Universidad del País Vasco — Euskal Herriko Unibertsitatea

Error Generation for a Grammar Checker in Basque: Correction and Detection

Author: Ariane Méndez Amuchategui

Advisors: Maite Oronoz and Gorka Labaka

hap/lap

Hizkuntzaren Azterketa eta Prozesamendua
Language Analysis and Processing

Master's Thesis

Departments: Computer Systems and Languages, Computational Architectures and Technologies, Computational Science and Artificial Intelligence, Basque Language and Communication, Communications Engineer.

Laburpena

Lan honen xedea euskaraz idatzitako testuetan suertatu daitezkeen errore gramatikalak automatikoki antzeman eta zuzentzea da. Zehazki, ikasketa sakoneko teknikak erabiltzen dira, alde batetik esaldi erroredunetan aurkitzen diren erroreak detektatu eta haien mota gramatikala identifikatzeko eta, bestetik, esaldi erroredunen zuzenketa lortzeko. Ataza hauek automatikoki ebazteko gai diren eredu neuronalak entrenatu ahal izateko, etiketatutako datu-kopuru handiak behar dira. Lan honetan, detekzio eta zuzenketa ereduez gain, ataza hauetarako aproposak diren etiketatutako datuak sortzeko metodo automatiko bat ere aurkezten dugu. Metodo hau errore gramatikalak dituzten esaldiak sortzeko gai da, baita esaldi horiek sortutako errore motaren arabera etiketatzeko ere. Burututako esperimentuek agerian uzten dute entrenamendu garaian mota eta ezaugarri askotariko datuak erabiltzearen garrantzia. Gainera, gure datu sorkuntza metodoarekin sortutako esaldiak sistema neuronalen entrenamendurako erabiltzeak ereduak euskarazko gramatika-erregelei buruzko jakintzaz hornitzen dituela frogatzen da.

Hitz-gakoak: Zuzenketa Gramatikal Neuronala, Errore Gramatikalen Detekzio Neuronala, Datu Sintetikoen Sorkuntza, Baliabide gutxiko hizkuntzak, Transformer arkitektura

Abstract

The objective of this work is to automatically detect and correct grammar errors made in texts written in Basque. More specifically, Deep Learning techniques are used in order to, on the one hand, detect grammatical errors in sentences and specify their grammatical type and, on the other hand, correct the incorrect sentences. So as to train models that are capable of automatically solving these tasks, large amounts of annotated data are needed. In addition to the detection and correction systems, in this work we also develop a method for generating grammatically incorrect sentences that can be used as training data for the aforementioned neural models. This method can automatically generate realistic grammar errors, as well as tagging them according to the error type. Experiments highlight the importance of using diverse data samples for training. Furthermore, using data generated with our error generation approach proves to be effective in providing the systems with knowledge about grammar-rules in Basque.

Keywords: Neural Grammar Error Correction, Neural Grammar Error Detection, Synthetic Data Generation, Low-resource languages, Transformer architecture

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

According to the most recent sociolinguistic studies, 28.4% of people (above the age of 16) who live in *Euskal Herria* are competent in *Euskara* or Basque, while 16.4% of the population is considered to be passive bilingual, i.e., people who never or rarely speak in Basque but are capable of understanding it (Jaurlaritza et al., 2016). Regarding the actual use of the language, however, the same studies conclude that only 16.5% of the population uses Basque more than Spanish or in the same amount, meaning that there is a high number of people who know Basque but are not accustomed to using it. Consequently, it is probable that these people who do not tend to speak in Basque make certain errors when trying to use the language, mostly due to lack of habit. These errors may include grammar errors, using words in other languages and/or mixing them with words in Basque, misspelling issues in written texts or any other phenomenon that does not follow the rules of standardised Basque defined by *Euskaltzaindia*, the official academic language regulatory institution for Basque. Making such errors does not mean that communication is not longer possible, and even people who use Basque as their main language can make them. Nevertheless, depending on the degree and number of errors, the message that the communicator intends to give might be hard to understand in some cases.

Developing openly available tools that detect and correct these errors is important, not only for the actual correction, which makes the message clearer and the communication easier, but also because they can help users learn. Having doubts about how to say something during a conversation or while writing an essay is normal; having someone close by who is fluent enough to solve those doubts might not be that frequent. Error detection and correction systems would allow us to instantly solve our uncertainties regarding words or language-related structures and patterns, among others, and would be of assistance in our individual learning process. Some specific error types are already being handled by systems that are publicly available for everyone's use, such as the well known spelling checkers and correctors. Precisely, a spelling checker — Xuxen (Agirre et al., 1992) — has also been developed for the Basque community and it is freely available online for everyone to use.

1.2 Objectives, Contributions and Document Description

This work aims to keep developing tools that will be of aid to those who choose to use or learn Basque. To be more precise, our intention is to use the newest techniques in the field of Natural Language Processing (NLP) to build systems that focus on detecting and correcting grammatical errors that are made in texts written in Basque.

Research in the fields of Machine Learning and Deep Learning has drastically advanced in recent years and neural architectures are currently the state of art for several tasks, including NLP problems. One of those tasks is Grammatical Error Correction (GEC),

which can be seen as a particular case of Machine Translation, being the grammatically incorrect sentence the sentence in the source language and its correction the "translation".

If the GEC task wants to be solved by using neural networks, grammatically incorrect sentences and their corrections are needed in order to train a model. These corpora can be hard to obtain, because for most languages a big enough number of sentences has not been annotated with errors or corrected. In such cases, a typical approach is to create synthetic data by generating errors from correct sentences. The easiest way to do so is replacing, inserting, eliminating or re-ordering tokens from the original sentences. This method is simple and, since the token-operations are randomly applied, the probability of generating an incorrect sentence is high. However, the sentences created using this approach are very artificial and extremely different from real-life grammatical errors.

In this project, the first objective is to find a way to, starting from correct sentences written in Basque, generate errors that imitate those that are made in real scenarios. To do so, different types of errors that are commonly made in Basque will be analysed, an attempt to find which are the grammatical elements that get mixed in each case will be made and different methods to replicate those changes and create errors will be used. The quality of the generated corpus will be measured by using it, across different settings, to train a neural model and observing how the trained system performs in the aforementioned task of GEC.

The second objective of this work is to extend the error generation method in such a way that, in addition to creating realistic grammatically incorrect sentences, the part of the sentence that has been modified so as to generate said errors will be tagged according to the type of grammar-error that is created. This tagged corpus will allow us to train a neural model that aims to solve the task of Grammar Error Detection (GED), where, unlike GEC, the main intention is not to produce a correct version of an incorrect sentence, but to identify which part (if any) of a sentence is erroneous. GED can be seen as a particular stance of a Named Entity Recognition task, where instead of identifying named entities, the objective is to identify the type of grammar error.

The motivation behind having two separate systems instead of combining both correction and detection in a single one is the fact that each of the systems has a different use case. A correction system will always be used when the main objective is obtaining the output itself, that is, a new, improved version of what has been written. A detection system, however, will more likely be used for educational purposes. For instance, detection systems can be of aid for analysing the number and type of errors that are made in a text, which then can be used to determine the level of linguistic competence the person who has written it has. Having information about the exact error that is being made also makes it easier to find an explanation of why it is an error and how to fix it, which means that these systems can support people in their learning process of a language.

Overall, this work presents two main contributions to the field of Natural Language Processing in Basque. The first one is an advanced method for automatic grammar error

generation in Basque, which is defined by complex rules that have been designed after an elaborate analysis of grammar errors that are often made. The second contribution is the actual design and training of a Neural Grammar Error Detection system for Basque which, to our knowledge, has never been done before.

Regarding the first main contribution, it should be noted that, even though we believe our error generation method is solid, the task is extremely challenging and improvements still need to be made. We are aware that not all the rules we have defined work as expected in all cases, meaning that sometimes unrealistic errors or even correct sentences might be generated. Readers should be aware of this when following our reasoning behind the design of the error generation method and expect us to discuss the downsides of our proposal throughout the sections where the experiments that have been carried out are detailed.

