duolingo, Mondly, and Memrise offer whole language curricula.

There are many reasons one might prefer an online or mobile application language course to the standard classroom setting. You can go at your own speed and focus on the things that interest you most. Of course, digital language courses are also perfectly compatible with standard language classes. Still, putting together good language courses requires a lot of work. In this thesis, we explore how to automatically generate exercises for a wide variety of English grammar topics, potentially reducing the workload for digital language course creators and human tutors alike. The combination of automatic exercise generation with memory models is especially interesting. Having a model of a student's strengths and weaknesses allows an application to train said student more effectively by focusing on weak spots and shortcomings.

The main technical contribution of our work is a new method for automatic exercise generation based on sequence taggers. We use a memory model for exercise selection and investigate three different research questions on Mechanical Turk:

- Do students perform better after having trained under a memory model as compared to a random exercise selection model?
- Do students perform better after having been provided with additional grammatical explanations during training as compared to solutions only?
- Do students, regardless of exercise selection model and grammatical explanations, perform better after having trained with the generated exercises?

Additionally, we discuss the suitability of Mechanical Turk for evaluating student learning. While we are not the first to use Mechanical Turk to investigate student learning[1], we believe that our experiences could nevertheless be valuable for future similar projects.

Chapter 2

Memory Models

This chapter introduces the concept of a memory model and explains the design, strengths, and weaknesses of the two main memory models used in this thesis. Other models are mainly used comparatively to highlight or explain design aspects.

Memory is elusive. It's very difficult if not impossible to explain why we remember one thing and not another. That being said, if we broaden our perspective and look at the rules that govern human memory as a whole, we can notice some patterns and trends. The German psychologist Hermann Ebbinghaus pioneered the experimental study of memory and introduced the so-called forgetting curve in his book *Memory and Forgetting*[2]. After having studied a number of nonsense syllables, Ebbinghaus investigated the rate of forgetting, that is the percentage of remembered syllables as a function of time:

$$r = e^{-t/s} \tag{1}$$

where r denotes memory retention, t denotes time passed, and s denotes the relative strength of a particular syllable in memory. According to this formula, memory decays exponentially over time and mastery over the item in question controls the speed of this decay. Ebbinghaus also documented the increased efficiency of spaced practice, i.e. the spreading out of training sessions over time, as opposed to massed practice, i.e. a single long training session. This effect is called the spacing effect.

Memory models are mathematical models that predict the probability of correctly recalling an item at a given point in time. Depending on the model's complexity, a smaller or larger number of factors are considered. The Ebbinghaus's forgetting curve can be viewed as a first type of memory model, using time passed and an estimation of a learner's mastery over an item to predict memory retention.

Obviously, no mathematical model is able to make such predictions with complete accuracy. Therefore, when we talk about a memory model predicting memory retention, it is understood that this prediction may not accurately reflect a student's knowledge state.

2.1 The Leitner Model

A simple method for spaced practice using flashcards was proposed by Leitner[3]. This approach defines multiple proficiency levels, each with its own training interval. Lower levels have shorter intervals, that is to say items belonging to a lower level will be practiced more frequently, and higher levels have longer intervals. A new item starts at the lowest proficiency level and moves up if it is recalled correctly during practice. Similarly, if an item is recalled incorrectly during practice, it can either be assigned directly to the lowest level or demoted to the respective next lower level. The shortest interval is usually set to one day, but students are generally free to set the lengths of the intervals, provided there is a substantial difference between the different levels.

It is worth noting that the standard Leitner model is not a memory model, as it does not predict a probability of success or failure. Later in the chapter, we will look at a modified version of the Leitner model able to predict memory retention for an item at a given point in time.

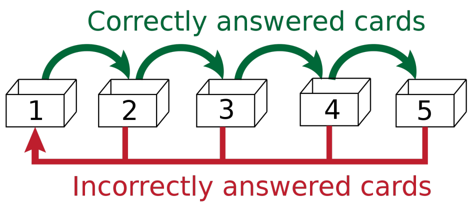


Figure 1: Leitner model version 1, retrieved from Wikipedia[17].

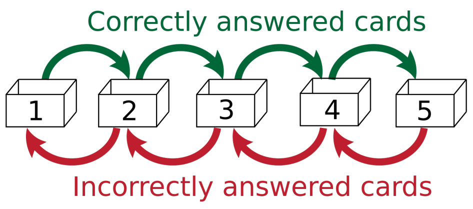


Figure 2: Leitner model version 2, retrieved from Wikipedia[18].

2.2 The Half-Life Regression Model

While being able to make predictions about memory retention is a feat by itself, memory models are additionally able to help improve student learning. This raises the question of when to best study an item. Bjork[4] introduces the notion of *desirable difficulty*, suggesting that studying is most efficient when an item is on the verge of being forgotten.

Half-Life Regression (HLR)[5] is a novel memory model for spaced repetition practice. It follows Ebbinghaus's idea of exponential memory decay over time:

$$p = 2^{-\Delta/h} \tag{2}$$

where p denotes the probability of correctly recalling an item, Δ denotes the lag time or time since the item was last practiced, and h denotes the item's *half-life* or measure of strength in the learner's long-term memory[ii]. Consider the following three cases:

- $\Delta = 0 \implies p = 2^0 = 1$: This reflects the idea that an item should always be available in memory immediately after having practiced it.
- $\Delta = h \implies p = 2^{-1} = 0.5$: Using the notion of desirable difficulty, HLR defines $p = 0.5$ as being on the verge of forgetting and therefore the optimal time to practice. According to this definition, the half-life of an item is equivalent to the time between training sessions, as $p = 0.5$ if and only if $\Delta = h$.
- $\Delta \gg h$: The lag time greatly exceeds the half-life, so the item has most likely been forgotten, i.e. $p \approx 0$.

The Pimsleur model[6] posits an exponential increase in the half-life of an item with each repeated exposure. HLR uses that assumption to estimate the half-life $\hat{h}_\Theta$ as follows:

$$\hat{h}_\Theta = 2^{\Theta \cdot x} \tag{3}$$

where x is a feature vector and Θ is a vector of corresponding weights.

[ii]HLR[5], page 1851 (4).

2.2.1 HLR features and weights

HLR does not create one model per student, but a single model i.e. a single weight vector Θ, derived from many thousands of study sessions, for all students. While Θ does not change between students, student A and student B will perform differently on different exercises, resulting in distinct feature vectors x_A and x_B. The memory model then predicts different half-lives $\hat{h}_\Theta = 2^{\Theta \cdot x}$ for these exercises. Therefore, it is the feature vector that "tailors" the model's predictions to a specific student and not the model itself. HLR employs two categories of features, though many more could be used:

- *Interaction features* denote a set of counters: x_n, the number of times a student has practiced a certain item; $x_\oplus$, the number of times that item was recalled correctly; and $x_\ominus$, the number of times it was recalled incorrectly.

- *Lexeme tag features* are a sparse vector of indicator variables. They allow the model to identify an item and assign a positive or negative weight to it. Lexeme tags reflect the model's estimation of an item's inherent difficulty and either reduce or increase the predicted half-life. These weights can give a good indication of where students generally struggle or excel.

There is also an optional fixed bias weight b, that can be used to push the predicted half-life up or down. Consider a simple example with only two words w_1 and w_2. A feature vector x for w_1 could look something like this: $\{x_n = 10,\ x_\oplus = 7,\ x_\ominus = 3,\ b = 1,\ 1,\ 0\}$ where the last two entries denote the lexeme tags for w_1 and w_2.

2.2.2 Leitner as a subcase of HLR

We stated earlier that the Leitner model does not compute probabilities, but is rather a simple algorithm for spaced practice. We can, however, use HLR with specific feature weights to create a Leitner-like memory model.

Consider the weights $\Theta = \{x_\oplus : 1,\ x_\ominus : -1\}$ where $x_\oplus$ and $x_\ominus$ are defined as explained before, giving the half-life formula $\hat{h}_\Theta = 2^{x_\oplus - x_\ominus}$. A correct response doubles the predicted half-life and an incorrect response halves it. By varying the base, we can vary how strongly the model reacts to the features. For the rest of the thesis, we will refer to this specific variant of HLR when talking about the Leitner model.