The rest of this document is structured as follows. Chapter 2 gathers recent works related to tasks of Grammar Error Correction and Detection. In Chapter 3, we delve into the proposed synthetic error generation method by explaining the approach, the implementation, the challenges and the features of the generated corpora in great detail. Chapter 4 is dedicated to the Error Correction task: the followed approach and the chosen architecture are explained, performed experiments are listed and the obtained results are discussed. As for the Error Detection task, it is described in Chapter 5 and, similarly to the previous chapter, it includes the taken approach, implementation details, the experiments that have been carried out and the evaluation results. Finally, in Chapter 6 we discuss the main conclusions of our work and propose some future lines of work on the topic.

Additionally, we provide some useful information in the last pages of this document. Appendix A includes a list of comparisons of morphological analyses of correct and incorrect sentences. Appendix B shows all the results obtained for the error detection models developed in this work. Finally, we make translations of all the example-sentences written in Basque that are used throughout this document available in Appendix C, in case they serve for better understanding.

2 Related Work

In this section, firstly, literature on the topic of data generation for GEC will be studied. Then, we will talk about the actual tasks of GEC and GED and analyse what the newest approaches are. Finally, we will focus on the particular case of Basque and discuss the progress that has been made in the field up until this point.

2.1 Synthetic Data Generation for GEC

Deep Learning models are capable of obtaining state of the art results in numerous tasks, often times thanks to the huge amounts of annotated data that is used to train them. Collecting large quantities of annotated data, however, is not an easy thing to do, specially for certain tasks where very specific knowledge is needed for producing the appropriate annotations. Human-made annotations are usually desired because they are likely to be more precise than those generated by automatic methods, but annotating millions of samples by hand is extremely expensive and time-consuming. Therefore, when huge amounts on data are needed for training neural models, the usual approach consists in trying to create synthetic data that is as accurate as possible. Manually revised annotations are left for testing the models, because datasets used for testing contain much less samples than those used for training, making it a considerably more manageable task for humans.

In the case of GEC, the annotated data that is needed for training models has quite specific features: sentence pairs are needed, formed by a sentence that contains at least a grammar error and a sentence that is the correct version of the previous one. It is true that this kind of annotation can easily be found in most language examinations or essays written for language classes, where teachers usually cross out words or structures that are incorrectly used and write the appropriate correction beside the error that was made. However, the truth is that almost no resources that contain these characteristics can be found digitally, specially for low-resource languages. That being the case, recent GEC approaches have proposed their own methods for automatically generating data which could later be used for model training.

For example, one of the techniques that is proposed is back-translation, which is a data augmentation strategy commonly used in Machine Translation. It consists in 1) using the data that is initially available to train a model, 2) using said model to translate the samples in the target language to the source language, 3) adding the newly generated pairs to the initial data and 4) training a definite model using all the data. Ge et al. (2018) propose an approach based on this technique for the particular case of GEC. In their work, first, a model is trained with a limited amount of correct and incorrect sentence pairs and used to correct a subset of incorrect sentences. Then, the n-best corrections of those sentences are paired with the original correct sentence, as long as the fluency of the automatically corrected sentence is below the fluency of the original correct sentence. These new pairs are then incrementally added to the training corpus in subsequent training epochs. The procedure is proven to effective in improving the quality of the correction models.

Boyd (2018) proposes to use Wikipedia edits as a source of correct and incorrect sentence pairs. Article dumps and revision histories are downloaded and filtered to preserve only those edits that contain small changes from one to another. Then, since Wikipedia edits can be content related and not necessarily grammar error corrections, a gold-standard GEC corpus and the Wikipedia edits are compared using error annotation tools. Finally, all edits that are not considered to follow the patterns or be similar to the content found in the gold-standard corpus are discarded. At evaluation time, models trained with the remaining Wikipedia edits as additional data show better performance than those trained just with the initial gold-standard corpus.

Methods that use simple token operations have also been presented in recent years. Grundkiewicz et al. (2019) suggest the insertion, deletion and replacement of tokens in order to generate errors in a sentence but, instead of randomly choosing the tokens to modify and add in each case, which can make the sentence really unnatural and artificial, they propose the usage of confusion sets. For each sentence, a number of words is chosen to operate on, based on the word error rate in the training data. For each chosen word, the operation to perform (insertion, deletion or replacement) is randomly chosen with a given probability. If replacement is selected, a spellchecker is used to generate a list of suggestions for the chosen word. These suggestions are sorted by the edit distance between the original word and the proposed word with the new spelling, and the distance between their phonetic equivalents is also taken into account; only those suggestions that are within a relatively close distance from the input word are chosen to form a confusion set. The word in the original sentence is replaced with a word that is randomly chosen from this confusion set. GEC systems trained using this approach obtained the best results on the Restricted and Low Resource tracks of the Building Educational Applications (BEA) 2019 Shared Task (Bryant et al., 2019). Both of these tasks restricted the use of corpora so as to encourage development of systems that were not reliant on huge amounts of data.

The second best system on the Restricted and Low Resource tracks, developed by Choe et al. (2019), was trained using synthetic data generated with another token replacement technique. In this case, authors first use a sample of annotated training data to extract error patters, i.e., the edits made from the incorrect sentence to the correct one and their frequency. Then, for a new set of correct sentences, errors are generated by searching for tokens that appear in the edit pattern dictionary and applying edits in reverse. For tokens that are not present in the edit dictionary, errors are generated, given a certain probability, based on the part-of-speech of each word: nouns are replaced with their singular or plural counterparts, verbs are replaced with their morphological variants and prepositions are replaced with other prepositions.

The most current approaches aim for generating more realistic errors, i.e., errors that imitate patterns that are present in manually revised GEC datasets. For example, Stahlberg and Kumar (2021) propose to use error annotation tools to guide synthetic data generation. Similarly to the previous approaches, grammatically correct sentences are taken as the starting point and modified in such a way that incorrect sentences are

generated. What differentiates this approach from the previous ones is that a Transformer model is trained to perform this error generation process. To do so, first, error annotation tools are used for tagging parallel GEC corpora. Then, the tags predicted by the annotation tool for each sentence pair are appended to the correct sentence of the pair. Finally, a Transformer is trained using the parallel corpus that also contains the error tags. After training, the system is capable of, given a correct sentence and the type of error that wants to be created, generate and incorrect sentence that contains a grammar error of the specified type. Authors of the paper report state-of-the-art results when testing their GEC models with respect to the BEA-19 test sets.

2.2 Current approaches for GEC and GED

2.2.1 Grammar Error Correction

Initially, Grammar Error Correction was addressed using rule-based approaches (Naber et al., 2003; Oronoz, 2008). Then, classification based methods were proposed: individual classifiers were trained to tackle each specific type of grammar error, such as preposition or determiner misuses (Izumi et al., 2003; De Felice and Pulman, 2008). However, having a system for each type of error is not practical for real use cases, so research towards systems that were capable or handling multiple grammar error types began.

The first methods proposed with this in mind were based on Statistical Machine Translation (Yuan and Felice, 2013; Ehsan and Faili, 2013). These approaches use parallel data to train statistical models that are capable of learning probability distributions of the corpora and thus, have the ability to compute the most probable correction of a given input sentence.

More recently, neural architectures are being explored for solving the GEC task, due to the growing popularity of Deep Learning. Some approaches propose to combine statistical and neural models, obtaining corrections that preserve the accuracy that statistical models are able to achieve while also being fluent thanks to the neural component (Grundkiewicz et al., 2019). Alikaniotis and Raheja (2019) strongly encourage the use of Transformer-based language models to rank the possible corrections suggested by neural correction systems and conclude that, like for other Natural Language Processing tasks, the Transformer architecture provides a really strong baseline for GEC, because it achieves consistent high performance in the task. More complex approaches opt for combining sequence tagging models, token-level transformations, several stages of fine-tuning and ensembles of models like BERT, RoBERTa and XLNet and obtain almost state-of-the-art results by doing so (Omelianchuk et al., 2020).

The approach that is currently considered to obtain the best results for the GEC task is presented by Rothe et al. (2021) and it consists on fine-tuning mT5, a multilingual pre-trained text-to-text Transformer (Xue et al., 2020), for the particular case of grammar error correction. To do so, an unsupervised language-agnostic objective is described, which tries to imitate patterns of corrections that are normally found in manually labelled data for

GEC. By doing so and scaling the number of parameters from 60 million to 11 billion, state-of-the-art results are obtained for four languages: English, Czech, German and Russian.

2.2.2 Grammar Error Detection

Research on the field of Grammar Error Detection began much later than research on the field of Grammar Error Correction, precisely because the focus was set on correcting errors, and information about error types and frequency was just a byproduct of the output produced by correction systems. It was not until a value for education was seen in the actual detection and analysis of errors that investigation on the topic started.

Rei and Yannakoudakis (2016) present some of the first experiments using neural architectures for error detection. In their investigation, several sequence labelling architectures are trained for solving the task, including convolutional networks, bidirectional recurrent networks and bidirectional LSTMs (Long-Short Term Memories). Results demonstrate that detection models can identify more errors than systems that are solely focused on correcting errors.

Bell et al. (2019) propose integrating contextualised word embeddings in sequential methods for error detection. Contextualised representations of words can capture compositional information in language and, therefore, facilitate error identification. Authors of this work experiment with three types of word embeddings, BERT, ELMo and Flair, and integrate them in bidirectional LSTMs for error labelling. Results show that adding contextual information does in fact improve the performance of the error detection systems.

Some of the most recent works propose to dive deeper into the architecture of neural models and, instead of just using the output given by the final layer of a model, taking advantage of information provided by previous layers as well. This is the case of the approach presented by Kaneko and Komachi (2019), where the effects of employing information from intermediate layers of pre-trained language representation models is studied for the particular case of Grammar Error Detection. Said work proposes a multi-head multi-layer attention model that determines which are the appropriate layers in BERT to use for error detection. The model achieves state-of-the art results on three different grammatical error detection datasets.

2.3 Neural Grammar Error Correction in Basque

This section aims to describe the work proposed in the field of Grammar Error Correction for the specific case of Basque. We focus the section solely on the correction task because, to our knowledge, no neural systems for grammar error identification and tagging have been proposed as of writing this document. As for correction, the line of research is just beginning and very few works regarding the topic can be found. As a matter of fact, we are only aware of two works on the topic of Neural Grammar Error Correction for Basque: our own previous research and a new approach that proposes a more complex corpus generation method for the task. We talk about these two works in greater detail below.

2.3.1 Antecedents: Own Previous Research Work

In our previous research work (Méndez, 2020), an approach to build a Neural Grammar Error Corrector for Basque was studied. Said approach consisted in understanding Grammar Error Correction as a subtask of Machine Translation. State of the art Machine Translation techniques were analysed and the choice to use Transformers[1] for training our models was made. These models need to be fed big amounts of data in order to learn and generalise from the samples they are shown and achieve reasonably good results when asked to make a prediction about data samples they have never seen.

In the particular case of the Grammar Error Correction task, grammatically correct and incorrect sentence pairs are needed to train the models, so that they can learn which correction is adequate for each incorrect sample. To obtain a large enough amount of sentence pairs written in Basque that fulfill these characteristics, errors were generated from a corpora of grammatically correct texts written in Basque by applying simple token-operations, i.e. inserting, deleting, replacing and re-ordering. Then, these newly generated sentences were paired with the original sentence they were generated from, obtaining correct-incorrect sentence pairs that could be used to train the models. In the end, four different sets of sentence pairs were built, each of them formed by a different number of sentence pairs. This allowed us to compare the evolution and performance of the models the more data they received at training time.

Four GEC systems were trained making use of these four datasets and all of them were evaluated against automatically generated grammatically incorrect sentences (generated following the same procedure as the incorrect sentences in the train set) and real-life, human-made grammatically incorrect sentences. The obtained results were not very favourable: for synthetic incorrect sentences, the best system was able to correct around 42% of the errors and, for real incorrect sentences, the best system only managed to correct around 2% of the incorrect sentences. It was concluded that, since the created errors did not resemble real grammatical errors, when testing against real data, models were not able to generalise and correct most of the errors, because they had not been faced with similar examples in the training phase. However, the system trained with the largest dataset, which contained almost 10 million sentence pairs, proved to be the best one at identifying correct sentences and understanding that no corrections should be proposed for them, which demonstrated that feeding these type of models huge amounts of data containing diverse examples does help the learning process.

Taking our previous experience into account, the objective of this work is to improve the synthetic error generation method proposed in our previous research work and generate errors that are more similar to real, human-made errors. Then, we aim to use them in order to train new models and study whether they are useful for improving the performance of the aforementioned models. The scope of the previous work also intends to be extended

[1] Section 4.1 of the present work delves deeper into the Transformer architecture and some of its current applications.

by using these generated errors for training Grammar Error Detection models.

2.3.2 Neural Grammar Error Corrector presented by Elhuyar

The present work has as one of its main references a paper published by the Elhuyar Foundation (Beloki et al., 2020). To our knowledge, this is the only published work regarding Grammatical Error Correction in Basque where neural models are used and synthetic corpora with grammatical errors is generated. In this paper, following a similar approach to the one introduced by Yuan and Felice (2013), instead of using an annotated corpus as reference, grammatically correct sentences are the initial point and errors are generated from them using linguistic rules.

The implemented rules generate grammatically incorrect sentences by replacing specific words in the reference sentences. The words to be replaced are chosen according to grammatical information obtained using morphosyntactic analysers and depending on the error-type that wants to be generated in each case. Four different types of errors are tackled in this work: those related to verb tense, the ones associated to paradigms of auxiliary verbs, errors that are created when there is a lack of agreement between the verb and the subject and errors regarding completive sentences. These 4 error categories were selected from the extensive list of errors for Basque presented in Oronoz (2008) upon consulting a professional translator/corrector and a professional lexicographer and asking them to point out those they thought were the most common errors in texts written in Basque.

After automatically creating grammatically incorrect sentences, different datasets for training are built by combining correct-incorrect and correct-correct sentence pairs. Each of the datasets varies on different features, such as the amount of total sentence pairs used, the number of errors each incorrect sentence contains or the balance between different error-types applied to the whole dataset. Evaluation datasets are also constructed following a similar approach but, in addition to automatically generated sets, manually revised evaluation datasets are also provided.

A total of four GEC systems are presented, all of them based on the Transformer architecture and evaluated in terms of precision, recall and F0.5 score. The best results are obtained by a model trained with over 9 million sentence pairs. In particular, the incorrect sentences of the dataset employed for training said model are generated by simultaneously applying multiple error-generation rules to each original, correct sentence. After comparing this model to the rest, findings show that Transformers seem to work well for correcting sentences that contain multiple errors and perform worse in cases where a single error (or none) exists. The study also highlights the poor performance of models trained with sentences generated by surface token operations (i.e., insertion, deletion, replacement).

3 Building Corpora: Approach and Implementation

In order to build a neural grammar error corrector, a large corpus formed by correct and incorrect sentence pairs is needed, so that the models can learn where the errors lay in the incorrect sentences and how to fix them according to their correct counterparts. However, manually creating and revising millions of sentence pairs of such characteristics is extremely costly and hard work, hence only small non-synthetic datasets of grammatically incorrect sentences and their correct pairs can be found. Since large enough corpora that fulfils those requirements does not exist, specially for low-resource languages like Basque, we will use a corpus of correct Basque sentences as the starting point and we will define a method that allows us to modify them in such a way that (hopefully) realistic grammar errors will be created. That way, we will have a collection of correct and incorrect sentence pairs, which we will later be able to use for training GEC and GED systems.

As previously mentioned in Section 2, this error generation approach is inspired by Beloki et al. (2020), which is why we use the synthetic dataset provided in their paper[2] as the basis. Said data collection is built over a total of 4.927.748 different correct sentences written in Basque, which were extracted from the following sites: `Berria.eus`, `Argia.eus` and `Tokikom.eus`. A subset of 4.921.748 sentences was chosen to apply different automatic error generation strategies, consequently creating several datasets.

The publicly available corpus is formed by a training dataset and 6 sets of evaluation data. All the sets are collections of incorrect-correct sentence pairs and for each pair the rule(s) used for creating the incorrect sentence are specified. The evaluation data is divided in two main groups: data created fully automatically (identified as "Dea") and manually revised data (identified as "Dem"). For each group, 3 datasets are available: None, Single and Multi. None-type datasets contain original unmodified sentences with no errors, i.e. both sentences in the pair are identical and no rules have been applied. The other two types of evaluation datasets do contain gramatically incorrect sentences and their corrections: Single-type datasets are formed by sentences with a single grammatical error and their original versions and Multi-type datasets by sentences with multiple grammatical errors and their original versions. Table 1 summarises the number of sentences in each dataset.