2.3 Strengths and weaknesses of Leitner and HLR

In this section, we discuss the suitability of the Leitner and HLR model for student learning, specifically for mastering different areas of English grammar.

2.3.1 Leitner Analysis

The Leitner model is simple and intuitive. Answering questions correctly leads to a half-life increase, answering questions incorrectly leads to a half-life decrease, and memory decays over time. There is no need for training, but an initial time step, that is to say the predicted half-life in the case of $x_\oplus = x_\ominus$, has to be chosen. The initial time step is usually set to one day.

There are many factors with a clear influence on memory retention that the Leitner model does not consider. Among these are diminishing returns of additional training, previous student knowledge, correlations between similar items, and the general ability to react differently to different items. Also, the assumption of a 100% recall probability directly after practice seems less justifiable for grammatical concepts than for single items such as words. Grammatical concepts might have various rules and exceptions that cannot be fully encompassed in only a few exercises let alone a single one. Unlike in the case of words, performance will improve slowly over time and not spike directly after having solved one exercise.

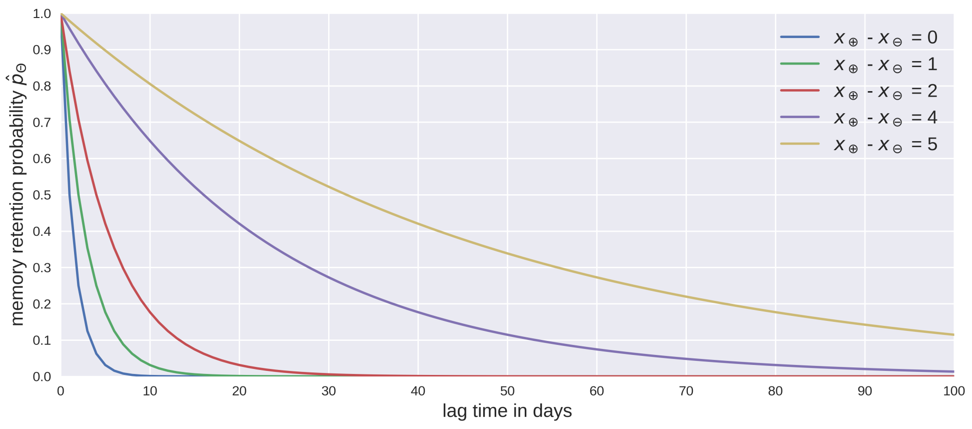


Figure 3: Leitner model predictions for different $x_\oplus$ and $x_\ominus$ values and an initial time step of 1 day.

2.3.2 HLR Analysis

So far, we have not discussed how to train HLR. The model was designed by the language-learning company Duolingo to replace the older Leitner model and is used to predict memory retention of words. Let us consider a data set $D = \{\langle p, \Delta, x\rangle_i\}_{i=1}^N$ of student practice sessions, collected from thousands of students. One data point consists of the observed recall rate p for a word, the lag time Δ, and a student-specific feature vector x for that particular word. To find optimal model weights Θ^*, we minimize a loss function ℓ:

$$\Theta^* = \arg\min_\Theta \sum_{i=1}^N \ell(\langle p, \Delta, x\rangle_i; \Theta) \tag{4}$$

HLR uses a form of the L_2-regularized squared loss function[iii]:

$$\ell(\langle p, \Delta, x\rangle; \Theta) = (p - \hat{p}_\Theta)^2 + \lambda \|\Theta\|_2^2 \tag{5}$$

where $\hat{p}_\Theta$ is the predicted recall rate and λ is a regularization parameter. Training is then performed using the AdaGrad[7] algorithm.

We talked of the observed recall rate p, but it isn't evident how we can measure this. In the end, the only thing that can be measured for certain is whether or not a student has recalled a word. It's either a one or a zero and we cannot say anything about the probability of these events after simply having observed them.

To circumvent this issue, the HLR paper defines p as follows: "In our setting, each data instance represents a full lesson or practice session, which may include multiple exercises reviewing the same word. Thus, p represents the proportion of times a word was recalled correctly in a particular session."[iv]

This definition poses certain problems, especially in the context of word learning. Let us assume that a word appears 10 times in a 10-minute study session. After its initial appearance, a student

[iii] HLR actually optimizes for both the observed recall rate p and the half-life h. The "true" half-life is unknown, but it can be estimated from p and Δ. We compute $h = \frac{-\Delta}{\log_2(p)}$ and use the loss function $\ell(\langle p, \Delta, x\rangle; \Theta) = (p - \hat{p}_\Theta)^2 + \alpha(h - \hat{h}_\Theta)^2 + \lambda\|\Theta\|_2^2$ where $\hat{p}_\Theta$ is the predicted recall rate, α is a parameter to control the influence of the half-life term, $\hat{h}_\Theta$ is the predicted half-life, and λ is a regularization parameter.

[iv] HLR, page 1851 (4).

is much more likely to know this word. After all, the HLR model is based on the assumption that memory retention is perfect immediately after having studied. This assumption is somehow suspended during the session itself. A consequence of this definition is a very high average observed recall rate $\overline{p} = 0.859$ in the HLR training data.

HLR is trained and evaluated using close to 13 million Duolingo practice sessions. Since the empirical evidence is based on this data, it is important to inspect it:

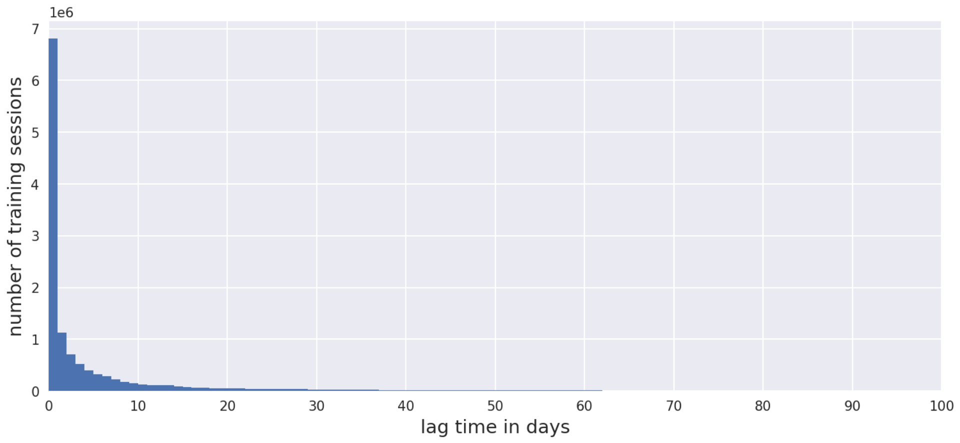


Figure 4: Distribution of Duolingo training sessions with lag time ≤ 100 days, 100 bins.

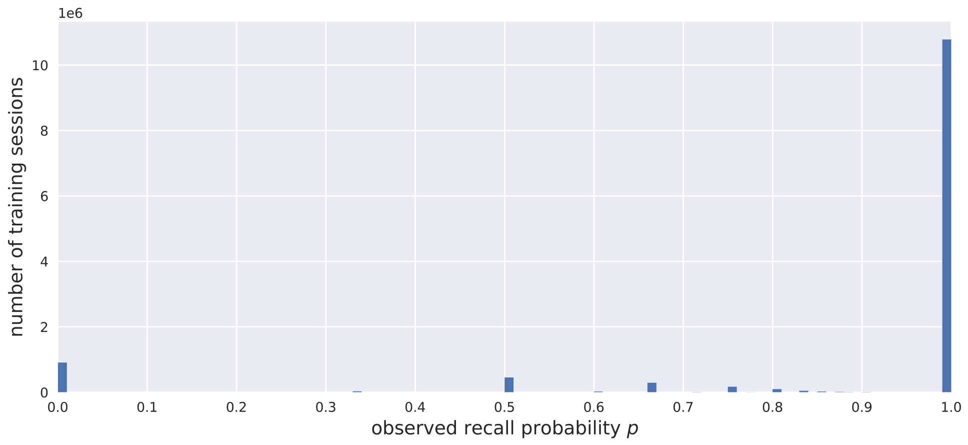


Figure 5: Distribution of Duolingo training sessions according to observed recall probability, 100 bins.