Out of all those resources, the correct sentence of each pair in the training corpus and the 3 automatically generated evaluation datasets are employed in this work. The correct sentences are used as the starting point from which we will generate synthetic grammatical errors by applying our own rules; the evaluation datasets are used for testing our models and comparing results.

There are several reasons why we try to generate our own errors instead of directly using the ones in the training corpus published by Elhuyar:

1. This work is born from the need to improve the synthetic error generation method

[2]Synthetic training and evaluation datasets publicly available here: `https://hizkuntzateknologiak.elhuyar.eus/assets/files/elh-gec-eu.tgz`

	Number of sentence pairs		
	Correct-Incorrect	Correct-Correct	Total
Train	4.41M	4.92M	9.333.672
Dea_Single	2.000	0	2.000
Dea_Multi	2.000	0	2.000
Dea_None	0	2.000	2.000
Dem_Single	250	0	250
Dem_Multi	221	0	221
Dem_None	0	201	201

Table 1: Number of sentence pairs for each dataset in the publicly available corpus developed by the Elhuyar Fundation.

proposed in our previous research work. We had already studied various approaches and it was clear that this particular task was the next step towards obtaining better results. Directly using the incorrect sentences provided by Elhuyar would not give us the chance to explore options for implementing this error generation method, and thus, there would no longer be a clear motivation behind this work.

2. Generating new synthetic sentences provides new resources to the investigation of GEC for Basque. It gives us the possibility of combining our synthetic examples with the ones created by Elhuyar and training new models with larger datasets.

3. By analysing some of the provided synthetic sentences, we detected some cases where the generated word does not exist. For instance, starting from the correct sentence *"Joxan Goikoetxea ariko da Lasarte-Oriako Akordeoi Jaialdian"*, the sentence *"*Joxan Goikoetxea artzen da Lasarte-Oriako Akordeoi Jaialdian"* is proposed as its incorrect version. The sentence is in fact incorrect, but because the word *"artzen"* does not exist. The word *"aritzen"* should have been generated in its place, simulating a typical case of verb tense misuse. Small mistakes like this one are normal because, after all, the error generation method proposed in the paper is automatic and it is extremely unlikely that all the generated samples will be perfect. Still, by defining our own rules for error generation, we might be able to produce appropriate samples in cases where their method fails (and vice versa).

The rest of this chapter elaborates on the approach taken for creating our own incorrect-correct sentence pairs given the correct sentences of the Elhuyar training corpus. As for the evaluation datasets, their use is further discussed in Section 4.3.2.

3.1 Error analysis

The first step for implementing our own error generation method is to analyse the errors described in Beloki et al. (2020) and see how many can be replicated. Four groups of errors have been studied: related to verb tense, concordance of the verb and the subject, completive sentences and auxiliary verb paradigms.

To determine what type of change needs to be made in a sentence in order to create each of those errors, we can take incorrect-correct sentence pairs in which we know that those errors occur and compare their morphological analyses. To do so, we employ Eustagger, a morphological analyser and Part-of-Speech tagger designed for Basque (Ezeiza et al., 1998). This analyser tries to disambiguate all the words in a sentence and then tags them with the grammatical information that corresponds to each of them. An example of the output obtained with this linguistic analyser can be seen in Figure 1. Servers of the IXA Research Group have been used in order to execute the analyser.

In our particular scenario, for each type of error, we have used Eustagger to obtain morphological analyses of the sentence pairs proposed in the reference paper. These sentences are presented as examples of incorrect sentences that contain each of those errors and their correct versions. Then, we have compared those analyses and identified the tags that differ between the two. Those tags are representative of the errors and, consequently, can be used as patterns to detect in which words of a sentence an error can be created.

For instance, let us take the correct sentence *"Ziur bihar jakingo dugula"* and the incorrect one that is often used in its place: *"*Ziur bihar jakiten dugula"*. This is an example of a sentence where the verb tense is not properly used. Figure 2 shows the analysis given by Eustagger for each of those sentences (the difference between both analyses is highlighted in red). As can be seen, they only differ in one of the tags of the third word, which changes from `GERO` in the correct example to `EZBU` in the incorrect one. This indicates that the verb in the correct sentence is in future tense (*etorkizuna/geroaldia*) and the verb in the incorrect sentence is not finished (*ez burutua*). Therefore, any time the `GERO` tag is found in a sentence, a verb tense related error can be generated if said tag is modified and replaced by `EZBU`.

```
/<Nire>/<HAS_MAI>/
        ("ni" IOR PERARR NI GEN NUMS MUGM w109,L-A-IOR-PERARR-3,lsfi122 @IZLG>)
/<ustez>/
        ("uste" IZE ARR BIZ- INS MG w110,L-A-IZE-ARR-125,lsfi123 @ADLG)
/<,>/<PUNT_KOMA>/
/<hori>/
        ("hori" DET ERKARR ABS NUMS MUGM w112,L-A-DET-ERKARR-2,lsfi124 @SUBJ)
/<horrela>/
        ("horrela" ADB ARR w113,L-A-ADB-ARR-17,lsfi125 @ADLG)
/<da>/
        ("izan" ADT A1 NR_HURA PNT w114,L-A-ADT-80,lsfi126 @+JADNAG)
/<.>/<PUNT_PUNT>/
```

Figure 1: Analysis of the correct sentence *"Nire ustez, hori horrela da"* obtained using Eustagger.

Figure 2: Differences between the analyses of the sentences *"Ziur bihar jakingo dugula"* and *"*Ziur bihar jakiten dugula"*.

However, not all cases are that simple. If we compare the sentences *"Gauza bat falta zait esateko"* and *"*Gauza bat faltatzen zait esateko"*, which were proposed in the original paper as an example of perfective/imperfective errors, we realise that Eustagger does not disambiguate the word *"falta"* as a verb. Instead, it identifies it as a noun (see Figure 3) and, thus, it is not possible to compare both analyses as a means to detect which tags should be used to identify errors related to finished vs. non-finished actions. The approach in such cases has consisted in trying to find another word that behaves similarly to the one not properly disambiguated, in hopes that given a different sentence and context Eustagger will be able to show the desired properties. To illustrate, in the case of the previous example, instead of the verb *"falta"*, the verb *"bururatu"* has been used. A correct sentence that uses it has been built, such as *"Gaur gauza bat bururatu zait"*, as well as its incorrect form, *"*Gaur gauza bat bururatzen zait"*, and both have been analysed (see Figure 4). In this case, the disambiguation does allow to find the tags needed to be changed for error generation.

The complete list of comparisons performed can be seen in Appendix A, where given several pairs of sentences (being the first sentence the correct one and the second one the grammatically incorrect one) their analyses are shown and the difference between both analyses is marked.

Once these comparisons between analyses of sentence pairs that serve as example of all the error types listed in the reference paper have been made, we can build a mapping table that matches each error type with the pattern that needs to be found in the analysis of a sentence so that said error can be applied to it. Our mapping table is provided in

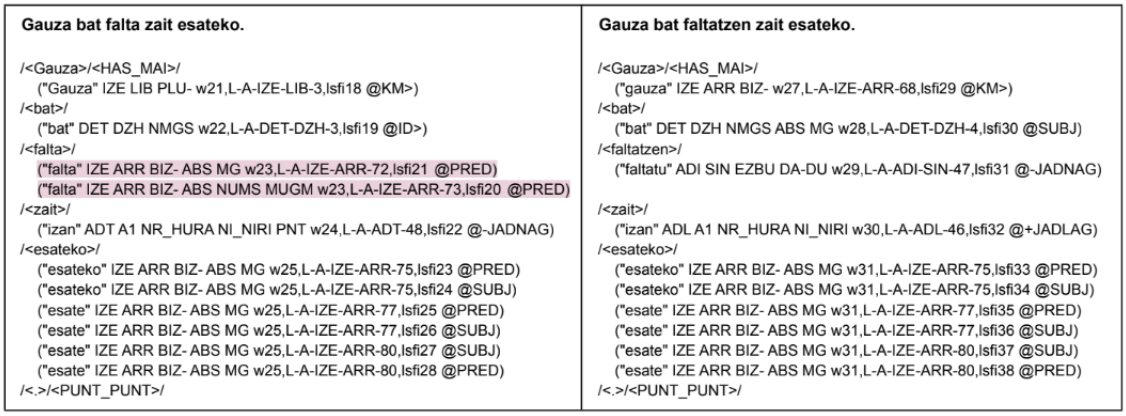

Figure 3: Differences between the analyses of the sentences *"Gauza bat falta zait esateko"* and *"*Gauza bat faltatzen zait esateko"*. The disambiguation issue is underlined.

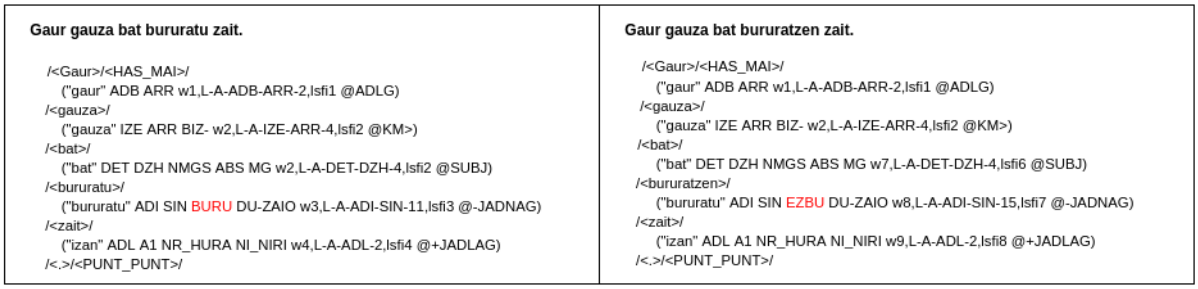

Figure 4: Differences between the analyses of the sentences *"Gaur gauza bat bururatu zait"* and *"*Gaur gauza bat bururatzen zait"*. The verb is properly disambiguated.

Table 2. The first column indicates the type of error and the second column indicates the pattern (as a regular expression) that needs to appear in the analysis of a word for it to be modified.

A third column named "FST" exists in our mapping table. FST stands for finite-state transducer, which is a type of finite-state machine (FSM). FSMs are mathematical models of computation defined by a list of states, an initial state and the inputs that trigger transitions from one state to another. The peculiarity of FSTs is that they produce output as well as reading input. To be more precise, in the case of FSMs, the defined transitions are followed, depending on the input, to jump between states and arrive at a certain final state, which will give a certain output. Using FSTs, this behaviour is extended by labelling the transitions with an input/output pair, in such a way that output will be produced every time a transition between two states is made. This means that, while FSMs can only be used for pattern-matching, FSTs can also be used for parsing.

Two FSTs derived from EDBL[3] (Aldezabal et al., 2006) have been used in this work:

[3]EDBL stands for *Euskararen Datu-Base Lexikala*, which translates to "Lexical Database of Basque".

Error	Pattern	FST
ko/go_ten/tzen	ADI SIN GERO	morpheme
nuke_nuen	ADL B2	analysis
nuen_nuke	ADL B1	analysis
NR-NK_NR-NI-NK	(edun—ukan—izan)(.)*NR_[^\s]+\sNK_[^\s]+\s	analysis
erg_abs	ERG(.)*@SUBJ	analysis
denik-dena_dela	ADL(.)*KONPL	morpheme
NR-NI_NR-NI-NK	(edun—ukan—izan)(.)*NR_[^\s]+\sNI_[^\s]+\s[^NK]	analysis
NR-NK_NR	(edun—ukan—izan)(.)*NR_[^\s]+\sNK_[^\s]+\s	analysis
NR-NI_NR-NK	(edun—ukan—izan)(.)*NR_[^\s]+\sNI_[^\s]+\s[^NK]	analysis
buru_ezbu	BURU	analysis
NR-NK_NR-NI	(edun—ukan—izan)(.)*NR_[^\s]+\sNK_[^\s]+\s	analysis
NR-NI-NK_NR-NI	(edun—ukan—izan)(.)*NR_[^\s]+\sNI_[^\s]+\sNK_[^\s]+\s	analysis
NR_NR-NK	(edun—ukan—izan)(.)*NR_[^\s]+\s[^NI_(.)*NK_]	analysis

Table 2: List of errors, patterns that need to be found in the analysis to apply them and which transducer to use in each case.

one that contains complete analysis and morphological information about the words, from now on referred to as "analysisFST", and one that only contains information about morphemes — roots and affixes —, from now on referred to as "morphemeFST". By way of illustration, the different outputs obtained when using the word *"etxean"* as input for these two transducers are shown in Figure 5.

Taking into account that FSTs are bi-directional, we can use these transducers, not only to get the analyses of words (as demonstrated in Figure 5), but also to generate words given a certain analysis. This means that, every time we detect that an error can be generated in a certain word, we can pass the word to a transducer, get its analysis, modify the appropriate tags and return the modified analysis to the transducer, which will generate a new word. For all cases, the analysis of the input words has been done in the upward direction (i.e. using the word as input to the transducer) and the generation in the downward direction (i.e. using the analysis as input to the transducer).

Depending on the error that wants to be generated in each particular case, one trans-

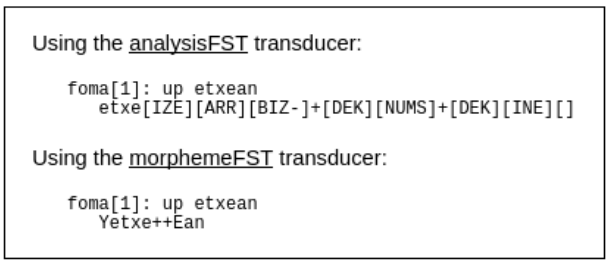

Figure 5: Differences between the outputs given by the two FSTs for the word *"etxean"*.

ducer or the other has been employed, which is what the third column in our mapping table is for: error types that can be created by just changing the suffixes of a word have the value "morpheme" in the "FST" column, which means that the error will be generated using the morphemeFST transducer; for the rest of error types, the analysisFST transducer will be used, as indicated by the "analysis" value in the third column of the mapping table.

In order to create the analysis that will be given as input to the transducer in the generation phase, simple rules have been written and built into FSTs using Foma, a finite-state toolkit for constructing finite-state automata and transducers (Hulden, 2009). The rules that have been defined modify some of the analyses obtained with the transducers so that, given a certain context, tags in the initial analysis are substituted by other tags. The following subsection details these rules, for which type of error each of them is used and the changes they produce in the analyses obtained by the transducers.

3.2 Error generation

3.2.1 Procedure

Once we have examined all the errors we want to automatically create and detected the patterns that indicate from which words those errors can be generated, the next step is to actually build the incorrect sentences. For that task, first, we have iterated over the list of correct Basque sentences, normalised them by lowercasing and tokenizing them and analysed all of them with Eustagger. Then, using our mapping table, we have identified all the words in those sentences whose morphological analysis contains a pattern that corresponds to an error. Next, we have passed those words to the corresponding transducer, modified the obtained analyses - in different ways depending on the error that wants to be generated from each word - and passed (in the opposite direction) the new analyses to the same transducer. Finally, in the correct sentences, we have replaced the original words with the outputs given by the transducer. By following this procedure, we have produced modified versions of the original sentences, in which the newly added words make the sentence grammatically incorrect but are not too far off from the original word and the original meaning of the sentence is mostly maintained.

Two versions of this method have been implemented: one where a single error has been created per sentence and one where multiple errors have been added to a sentence. In the first case, if a sentence contains 5 words that can be modified so as to create errors, 5 sentence pairs are generated, each one containing one of those 5 errors (see Table 3). In the second case, all the errors that can be created from the correct sentence are applied in a single incorrect sentence (see Table 4).

In practice, because of time limitations, we have decided to only use those incorrect sentences created with the first method, that is, those that have a single error in them. It is also worth mentioning that, even if several different incorrect sentences can be created from the same initial sentence by using this method, we have decided to randomly select one of the pairs and discard the rest. The reason behind this is that we want to avoid

Correct sentence	Generated sentence (with errors)
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritza gaur hasiko du urte politikoa , donostian .
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritzak gaur hasten du urte politikoa , donostian .
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritzak gaur hasiko dio urte politikoa , donostian .
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritzak gaur hasiko da urte politikoa , donostian .
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritzak gaur hasiko zaio urte politikoa , donostian .

Table 3: Example of multiple incorrect sentences that can be generated from a single correct sentence if only one error per sentence is applied.