About 80% of the training sessions have a lag time of 10 days or less, about 50% have a lag time of one day or less, and about 30% have a lag time of 15 minutes or less. Since most sessions have shorter lag times, HLR will learn to focus more on short lag time interactions.

More than 80% of the training session have a recall probability of 1. This is, in all likelihood, mostly due to the measuring methodology. Consistently high recall predictions will already lead to a high accuracy, so there is a danger of HLR ignoring the less frequent low recall sessions.

While HLR does outperform Leitner, Pimsleur, and a few other methods on evaluation measures like mean absolute error (MAE), the "constant prediction model" $\hat{p} = 1$ performs even better (HLR: MAE=0.129; $\hat{p} = 1$: MAE=0.105).

Retraining an HLR model on the Duolingo data without lexeme tags and with a bias weight resulted in the weight vector $\Theta = \{x_\oplus : -0.012,\ x_\ominus : -0.2234,\ b : 7.5264\}$. The weights for $x_\oplus$ and $x_\ominus$ are negative, meaning that recalling a word correctly as well as recalling it incorrectly will lower the predicted recall probability $\hat{p}_\Theta$. HLR actually uses $\sqrt{1 + x_\oplus}$ instead of $x_\oplus$ and $\sqrt{1 + x_\ominus}$ instead of $x_\ominus$. This introduces a type of diminishing returns of training[v].

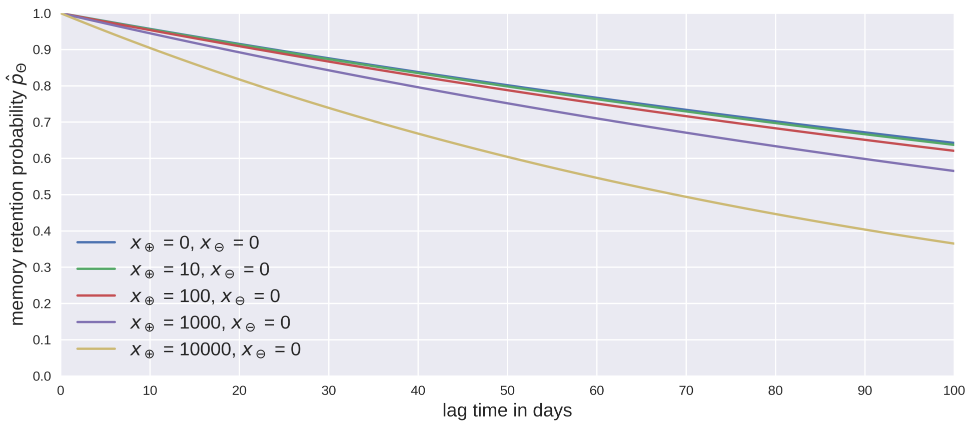


Figure 6: HLR model predictions for different values of $x_\oplus$.

The predicted performance becomes worse the more correct answers are given. Since the weight for $x_\oplus$ is very small, even a hundred correct answers in a row barely influence the model's predictions.

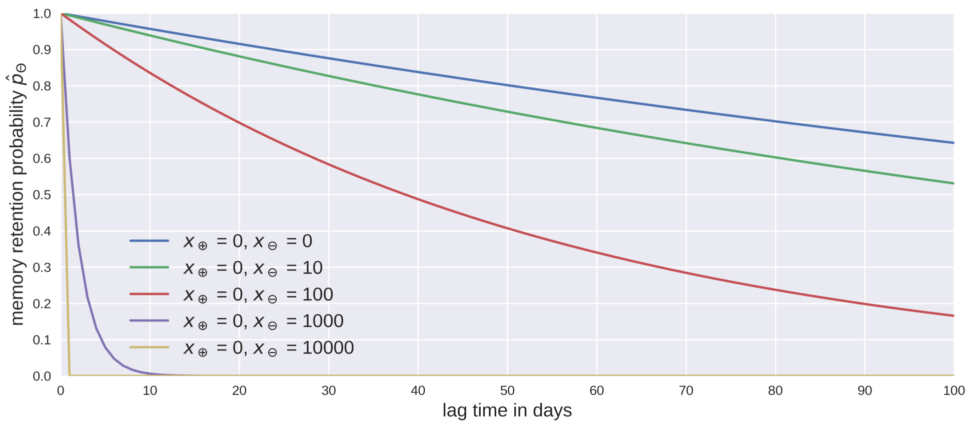


Figure 7: HLR model predictions for different values of $x_\ominus$.

[v]The same trick can be used to add diminishing returns of training to the Leitner method.

Incorrect answers have a bigger influence than correct answers and also decrease the predicted recall probability, though predictions are still high for shorter lag times and a low number of incorrect answers.

It is obvious that the model does not work the way it was intended to. In our case, HLR has merely learned to predict a high recall probability for short lag times while more or less ignoring the influence of student performance. We have actually managed to achieve the same MAE score (0.129) with a bias-only HLR model as with the standard one, showing the relative insignificance of the interaction features.

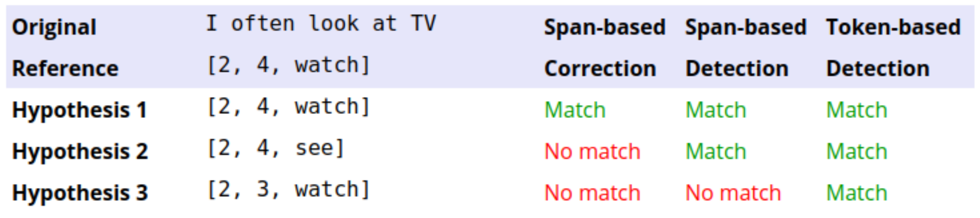
HLR has some of the weaknesses of the Leitner model, namely no consideration of previous student knowledge and of correlations between similar items. HLR also relies on good quality training data. While the measuring methodology for p is certainly unsuited for estimating the recall probability of single items such as words, it might work better for grammatical concepts. As argued before, grammatical concepts cannot be encompassed in just a few exercises and significant performance improvements are unlikely to occur within a single training session, so it is reasonable to estimate the recall rate p as the proportion of correct answers during such a session. As in the case of Leitner, the assumption of a 100% recall probability directly after practice for grammatical concepts is unrealistic for HLR as well.

Chapter 3

Grammatical Error Correction

This chapter discusses grammatical error correction (GEC) in general and GECToR, the GEC sequence tagger used for generating exercises, in particular.

GEC is the task of correcting errors such as misspellings, wrong usage of verb tenses, faulty punctuation, et cetera, in text. Consider, as an example, the sentence *the cat in the hat come back*. There is a noun-verb agreement error in this sentence, with the noun being *the cat in the hat* and the verb being *come*. Without knowing the intended meaning of the sentence, it is unclear if the error lies with the noun or if it lies with the verb. We can change the noun number (*the cats in the hat come back*), but we can also change the verb form (*the cat in the hat comes back*). Most GEC data sets contain only a single solution for an erroneous sentence, normally determined by a human annotator, so one of the above corrected sentences would be treated as incorrect. Even if we can agree on a solution, there are different metrics to evaluate corrected sentences. The widely used ERRANT scorer[8] can, for example, use span-based correction, span-based detection, and token-based detection:



Original	`I often look at TV`	**Span-based**	**Span-based**	**Token-based**
Reference	`[2, 4, watch]`	**Correction**	**Detection**	**Detection**
Hypothesis 1	`[2, 4, watch]`	Match	Match	Match
Hypothesis 2	`[2, 4, see]`	No match	Match	Match
Hypothesis 3	`[2, 3, watch]`	No match	No match	Match

Figure 8: Overview of ERRANT metrics, retrieved from BEA 2019 Shared Task[19].

In order to deal with the ambiguity in natural language, design choices have to be made; there are different approaches, each with its own merits, for different settings. It is important to take note of these choices when evaluating a GEC model.

3.1 GECToR

Sequence tagging is the task of predicting tags for each unit of a sequence. A unit can correspond to a letter, a word, a sentence, et cetera. In our case, it corresponds to a word.

GECToR[9] - short for Grammatical Error Correction: Tag, Not Rewrite - is a GEC sequence tagger. An input sentence is broken into word tokens[vi] and each token is embedded using a pretrained transformer-based embedder[10]. Each embedding vector is fed separately through two fully connected linear layers and a softmax layer is used to predict the tag. The training of the model, that is to say of the linear layers, is performed using cross entropy loss of the predicted tags together with the Adam[11] algorithm.

GECToR predicts one tag per token. Since a token might require multiple transformations and since corrections in a sentence might depend on other corrections, GECToR uses an iterative approach.

[vi]GECToR actually breaks words further into subwords and predicts a tag for the first subword of each word. However, the corresponding transformations are applied to the whole words.

An input sentence S is fed through the model, producing the corrected sentence S'. S' is then treated as the new S and fed through the model again. This process is repeated until there are no more predicted transformations (except $KEEP) or until a specified number or iterations has been reached.

3.1.1 Token-level transformations

The tags can be considered the vocabulary of a sequence tagger, with each tag representing some kind of edit operation. GECToR's default vocabulary consists of 5000 different tags that can be split into two general categories:

- Basic transformations: There are the $KEEP and $DELETE transformations and multiple $REPLACE and $APPEND transformations. $KEEP returns the original token, $DELETE returns an empty string, $REPLACE_X returns the string X, and $APPEND_Y returns a concatenation of the original token and the string Y.
- g-transformations: There are 29 so-called g-transformations. Each performs a task designed for a particular type of grammatical error, e.g. uppercasing or lowercasing a token, adding an "s" to or removing it from the end of of a token, or changing one verb tense into another. GECToR uses a verb conjugation dictionary to look up the specific verb forms.

Unlike basic transformations, g-transformations can fix a wide range of errors, since they are not bound to specific words. Apart from $KEEP and $DELETE, basic transformations are created automatically by selecting the most frequently used $REPLACE and $APPEND transformations from the training data. The default vocabulary has 3802 $REPLACE and 1167 $APPEND transformations. The vocabulary size can be freely chosen, though there is a trade-off between model size and model performance.

3.1.2 Preprocessing

In order to train the GECToR model, we need an ordered sequence of tags $(t_1, \ldots, t_N)$ for each input sentence. The training data helps explain some of GECToR's properties, so we will briefly outline how it is created[vii]:

- We start off with erroneous source sentences and the corresponding corrected target sentences. For each source sentence, we detect minimal-difference spans[viii] between source tokens $(x_1, \ldots, x_N)$ and target tokens $(y_1, \ldots, y_M)$.
- For each source token x_{j_i} within such a minimal-difference source span s_j, we search for the best-fitting subsequence $(y_{j_1}, \ldots, y_{j_2}), 1 \leq j_1 \leq j_2 \leq M$, in the corresponding target span t_j. The best fit is computed using a modified Levenshtein distance, where g-transformations are given a zero cost.
- If there are multiple transformations for a source token, we keep the first transformation that is not $KEEP. GECToR can only handle one transformation per word at a time, so we leave additional transformations for another iteration.

There are three significant areas of overlap between the different g-transformations, namely between $NOUN_NUMBER_SINGULAR and $VBZ_VB, between $NOUN_NUMBER_PLURAL and $VB_VBZ, and between all of the different verb transformations like $VB_VBZ, $VBZ_VBN, $VBN_VBD, $VBD_VBG, $VBG_VB, ...

[vii]GECToR[9], pages 2&3.
[viii]difflib Python library using a form of Gestalt Pattern Matching[12].

Tag	Example
$NOUN_NUMBER_SINGULAR	… into every {**appliances** ⇒ **appliance**} …
$NOUN_NUMBER_PLURAL	… able to improve {**relationship** ⇒ **relationships**} …
$VB_VBZ	… what he or she {**prefer** ⇒ **prefers**} …
$VBZ_VB	… networking sites {**provides** ⇒ **provide**} …
$VB_VBN	… have never {**work** ⇒ **worked**} …
$VBN_VB	… for me to {**bought** ⇒ **buy**} …
⋮	⋮

Table 1: Overview of selected g-transformations. A complete list of g-transformations can be found in the GECToR paper[9].

Consider the sentence *one lights is bright enough*. In order to fix this sentence, GECToR might use $NOUN_NUMBER_SINGULAR (noun plural to noun singular), but, since light can act as a verb as well, GECToR might also use $VBZ_VB (third person singular simple present to verb base). As mentioned before, a modified Levenshtein distance with zero cost g-transformations is used in creating the training data. When calculating the minimal-difference spans, we first check if any of the g-transformations can be applied and stop as soon as one is found. Otherwise, we use a standard add, replace, or delete operation. As for the g-transformations themselves, verb transformations are considered before singular and plural transformations. This means that the preprocessing will always favor $VBZ_VB over $NOUN_NUMBER_SINGULAR if both of are possible, even if the token in question is a noun. The same issue exists for $VB_VBZ and $NOUN_NUMBER_PLURAL.

To check if a verb transformation (e.g. add → added) can be used, a top to bottom search through the alphabetically ordered verb conjugation dictionary is performed and the first hit is returned. For the verb ”add”, $VBD (simple past) and $VBN (past participle) are identical. $VB_VBN will never be used, since $VB_VBD appears earlier in the dictionary. Whenever two or more verb forms overlap, the one that appears first will be returned regardless of other considerations. That is not necessarily an issue if one is only interested in achieving a valid correction. After all, both transformations produce the same output. It can however pose a problem for exercise generation, as we will see in the following chapter.

add_added:VB_VBD	added_added:VBD_VBN
add_added:VB_VBN	added_added:VBN_VBD
add_adding:VB_VBG	added_adding:VBD_VBG
add_adds:VB_VBZ	added_adding:VBN_VBG
added_add:VBD_VB	added_adds:VBD_VBZ
added_add:VBN_VB	added_adds:VBN_VBZ

Table 2: Conjugation dictionary entries for the verb ”add”. In alphabetical order from top left to bottom left to top right to bottom right.

Chapter 4

Exercise Generation

This chapter details our approach to exercise generation using the GECToR model. We compare our approach to synthetic error generation methods and discuss its strengths and weaknesses. We also describe how to generate the two data sets used in the Mechanical Turk survey.

GEC can be viewed as a translation task where erroneous sentences correspond to the source language and grammatically correct sentences correspond to the target language. For synthetic error generation, it is the other way around. Synthetic error data has become more important in recent years and is often used to pretrain or even fully train neural machine translation systems.

Errorify[13] applies a mixture of common deletes, inserts, and replaces to grammatically correct sentences to artificially add errors. While the error range is constrained to the available deletes, inserts, and replaces, it is easy to control error frequencies. Errorify is a rather crude method and not well suited for generating exercises for students or for fine-tuning a model, but it can be used for pretraining (as in the case of GECToR).

Tagged corruption models[14] are trained to output an erroneous sentence given a grammatically correct sentence and an error tag, with the inserted error corresponding to the tag. They are based on transformers and, unlike Errorify, able to generate a wide range of errors. Also, the error type distribution can be controlled via tag selection.

Both of these methods start from grammatically correct sentences and insert errors. Our approach starts from erroneous sentences and produces a sentence pair (S_g, S_e), S_g being a grammatically correct sentence and S_e being an erroneous sentence. Similar to the tagged corruption model, errors are divided into classes that correspond to one or multiple tags.

In this chapter, exercise generation and generation of sentence pairs (S_g, S_e) is synonymous. In the following chapter, we will describe in detail how the sentence pairs are actually used to train students.

4.1 Generating exercises

The first step in our approach is defining the error categories. An error category can consist of a single tag or a collection of tags, but there should be no overlap between the categories. Let us consider the following three cases:

- The input sentence S is grammatically correct: No sentence pairs will be generated. The sentence is broken into tokens $(x_1, \ldots, x_N)$ and for each token x_i a \$KEEP tag is predicted. Since none of the predicted tags belong to one of the defined error categories and since there are no edit operations to perform, our method moves on to the next sentence.