Correct sentence	Generated sentence (with errors)
jaurlaritzak gaur hasiko du urte politikoa , donostian .	*jaurlaritza gaur hasten zaio urte politikoa , donostian .

Table 4: Example of an incorrect sentence that can be generated from a correct sentence by applying multiple errors to the sentence.

having a single correct sentence (and several incorrect versions of it that have very little variance) appear multiple times in our training corpus, because the system may memorise the sentence itself instead of trying to learn correct and incorrect grammar patterns.

Having described the general steps of our error generation approach, let us explain in detail the process followed for creating each type of error:

3.2.1.1 Errors related to Verb Tense

This section describes the proposed error generation methods for cases where the verb tense or aspect is inappropriately used. Mainly, the following scenarios are studied: mixing the present and the future tense, mistaking the past tense and the conditional and confusing the finished aspect with the unfinished aspect.

3.2.1.1.1 ko/go → ten/tzen

The first error that has been examined is regarding the case where two aspects of verbs get mixed: actions that have not been finished and actions in the future. This is the case of sentences such as *"Ziur bihar jakingo dugula"*, where, instead of using the future tense, a common mistake is to use the unfinished aspect and say *"*Ziur bihar jakiten dugula"*. The proposed method to automatically generate this type of error is simple:

First, among all the sentences, words that are simple verbs in the future tense are found, that is, those that end with the morphemes "-ko" and "-go". This is done by looking for the `GERO` tag[4], as mentioned in Section 3.1, throughout the analyses of all

[4]In practice, we look for the tags `ADI` and `SIN` along with the tag `GERO`, which correspond to verb (*aditza*) and simple, respectively. This is done to make sure we are working with a simple, not a composed or periphrastic verb, in the future tense.

```
foma[1]: up jakingo
   XjakiN++ko
foma[1]: down XjakiN++ten
   jakiten
   9jakiten
foma[1]: down XjakiN++tzen
   ???
```

Figure 6: First command is used to obtain the analysis of the word *"jakingo"*. Second and third commands show both possible morphemes added to the base form and how only one of the constructions returns the word *"jakiten"*.

the sentences in the corpus. Then, in an effort to get their base forms, every word that has those features is passed as input to the transducer that only gives information about morphemes. Since verbs that have not yet been finished can either end with the morphemes "-ten" or "-tzen", the two possible forms are created by concatenating the base form to each of these morphemes. After that, the two forms are passed to the transducer in the opposite direction; the form that exists will give a verb in Basque and the other form will return an empty output. Finally, in the correct sentence, the newly generated verb is added in place of the original verb.

Let us follow the previous example and suppose that *"Ziur bihar jakingo dugula"* is the input sentence. After analysing it with Eustagger, the system will detect, based on the tags of the analysis, that *"jakingo"* is a verb in the future tense. Then, this word will be passed as input to the corresponding transducer and the output will be `XjakiN++ko`. In this case a single output will be obtained, but other words can have several outputs, so we need to make sure we choose the right one by ensuring it starts with the "X" character, which specifies that said word is a verb. After that, the base form — `XjakinN` — will be extracted from the analysis and two strings will be created, one for each of the possible morphemes the unfinished aspect can have: `XjakiN++ten` and `XjakiN++tzen`. Finally, these two strings will be passed to the transducer in the opposite direction; `XjakiN++ten` will return the word *"jakiten"* and `XjakiN++tzen` will return and empty output (because it does not exist). The operations done using the transducer are shown in Figure 6. Finally, the word *"jakingo"* will be substituted by *"jakiten"* in the original sentence, thus creating the incorrect sentence *"*Ziur bihar jakiten dugula"*.

3.2.1.1.2 nuen → nuke / nuke → nuen

The next error-type that has been approached is the one where the past tense (indicative) and the conditional (indicative) get mixed. An example of this error is using the incorrect sentence *"*Gustura egingo nuen"* instead of *"Gustura egingo nuke"*. This error can be generated bidirectionally, that is, taking words that use the past tense and changing

them to conditional or taking conditional words and changing them to past tense. The process is the following:

First, words in past tense or conditional form (both indicative) are found. Then, every word that has those features is passed as input to the transducer and their morphological information is retrieved. Once we have the analysis, the tag that corresponds to the past tense or conditional is changed and passed to the transducer in the opposite direction. Finally, in the correct sentence, the original word is replaced by the newly generated one.

If the initial sentence is *"Gustura egingo nuke"*, for example, the word *"nuke"* is passed to the corresponding transducer and the following output is obtained: `edun [ADL] [B2] [NR_HURA] [NK_NIK]` and `ukan [ADT] [B2] [NR_HURA] [NK_NIK] [PNT]`. The tag to be changed is the second one: "B2" is replaced with "B1" and vice versa. Note that the `B1` tag stands for past indicative and the `B2` tag for conditional indicative. After changing the tag[5], the analysis is passed to the transducer in the opposite direction, resulting in the word *"nuen"*.

3.2.1.1.3 Perfective (burutua) → Imperfective (ez burutua)

The last error related to verb tense that has been analysed is the one where verbs that should be used to talk about unfinished actions are used when verbs that correspond to finished actions should be. For instance, instead of the sentence *"Zerbait falta zait esateko"*, a common mistake is to use the sentence *"*Zerbait faltatzen zait esateko"*.

This error is specially complex, because changing the aspect of a verb from finished to unfinished does not always generate an error. For example, although the only thing that changes is the aspect of the verb (as in the example just mentioned), both the sentences *"Afaltzera gonbidatu ninduen"* and *"Afaltzera gonbidatzen ninduen"* are grammatically correct. This occurs because while some verbs take the morpheme *"-tzen"*, others do not. In order to simplify, we will suppose that an error is always generated if a sentence contains a verb whose aspect is finished and it is changed to unfinished.

To create this type of error, words whose aspect is the finished one have been identified and passed to the transducer that outputs the morphological analysis. In this case, the transducer returns several possible analyses; we need to keep the one that informs about the finished aspect (the one that contains the tag `BURU`). For example, in the case of the word *"gonbidatu"*, the outputs given by the transducer are `gonbidatu [ADI] [SIN] + [AMM] [PART]`, `gonbidatu [ADI] [SIN] + [AMM] [PART] + [DEK] [ABS] [MG] []` and `gonbidatu [ADI] [SIN] + [AMM] [PART] + [ASP] [BURU]`, so the last one will be used. Once we have this analysis, we only need to change the tag `BURU` for the tag `EZBU`, which corresponds to the unfinished aspect, and the tag `PART` for the tag `ADOIN`, which correspond to participle and base of verb respectively, and pass this transformed analysis to the transducer in the opposite direction. The generated word will be *"gonbidatzen"*.

[5]We can perform the replacement in either of the outputs, because both will give us the same result.

3.2.1.2 Errors related to Auxiliary Verb and Subject Agreement

This section illustrates the error generation method in the scenario where there is an absence of concordance between the verb and the subject. Often times, this is due to confusion when declining the suffix of the subject.

3.2.1.2.1 Ergative subject → Absolutive subject

A common example where a lack of agreement between the verb and the subject can be found is the case of the subject getting mixed and using the absolutive case instead of the ergative case. This occurs, for example, when *"*Ni ez dut nahi"* is used instead of *"Nik ez dut nahi"*.

The overall procedure to generate this kind of error is the same we used for errors 3.2.1.1.2 and 3.2.1.1.3. The only thing that changes is that, in this case, when the morphological analysis from the transducer is received, we need to ensure that, out of all the possible analyses, the one that contains the tag that marks the ergative case — `ERG` — is chosen. This is important because, in cases such as the word *"langileak"*, the word can be ergative case (*"langileak egin du"*, a single worker has done it) or absolutive case (*"langileak jatorrak dira"*, the workers (plural) are nice), and we only want to change the ergative ones.

3.2.1.3 Errors related to Completive sentences

This section defines the approach for generating cases of verb suffix misuse, specifically those in completive sentences, where it is usual to get the suffixes *"-(e)la"*, *"-(e)nik"* and *"-(e)na"* mixed up.

3.2.1.3.1 denik/dena → dela

In this scenario, cases where the words *"denik"* and *"dena"* get mixed with the word *"dela"* have been studied. For example, the incorrect sentence *"*Ez dut uste hori egia dela"* is often used instead of *"Ez dut uste hori egia denik"* or *"*Badago beste aukera bat hobea dela"*, which is also incorrect, is said instead of *"Badago beste aukera bat hobea dena"*.