- The input sentence S is erroneous, but the predicted tags do not belong to one of the defined error categories: No sentence pairs will be generated. The sentence is broken into tokens $(x_1, \ldots, x_N)$ and for each token x_i a tag is predicted. For all predicted tags except \$KEEP, we store the tag identifier and the token index. Since none of the predicted tags belong to one of the defined error categories, our method simply applies the predicted edit operations, resulting in a new sentence S'. GECToR uses an iterative GEC approach, so S' is treated as the new S and fed through the model again.

- The input sentence S is erroneous and the predicted tags belong to one of the defined error categories: As before, we break the sentence S into tokens $(x_1, \ldots, x_N)$ and predict a tag for each token x_i. For those predicted tags that belong to one of the defined error categories, we store the tag identifier, the token index, and the token itself. For all other predicted tags except $KEEP, we store the tag identifier and the token index. Then we perform all the predicted edit operations, resulting in a new sentence S', which we feed through the model again.

We now have the final corrected sentence S' and for each GECToR iteration a (potentially empty) list of applied edit operations. Consider the following example:

Iteration 1:
Input sentence $S =$ *Money make world goes round*
Predictions: [Money → Money]: $KEEP, [make → makes]: $VB_VBZ, [world → world]: $KEEP, [goes → go]: $VBZ_VB, [round → round]: $KEEP
Output sentence $S' =$ *Money makes world go round*
Applied edit operations: {iteration_1: [[$VB_VBZ, 1, make], [$VBZ_VB, 3, goes]]}

Iteration 2:
Input sentence $S =$ *Money makes world go round*
Predictions: [Money → Money]: $KEEP, [makes → makes the]: $APPEND_the, [world → world]: $KEEP, [go → go]: $KEEP, [round → round]: $KEEP
Output sentence $S' =$ *Money makes the world go round*
Applied edit operations: {iteration 1: [[$VB_VBZ, 1, make], [$VBZ_VB, 3, goes]], iteration 2: [[$APPEND_the, 1]]}

Iteration 3: Input sentence $S =$ *Money makes the world go round*
Predictions: [Money → Money]: $KEEP, [makes → makes]: $KEEP, [the → the]: $KEEP, [world → world]: $KEEP, [go → go]: $KEEP, [round → round]: $KEEP
Output sentence $S' =$ *Money makes the world go round*
Applied edit operations: {iteration 1: [[$VB_VBZ, 1, make], [$VBZ_VB, 3, goes]], iteration 2: [[$APPEND_the, 1]], iteration 3: []}

To produce a sentence pair (S_g, S_e) for the tag $VB_VBZ, we look up the token index and the original token and substitute it back into S', producing the sentence $S'' =$ *Money make the world go round.* S' now corresponds to S_g. S'' corresponds to S_e and is of the type $VB_VBZ. If we try to apply these steps to the tag $VBZ_VB, then we end up with the sentence $S'' =$ *Money makes the goes go round.* By adding a token to the sentence, the $APPEND transformation in the second iteration has shifted some indices. It is also possible for a later transformation to "overwrite" an earlier transformation by changing a token a second time or to simply delete it. The below algorithm can be used to manage these issues.

After processing the stored tags, we can now substitute the original tokens back into the final corrected sentence S', as we did before. For each error belonging to one of the predefined error categories, one sentence pair is produced. If no such errors exist in our input, then we are unable to produce any output. Our approach therefore relies more heavily on suitable data than the previously discussed synthetic error generation methods do.

Algorithm: Dictionary processing

Input: dictionary of applied edit operations 'd'
Result: "processed" edit operations dictionary 'transformations'

```
# Algorithm given in Python pseudocode
transformations = []

for iter in range(num_iterations):
    current_ts = d[iter]
    # is there an overwrite for an (older) existing transformation?
    keep_indices = [1] * len(transformations)
    for i in range(len(transformations)):
        for t in current_ts:
            if t["index"] == transformations[i]["index"] and not is_append(t):
                keep_indices[i] = 0
    # keep only "untainted" transformations
    transformations = [t for i, t in enumerate(transformations) if keep_indices[i]]
    # change indices of (older) existing transformations
    for i in range(len(transformations)):
        index_shift = 0
        for t in current_ts:
            if t["index"] < transformations[i]["index"]:
                index_shift += change_to_index(t)
        transformations[i]["index"] += index_shift
    # change indices of current transformations
    for i in range(len(current_ts)):
        index_shift = 0
        # inspect "earlier" current transformations in reverse order
        for j in range(index - 1, -1, -1):
            index_shift += change_to_index(current_ts[j])
        current_ts[i]["index"] += index_shift
    # drop DELETE, REPLACE, APPEND transformations
    current_ts = [t for t in current_ts and is_g_transformation(t)]
    transformations.extend(current_ts)
```

4.2 Study data sets

We use preexisting sentences from the CoNLL-2014[15] test set and the BEA-2019[16] dev and test sets to generate the exercises with the trained ensemble model from the GECToR paper. None of these sentences have been used for training the GECToR model, so there is no undue advantage in finding and classifying the errors correctly. Note that the model was optimized for span-based correction with the $F_{0.5}$ metric. Precision is weighted twice as high as recall, so there is an incentive for the model to only predict a transformation given high levels of confidence. On one hand, this might increase the quality of the predicted transformations; on the other hand, it might lead to more uncorrected errors, since fewer basic transformations and g-transformations will be predicted and applied.

The CoNLL-2014 test set consists of 1311 English sentences, the BEA-2019 dev and test sets consist of 4886 English sentences. Our goal isn't to create the largest possible number of exercises, but only a sufficiently large number for the Mechanical Turk study. In order to raise the quality of the generated exercises, we apply a preprocessing step to eliminate unsuitable sentences. Among other things, we enforce length requirements and remove all sentences containing unusual punctuation symbols, since GECToR is mostly unable to manage them. After preprocessing, we retain 1101 CoNLL sentences and 3260 BEA sentences, giving a total of 4361 sentences.

As error categories, we use the 29 GECToR g-transformations. Using the outlined exercise generation approach on all 4361 sentences results in 1157 sentence pairs, with some error categories appearing much more frequently than others. Considering the size of the categories as well as their relevance to students, we settle on eight error groups for further use:

Error group	**Transformation**	**Example**
Capitalization	$CASE_CAPITAL	... arrived in {**singapore** ⇒ **Singapore**} ...
	$CASE_LOWER	... of the {**Managers** ⇒ **managers**} ...
Singular	$NOUN_NUMBER_SINGULAR	... into every {**appliances** ⇒ **appliance**} ...
Plural	$NOUN_NUMBER_PLURAL	... able to improve {**relationship** ⇒ **relationships**} ...
VB_VBZ	$VB_VBZ	... what he or she {**prefer** ⇒ **prefers**} ...
VBZ_VB	$VBZ_VB	... networking sites {**provides** ⇒ **provide**} ...
VB__VBN	$VB_VBN	... have never {**work** ⇒ **worked**} ...
	$VBN_VB	... for me to {**bought** ⇒ **buy**} ...
VB__VBG	$VB_VBG	... high chance of {**carry** ⇒ **carrying**} ...
	$VBG_VB	... it will just {**adding** ⇒ **add**} ...
other_verb_errors	$VB_VBD	... I went and {**give** ⇒ **gave**} ...
	$VBZ_VBN	... everything is {**bases** ⇒ **based**} ...
	⋮	⋮

Table 3: Overview of custom error groups.

After randomly shuffling each error group, the first 65 sentence pairs of each group are split into a training set of size 60 and a test set of size 5. We inspect the test sets to guarantee that for each sentence pair (S_g, S_e), S_e contains exactly one error of the correct type and S_g is a valid solution. The training sets are not inspected in this manner. A random training sentence pair (S'_g, S'_e) might therefore contain additional errors that were not corrected by GECToR and that we are not aware of. Also, (S'_g, S'_e) might be assigned to the "wrong" error group due to overlap between g-transformations.

Chapter 5

Mechanical Turk Study

In order to evaluate the usefulness of the generated exercises, we conduct a 10-day study on Mechanical Turk, an Amazon crowdsourcing platform, where the users (Turkers) solve so-called human intelligence tasks (HITs). We investigate the following three research questions:

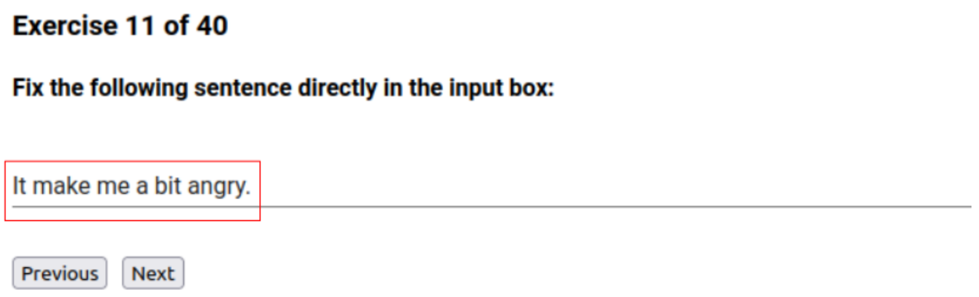
- Q1: Do students perform better after having trained under a memory model as compared to a random exercise selection model?
- Q2: Do students perform better after having been provided with additional grammatical explanations during training as compared to solutions only?
- Q3: Do students, regardless of exercise selection model and grammatical explanations, perform better after having trained with the generated exercises?

Our target group is beginner to intermediate English speakers. We limit participation to non-English speaking countries. Excluded countries are Australia, Canada, Great Britain, New Zealand, and the United States. We also require a minimum of 101 completed HITs and a 95% approval rating.

5.1 Language test

We use the test sets from the previous chapter to build a language test. Each participant solves this test twice - once at the the beginning and once at the end of the survey. This allows us to measure progress.

We have eight different error groups and for each group a test set consisting of five sentence pairs. For each sentence pair (S_g, S_e), we create an exercise in the following way:


Exercise 11 of 40

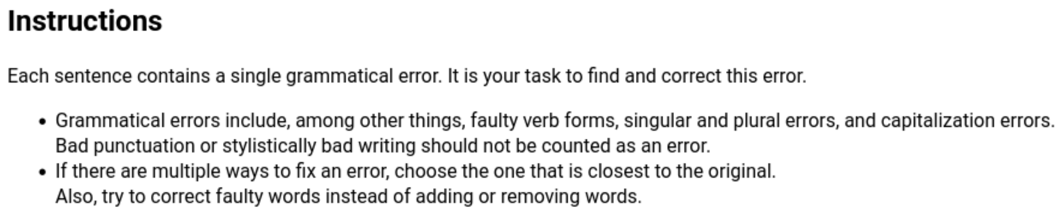
Fix the following sentence directly in the input box:

It make me a bit angry.

Figure 9: Test exercise. The highlighted sentence corresponds to the erroneous sentence S_e. The student's task is to change S_e into the solution sentence S_g (*It makes me a bit angry.*). The two buttons at the bottom are used to move between the different exercises.

In order to minimize distractions, only a single exercise is visible at a time. We have a total of 40 exercises and each exercise is graded as pass or fail. If the student's answer matches the solution, one point is awarded. The highest possible score is 40 points and the lowest is zero points. The exercises are shuffled once, so they appear in a random order. However, we use the same random order for every participant. A time lock enforces a minimum time of five seconds per exercise. This is to keep

Turkers from simply rushing through the test. We do not provide any feedback for the language test, so students will not know if they made any mistakes. This allows us to reuse the same exercises for the second language test. The exercises are shuffled once more, so the order is different than the first time. Apart from the order, both tests are identical. We also provide the following instructions at the beginning of both language tests:



Instructions

Each sentence contains a single grammatical error. It is your task to find and correct this error.

- Grammatical errors include, among other things, faulty verb forms, singular and plural errors, and capitalization errors.
  Bad punctuation or stylistically bad writing should not be counted as an error.
- If there are multiple ways to fix an error, choose the one that is closest to the original.
  Also, try to correct faulty words instead of adding or removing words.

Figure 10: Language test instructions.

5.2 Training

Aside from investigating student learning, we are additionally interested in the influence of the exercise selection model and of grammatical explanations. In order to study this influence, we split the participants into three different groups: test, control, and base.

Test group members are trained using a memory model and grammatical explanations, control group members are trained using a random exercise selection model and grammatical explanations, and base group members are trained with just a random exercise selection model. We investigate the influence of the exercise selection model by comparing the test and control groups. Similarly, we investigate the influence of the grammatical explanations by comparing the control and base groups.

A normal procedure would be to randomly assign participants to the different groups. This works well if the number of participants is sufficiently large. In our case, we use a different approach. We rank all Turkers based on their score in the first language test. The Turker with the lowest score is assigned to the test group, the one with the next lowest score to the control group, the one with the third lowest score to the base group, the one with the fourth lowest score again to the test group and so on until all Turkers have been assigned. That way, the control group should have Turkers at least as proficient as the test group and the base group should have Turkers at least as proficient as the control group. If the test group should nevertheless outperform the control group or similarly the control group outperform the base group, this could be attributed to the actual positive effect of memory models or of grammatical explanations, rather than to the more highly skilled Turkers accumulating in one particular group by chance.

5.2.1 Exercise selection models

Once a day, new training HITs are published. For participants in the test group, the HLR version of the Leitner model is used to determine how many of the eight error groups will be practiced. More specifically, an error group's half-life is predicted as $\hat{h}_\Theta = 2^{x_\oplus - x_\ominus}$ and its recall probability as $\hat{p}_\Theta = 2^{-\Delta/\hat{h}_\Theta}$. The initial time step is set to 0.98 days. We define it as slightly below 1, so that an error group with $x_\oplus = x_\ominus$ has a predicted half-life of less than one day and is therefore eligible for the training HIT the next day.

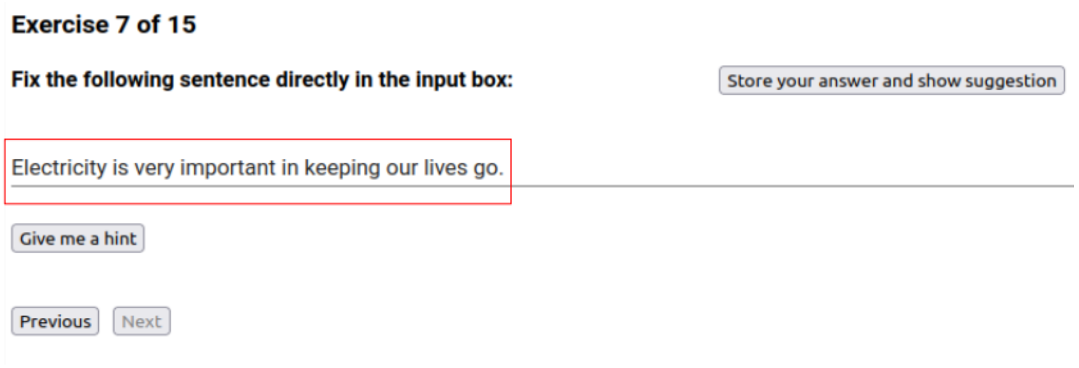
Unlike the test exercises that were inspected one by one, the training exercises might contain additional errors that were not corrected by GECToR and that we are not aware of. Sentence pairs (S_g, S_e) always differ in exactly one token t_x. Since we do not fully trust the solution sentence S_g, we score an exercise as correct if the student manages to correctly transform t_x instead of comparing the student's full answer to S_g.

Also, we incrementally increase $x_{\oplus}$ not by the number but by the percentage of exercises that were solved correctly during a training session. Successfully completing five of five exercises will therefore increase $x_{\oplus}$ by 1 and not by 5. The same goes for $x_{\ominus}$ and incorrect answers. This limits the speed with which the half-life can change.

First, we predict the recall probability for each of the eight error groups. If there are no predicted probabilities smaller or equal 0.5, then the participant receives no invitation that day. If there are three or fewer error groups with probabilities smaller or equal 0.5, then we select all of them. If there are more than three error groups with probabilities smaller or equal 0.5, then we select those three groups with the lowest probabilities, i.e. those most in need of training. For each of the selected error groups, we randomly choose five exercises from the corresponding training set that haven't yet been seen by the student.