The words *"denik"*, *"dena"* and *"dela"* are all auxiliary verbs derived from the form *"da"* and completives. In order to simplify, the approach in this case has been to find all words that have those two features and follow a similar process to the one used in error 3.2.1.1.1: to change the morphemes at the end of the words.

For instance, if the input sentence is *"Ez dut uste hori egia denik"*, the word *"denik"* will be passed to the corresponding transducer and two outputs will be given: `d@+En+Rik`

and `d@+Enik`. Then, the base form `d@` will be extracted from either of the outputs and concatenated with the morpheme "-ela", obtaining the string `d@+Ela`. Passing this string to the transducer in the opposite direction will result in the word *"dela"*, which is the one used in erroneous cases.

For this particular group of errors, it is important to mention two things:

1. We have defined that, in order to create errors of this category, we need to find words whose analyses contain the tags `ADL` and `KONPL`. However, the analysis of the word *"dela"* itself also contains those two tags. This means that, whenever an input sentence has the word *"dela"* in it, it will be presumed that an error of this type can be generated. In reality no error will be created when that occurs, because, for errors of this type, we always assume that the error is generated when the word *"dela"* is used. In other words, the generated sentences will still have the word *"dela"* in them, so we will have generated a sentence that is identical to the reference sentence. To determine which words that are auxiliary verbs and completives can generate errors and which should not be modified, further analysis needs to be done.
2. In all the examples we have tried, the word *"dena"* has been disambiguated as an adverb that means "all" or "everything", so we have not been able to test our error generation method for sentences like *"Badago beste aukera bat hobea dena"*.

3.2.1.4 Errors related to Auxiliary Verb Paradigms

The final section in the error generation step considers errors related to auxiliary verb paradigms. Four auxiliary verb paradigms — NOR, NOR-NORK, NOR-NORI and NOR-NORI-NORK — exist in Basque and it is very common to use them inappropriately. To be precise, methods for generating the following types of verbal paradigm changes have been implemented: NOR-NORK to NOR-NORI-NORK, NOR-NORK to NOR, NOR-NORI to NOR-NORI-NORK and NOR-NORI to NOR-NORK. Some of these changes have been implemented in both directions.

3.2.1.4.1 Nor-Nork → Nor-Nori-Nork

"Atzo ikusi zintudan" is an example of a sentence where the paradigm of the auxiliary verb is NOR-NORK (NOR: *"zu"*, NORK: *"nik"*). A typical error is to use *"nizun"* instead of *"zintudan"*, that is, to use NOR-NORI-NORK (NOR: *"hura"*, NORI: *"zuri"*, NORK: *"nik"*).

To generate errors of this kind, all NOR-NORK elements have been detected and changed to NOR-NORI-NORK. After examining several examples where this type of change is made, we have realised that, when adding the NORI element, the common way to do it is to use the person and number that where previously used in the NOR element in the NORI element, in this case 2nd person singular (the NOR element of *"zintudan"* is *"zu"* and the

```
#1   def NorNoriBukRI [..] -> {RI} || {NR_} [[H|N] {I} | [G|Z] {U}] _ %] ;
#2   def NorNoriBukI K -> I || {NR_} [? - {K}]* _ %] ;
#3   def NorNori3PS {HURA} -> {HARI} || {NR_} _ ;
#4   def AldatuNorNori {NR} -> {NI} || _ %_ ;
#5   def GehituNorHura [..] -> {[NR_HURA]} || _ {[NI_} ;
```

Figure 7: Foma rules for modifying the verbal paradigm from NOR-NORK to NOR-NORI-NORK.

NORI element of *"nizun"* is *"zuri"*), and to add the 3rd person singular as the NOR element (the NOR element of *"nizun"* is *"hura"*).

To perform this change, the morphological tags of the word have been obtained using the corresponding transducer and then, using Foma, several rules have been created to change the string of the morphological analysis appropriately. For example, if the analysis of the word *"zintudan"* is `edun [ADL] [B1] [NR_ZU] [NK_NIK]`, using the defined rules it is changed to `edun [ADL] [B1] [NR_HURA] [NI_ZURI] [NK_NIK]` (the analysis that corresponds to the word *"nizun"*). Once this change has been made, the newly generated analysis has been used as the input of the transducer to obtain the desired word.

As for the specific Foma rules designed to modify the analysis, we present Figure 7 as means of illustration. Although these rules are only used for changing the auxiliary verb paradigm from NOR-NORK to NOR-NORI-NORK, all the scenarios related to auxiliary verbal paradigm that will be later introduced in this document work with a similar set of rules. In this particular case, rules #1 to #4 are used for substituting the NOR element with the NORI element while maintaining the person and number, and rule #5 adds the 3rd person singular as the new NOR element. The exact modifications performed by each of these rules are listed below:

rule #1: adds the *"-ri"* suffix if *""hi"*, *"ni"*, *"gu"* or *"zu"* are found in the NOR tag, obtaining the tags `NR_HIRI`, `NR_NIRI`, `NR_GURI` and `NR_ZURI`[6].

rule #2: adds the *"-i"* suffix in place of the *"-k"* suffix if *"zuek"* or *"haiek"* are found in the NOR tag, obtaining the tags `NR_ZUEI` and `NR_HAIEI`[6].

rule #3: replaces *"hura"* by *"hari"* if *"hura"* is found in the NOR element, obtaining the tag `NR_HARI`[6].

rule #4: changes all the tags that reference the NOR element so that they reference the NORI element, obtaining the tags `NI_HIRI`, `NI_NIRI`, `NI_GURI`, `NI_ZURI`, `NI_ZUEI`, `NI_HAIEI` and `NI_HARI`.

[6] The tags created in this step are not correct: they reference the NOR element but contain values that correspond to the NORI element. These are not final tags and rule #4 is designed specifically to fix this issue and create the definite ones.

```
foma[1]: up zintudan
   edun[ADL][B1][NR_ZU][NK_NIK]
   edun[ADL][B1][NR_ZU][NK_NIK]+n[ERL][ZHG][]
   edun[ADL][B1][NR_ZU][NK_NIK]+n[ERL][ERLT][]
   edun[ADL][B1][NR_ZU][NK_NIK]+n[ERL][MOS][]
   edun[ADL][B1][NR_ZU][NK_NIK][PNT]
   edun[ADL][B1][NR_ZU][NK_NIK][PNT]+n[ERL][ZHG][]
   edun[ADL][B1][NR_ZU][NK_NIK][PNT]+n[ERL][ERLT][]
   edun[ADL][B1][NR_ZU][NK_NIK][PNT]+n[ERL][MOS][]
foma[1]: down edun[ADL][B1][NR_ZU][NK_NIK]
   zintudan
   9zintudan
foma[1]: down edun[ADL][B1][NR_HURA][NI_ZURI][NK_NIK]
   nizun
   9nizun
foma[1]: down edun[ADL][B1][NR_HURA][NI_ZURI][NK_NIK]+n[ERL][ZHG][]
   nizun
   9nizun
foma[1]: down edun[ADL][B1][NR_HURA][NI_ZURI][NK_NIK]+n[ERL][ERLT][]
   nizun
   9nizun
foma[1]: down edun[ADL][B1][NR_HURA][NI_ZURI][NK_NIK]+n[ERL][MOS][]
   nizun
   9nizun
foma[1]: down ukan[ADT][B1][NR_HURA][NI_ZURI][NK_NIK][PNT]
   nizun
   9nizun
foma[1]: down ukan[ADT][B1][NR_HURA][NI_ZURI][NK_NIK][PNT]+n[ERL][ZHG][]
   nizun
   9nizun
foma[1]: down ukan[ADT][B1][NR_HURA][NI_ZURI][NK_NIK][PNT]+n[ERL][ERLT][]
   nizun
   9nizun
foma[1]: down ukan[ADT][B1][NR_HURA][NI_ZURI][NK_NIK][PNT]+n[ERL][MOS][]
   nizun
   9nizun
```

Figure 8: All possible morphological analyses of the word *"zintudan"* and how they all can be used to generate the word *"nizun"*.

rule #5: adds the 3rd person singular of the NOR element — `NR_HURA` — as a new tag and places it before the tag that corresponds to the NORI element.