In order to compare the different groups, we need to ensure that participants from all three groups receive roughly the same amount of training. For this, we rank the members of each group by their scores in the first language test. For each test group member t_i of rank i, the corresponding control group member c_i and base group member b_i who both also rank i in their groups will practice the same number of error groups. If t_i does not receive a training invitation, then b_i and c_i do not receive one either. In contrast to the memory model, the random exercise selection model selects error groups not based on their predicted recall probabilities, but randomly. For the selected error groups, we again randomly choose five exercises from the corresponding training set that haven't yet been seen by the student.

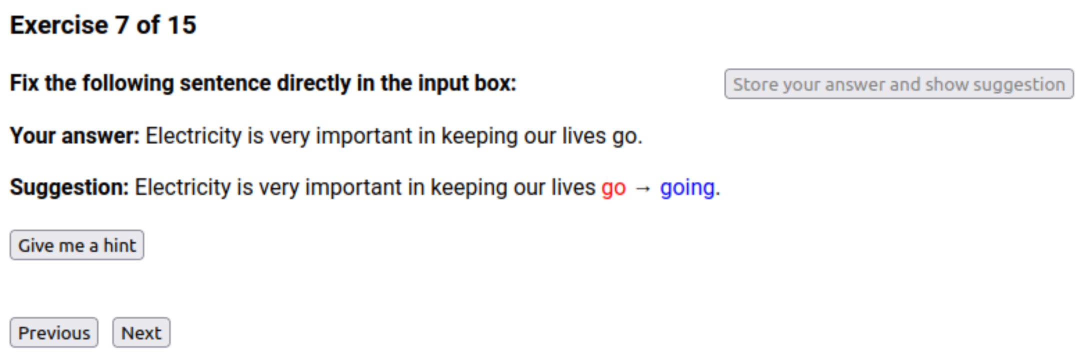
We limit the number of error groups per training to three and the number of exercises per error group to five in order to keep the training HITs short. Also, if only a few error groups can be trained each day, the importance of the exercise selection model increases, making it easier to potentially find a significant result.


Exercise 7 of 15

Fix the following sentence directly in the input box:

Electricity is very important in keeping our lives go.

Figure 11: Training exercise. The button *Store your answer and show suggestion* displays S_g and stores the student's answer. We use the term suggestion instead of solution to indicate that the GECToR produced solution sentence S_g is not always correct.


Exercise 7 of 15

Fix the following sentence directly in the input box:

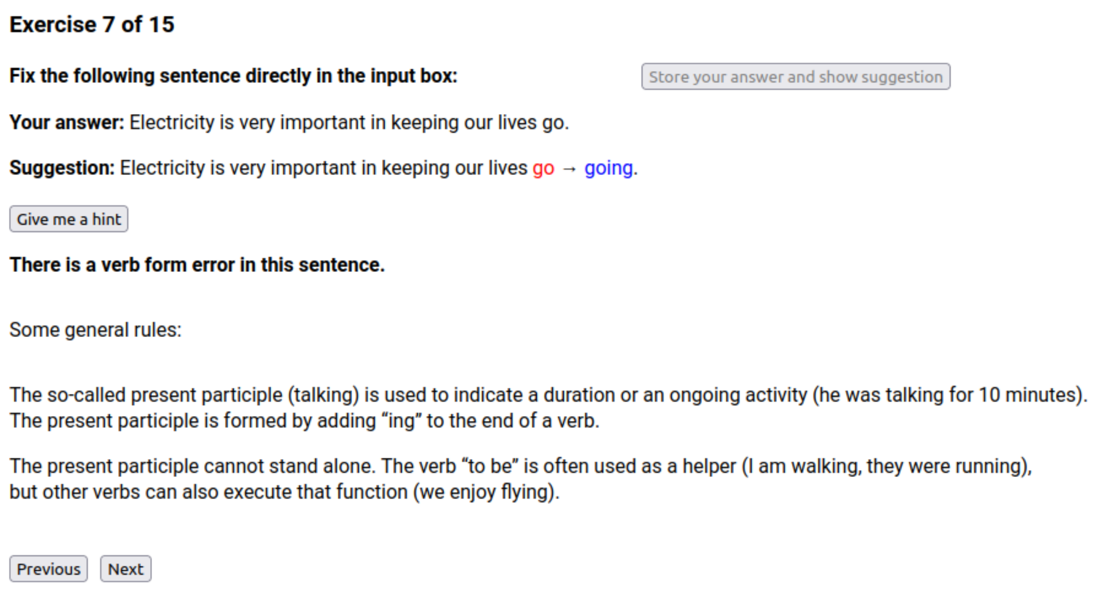
Your answer: Electricity is very important in keeping our lives go.

Suggestion: Electricity is very important in keeping our lives go → going.

Figure 12: Training exercise. The input box is hidden once the solution is displayed, so that students can no longer change their answers. Students cannot proceed to the next exercise before viewing the solution. Additionally, there is a time lock that enforces a minimum time of seven seconds per exercise.

5.2.2 Grammatical explanations

For each of the eight error groups, we try, with varying degrees of success, to provide some helpful grammatical explanations. These explanations, shown if the student clicks the hint button, are optional. Since the explanations are not written for specific sentences but for the groups as a whole, we have to focus on general rules. This works well for areas like capitalization, but is harder for areas with many exceptions and irregularities like verb conjugation. Furthermore, the error groups $VB_VBZ and $VBZ_VB, because of overlap between the different GECToR transformations, quite often contain singular-plural errors instead of verb conjugation errors, which makes it again harder to provide grammatical explanations.


Exercise 7 of 15

Fix the following sentence directly in the input box:

Your answer: Electricity is very important in keeping our lives go.

Suggestion: Electricity is very important in keeping our lives go → going.

There is a verb form error in this sentence.

Some general rules:

The so-called present participle (talking) is used to indicate a duration or an ongoing activity (he was talking for 10 minutes). The present participle is formed by adding “ing” to the end of a verb.

The present participle cannot stand alone. The verb “to be” is often used as a helper (I am walking, they were running), but other verbs can also execute that function (we enjoy flying).

Figure 13: Training exercise. Displayed hint for the error group $VB__VBG. The *Give me a hint* button is only visible for the test and control groups, but not for the base group.

5.3 Survey application

We recruited 105 Turkers over the course of three days. Out of the submissions for the first language test, 69 were accepted and 36 were rejected. In our experience, participants can roughly be divided into two categories, namely Turkers interested in doing the task and Turkers interested in outsmarting the task. Submissions, therefore, should not simply be taken at face value but always reviewed, either by automatic testing or by manual inspection.

Over the course of the 10 training surveys, 12 additional Turkers were barred for participation rates below 50% and one additional Turker failed to complete the second language test. Out of the 13 Turkers that did not finish the survey, seven were from the test group, four from the control group, and two from the base group. Overall, 56 Turkers successfully completed the survey.

5.4 Evaluation

Q1: Do students perform better after having trained under the Leitner memory model as compared to a random exercise selection model?

On the second language test, the test group scores a mean of 27.88 points and the control group scores a mean of 29.79 points. However, a different number of people from the test group than from the control group completed the survey. In addition, mostly lower-scoring Turkers dropped out in the control group, while also some higher-scoring Turkers dropped out in the test group. For this analysis, we consider only Turkers t_i from the test group if the corresponding Turker c_i from the control group also completed the survey and vice-versa. This ensures that both groups have received the same number of invitations and that only Turkers of similar ranks are compared. 14 Turkers per group remain and both groups now have an identical mean of 29.43 points. Overall, no significant positive effect of the Leitner model is discernible ($p = 0.5$[ix]).

Q2: Do students perform better after having been provided with additional grammatical explanations during training as compared to solutions only?

On the second language test, the control group scores a mean of 29.79 points and the base group scores a mean of 28.52 points. We use the same approach as in Q1 and eliminate Turkers if their counterpart did not complete the survey. 18 Turkers per group remain. The control group has a new mean of 29.94 and the base group of 29.17. Overall, no significant positive effect of grammatical explanations is discernible ($p = 0.31$[x]).

[ix]Tailed T-test for independent samples.
[x]Tailed T-test for independent samples.

Q3: Do students, regardless of exercise selection model and grammatical explanations, perform better after having trained with the generated exercises?

On average, study participants that successfully completed the survey solved 55.04% of the exercises in the initial language test correctly. The performance varied across the different error groups:

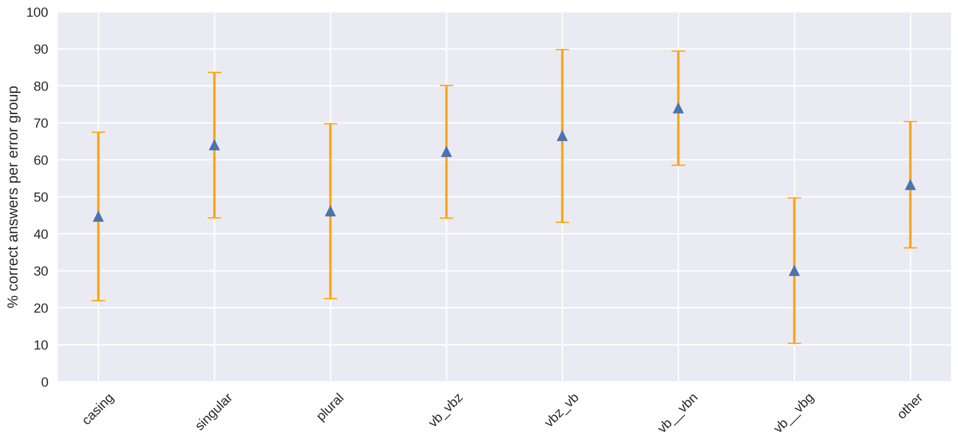


Figure 14: Average percentage of correct answers in the first language test per error group. The orange line segments display the standard deviation.

After training for 10 days, the study participants solved an average 71.92% of the exercises correctly in the second language test. While Turkers improved across all error groups, the differences in performance between the different error groups remained relatively stable.

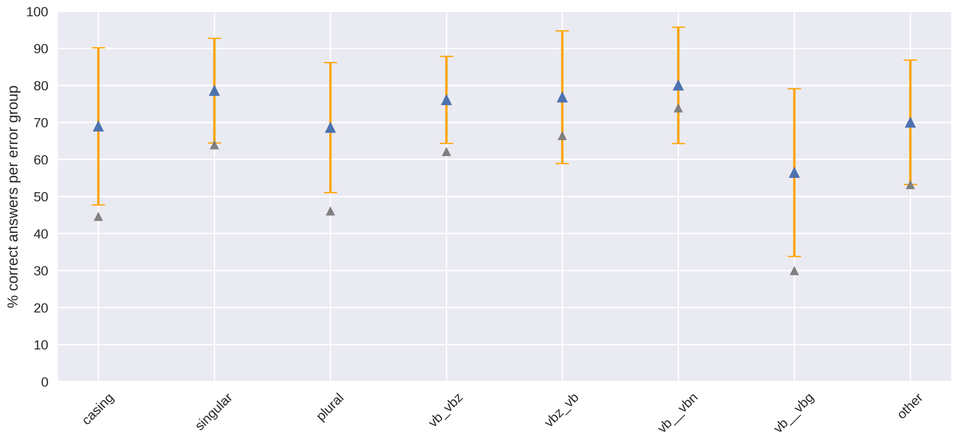


Figure 15: Average percentage of correct answers in the second language test are drawn as blue triangles. Grey triangles indicate the score in the first language test. The orange line segments display the standard deviation.

A clear improvement is visible across all error groups. Students have significantly improved their English grammar performance both overall ($p \leq 0.01$)[xi] and for each error group separately ($p \leq 0.05$)[xi].

[xi]Tailed T-test for dependent samples

Chapter 6

Discussion

We introduce a novel approach for automatically generating grammatical exercises based on GEC sequence taggers. Using the GECToR model, we generate such exercises for eight different error groups and manage to significantly improve student English grammar performance as a result of a 10-day Mechanical Turk training study. Given enough data and the right kind of tags, exercises could be generated for many more areas of English grammar.

We are unable to observe a clear influence of the Leitner memory model on student performance. A longer study duration with a larger number of error groups might yield different results.

We are also unable to observe a significant positive influence of grammatical explanations on student performance. The effect of grammatical explanations could become more noticeable after a few weeks without daily training HITs, as the explanations might help students retain their knowledge despite a lack of repetition. This could be investigated with a follow-up study. Additional research into the possibility of generating simple grammatical explanations tailored to the individual exercises might also be worth pursuing.

We show Mechanical Turk to be a valid option for language-related studies. The large pool of available Turkers and the intuitive Mechanical Turk interface make it possible to quickly run scientific studies. Still, a certain type of Mechanical Turk selection bias exists, since the typical Turker might not accurately represent the target demographic and Turkers decide themselves whether or not to accept a task. For that reason, it would be interesting to test the generated exercises in a more standard classroom setting as well.

References

[1] 2017, Emily Moeng, Distributional learning on Mechanical Turk and effects of attentional shifts, `https://core.ac.uk/download/pdf/230196063.pdf`

[2] 1885, Hermann Ebbinghaus, Über das Gedächtnis, `https://www.projekt-gutenberg.org/ebbingha/memory/gdaecht.html`

[3] 1972, Sebastian Leitner, So lernt man lernen. Der Weg zum Erfolg, Verlag Herder, Freiburg im Breisgau, Germany

[4] 1994, Robert Bjork, Memory and metamemory considerations in the training of human beings, `https://psycnet.apa.org/record/1994-97967-009`

[5] 2016, Burr Settles and Brendan Meeder, A Trainable Spaced Repetition Model for Language Learning, `https://www.aclweb.org/anthology/P16-1174.pdf`

[6] 1967, Paul Pimsleur, A Memory Schedule, `https://www.jstor.org/stable/321812`

[7] 2011, John Duchi, Elad Hazan, and Yoram Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, `https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`

[8] 2017, Christopher Bryant, Mariano Felice, and Ted Briscoe, Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction, `https://aclanthology.org/P17-1074.pdf`

[9] 2020, Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi, GECTor - Grammatical Error Correction: Tag, Not Rewrite, `https://github.com/grammarly/gector`

[10] Hugging Face, `https://huggingface.co/`

[11] 2015, Diederik P. Kingma and Jimmy Lei Ba, Adam: A Method For Stochastic Optimization, `https://arxiv.org/pdf/1412.6980v5.pdf`

[12] 1988, John W. Ratcliff and David Metzener, Pattern Matching: The Gestalt Approach, Dr. Dobb's Journal, Issue 46, July 1988

[13] 2019, Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla, Parallel Iterative Edit Models for Local Sequence Transduction, `https://arxiv.org/abs/1910.02893.pdf`

[14] 2021, Felix Stahlberg and Shankar Kumar, Synthetic Data Generation for Grammatical Error Correction with Tagged Corruption Models, `https://arxiv.org/pdf/2105.13318.pdf`

[15] 2014, Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant, The CoNLL-2014 Shared Task on Grammatical Error Correction, `https://www.comp.nus.edu.sg/~nlp/conll14st/CoNLLST01.pdf`

[16] 2019, Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe, The BEA-2019 Shared Task on Grammatical Error Correction, `https://aclanthology.org/W19-4406/`

[17] *Zirguezi*, `https://en.wikipedia.org/wiki/Leitner_system#/media/File:Leitner_system_alternative.svg`, retrieved 9th August 2021

[18] *Zirguezi*, `https://en.wikipedia.org/wiki/Leitner_system#/media/File:Leitner_system.svg`, retrieved 9th August 2021

[19] Building Educational Applications 2019 Shared Task: Grammatical Error Correction, `https://www.cl.cam.ac.uk/research/nl/bea2019st/`, retrieved 29th August 2021

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Automated English Grammar Learning

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Zweig	Yoel Elisha

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 6.9.21	Y. Zweig

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.