An important note to be made is that the morphological analysis of the input word may return many outputs, some of which even differ on the base form of the verb. For errors of this particular case, however, it is not a problem because the desired final word can be obtained by modifying any of the analysis outputs, as seen in Figure 8.

3.2.1.4.2 Nor-Nork → Nor

Sometimes elements are removed from the verbal paradigm instead of being added. This occurs, for instance, when *"*Azkenaldian asko argaldu da"* is used instead of *"Azkenaldian asko argaldu du"* or, in other words, when NOR is employed where NOR-NORK is supposed to be used.

The Foma rules implemented to generate this type of errors are quite straightforward; we just remove the NORK element. What needs to be taken into account is that, in

```
foma[1]: up du
   edun[ADL][A1][NR_HURA][NK_HARK]
   ukan[ADT][A1][NR_HURA][NK_HARK][PNT]
foma[1]: down edun[ADL][A1][NR_HURA]
   ???
foma[1]: down ukan[ADT][A1][NR_HURA][PNT]
   ???
foma[1]: down izan[ADL][A1][NR_HURA]
   da
   9da
foma[1]: down izan[ADT][A1][NR_HURA][PNT]
   da
   9da
```

Figure 9: First command shows morphological analyses of the word *"du"*. Second and third commands show empty outputs if the base form is not changed. Fourth and fifth commands show that by changing the base form and removing the NORK element, both analyses can be used to generate the word *"da"*.

this scenario, the base form of the input words needs to be changed in order to generate the output word. For example, if the input word is *"du"*, two morphological analyses will be given: `edun [ADL] [A1] [NR_HURA] [NK_HARK]` and `ukan [ADT] [A1] [NR_HURA] [NK_HARK] [PNT]`. Both can be used to generate the word *"da"*, but in either case we need to change the forms *"edun"* and *"ukan"* to the form *"izan"* (see Figure 9).

3.2.1.4.3 Nor-Nori → Nor-Nori-Nork

In some other occasions, such as when phrases like *"*Aholku kontrajarriak ematen ari digu"* are used instead of its correct form *"Aholku kontrajarriak ematen ari zaigu"*, the paradigm NOR-NORI-NORK is being used when NOR-NORI is supposed to be. Two main changes need to be done to the morphological analysis of the correct sentence so that this type of errors can be generated: adding the NORK element and changing the base form.

Taking *"zaigu"* as the input, the transducer outputs these two analyses: `izan [ADT] [A1] [NR_HURA] [NI_GURI] [PNT]` and `izan [ADL] [A1] [NR_HURA] [NI_GURI]`. No matter which of them is chosen, the NORK element (3rd person singular) has to be added, so the analyses read as follows: `izan [ADT] [A1] [NR_HURA] [NI_GURI] [NK_HARK] [PNT]` and `izan [ADL] [A1] [NR_HURA] [NI_GURI] [NK_HARK]`. What differentiates these two analyses is that the first one corresponds to a verb composed by a single element (*aditz trinkoa* - `ADT` tag) and the second one corresponds to an auxiliary verb (*aditz laguntzailea* - `ADL` tag). To generate the word that will make the sentence grammatically incorrect, *"digu"* in this particular example, either the base form of the first analysis needs to be changed from *"izan"* to *"ukan"* or the base form of the second analysis needs to be changed from *"izan"* to *"edun"*. Figure 10 illustrates this example.

```
foma[1]: up zaigu
   izan[ADT][A1][NR_HURA][NI_GURI][PNT]
   izan[ADL][A1][NR_HURA][NI_GURI]
foma[1]: down izan[ADT][A1][NR_HURA][NI_GURI][NK_HARK][PNT]
   ???
foma[1]: down edun[ADT][A1][NR_HURA][NI_GURI][NK_HARK][PNT]
   ???
foma[1]: down ukan[ADT][A1][NR_HURA][NI_GURI][NK_HARK][PNT]
   digu
   9digu
foma[1]: down izan[ADL][A1][NR_HURA][NI_GURI][NK_HARK]
   ???
foma[1]: down ukan[ADL][A1][NR_HURA][NI_GURI][NK_HARK]
   ???
foma[1]: down edun[ADL][A1][NR_HURA][NI_GURI][NK_HARK]
   digu
   9digu
```

Figure 10: First command shows morphological analyses of the word *"zaigu"*. All the other commands show how the word *"digu"* is only obtained when, as well as adding the NORK element, the base form is changed appropriately.

3.2.1.4.4 Nor-Nori → Nor-Nork

The last error we have worked on is the one where NOR-NORK is used when NOR-NORI is supossed to be used. For instance, this happens in the case of the sentence *"*Paisaia asko gustatzen nau"* (the correct sentence would be *"Paisaia asko gustatzen zait"*). The pattern that is commonly used to produce these changes is to add the 3rd person singular case as the NORK element and to use the case of the NORI element of the input as the NOR element of the output.

Going back to the previously given example, one of the possible analysis of the word *"zait"* is `izan [ADL] [A1] [NR_HURA] [NI_NIRI]`; the NOR component is the 3rd person singular case and the NORI component is the first person singular case. This means that, to obtain the word *"nau"*, first we need to use the case of the NORI element of *"zait"* in the NOR element, and then we need to add the NORK element. These changes will give us the following analysis: `izan [ADL] [A1] [NR_NI] [NK_HARK]`.

To generate errors from this family we also need to take into account the base forms and change them accordingly. This means that, if we have a verb with a single component, *"izan"* has to be substituted by *"ukan"* and, if we have an auxiliary verb, *"izan"* has to be substituted by *"edun"*, the exact same way we did in error 3.2.1.4.3.

3.2.2 Challenges

Even though the general error generation process seems straightforward, we have come across some problems while implementing it.

The first challenge has been the huge number of correct sentences we use as the starting point. In order to analyse all of them with Eustagger, first, we have had to separate all the sentences in files, each of them containing around 50.000 tokens (because that is the maximum amount of tokens the analyser can process in a single file). To do so, we have calculated the average number of tokens per sentence in the original corpus: around 16 tokens per sentence. If we assume that a single sentence contains 16 tokens, we can conclude that we need 3.125 sentences to gather 50.000 tokens. Therefore, we have divided the whole corpus in files that contain 3.125 sentences, obtaining a total of 2.987 files. Then, we have fed each of these files to the linguistic analyser. Taking into account that Eustagger performs a complex analysis, where disambiguation of each token is carried out, the high number of sentences to analyse has made this a lengthy process. In the end, around 20 days have been needed in order to analyse all the reference sentences.

The second setback has been caused by a mismatch between the tokenizer we have used in the reference sentences — Moses Tokenizer (Koehn et al., 2007) — and the tokenizer Eustagger uses. For instance, if we take the clause *"esklusibekin-eta."*, whereas Eustagger separates it in three tokens (see Figure 11), Moses considers it a single token.

A consistency between the tokenization of the original sentences and their analysis is crucial for our implementation of the error generation procedure. Specifically, it is important for the step where we iterate over the Eustagger analyses in search of words from which an error can be generated (i.e., words whose morphological analysis matches one of the error patterns previously described). During this step, each file we work with contains the analyses of 3.125 sentences, so we need to delimit the sentences. Let us use Figure 12 for illustrating how we define where a sentence ends and the next one starts. The figure shows the analyses of the sentences *"*gustura egingo nuen orain."* and *"*jon ez daki ezer."* and, as we can see, there is no blank line that separates the two analyses.

```
/<sortu>/
      ("sortu"  ADI SIN BURU DA-DU w11428,M-A-ADI-SIN-5744 @-JADNAG)
/<duelako>/
      ("*edun"  ADL A1 NR_HURA NK_HARK ABS MG w11429,M-A-ADL-1313
       @+JADLAG_MP_OBJ @+JADLAG_MP_SUBJ)
/<esklusibekin>/
      ("esklusiba"  IZE ARR SOZ NUMP MUGM w11430,M-A-IZE-ARR-22508 @ADLG)
/<->/<BEREIZ>/
/<eta>/
      ("eta"  LOT JNT EMEN w11432,M-A-LOT-JNT-15 @PJ)
/<.>/<PUNT_PUNT>/
```

Figure 11: Output produced by Eustagger given the sentence *"...sortu duelako esklusibekin-eta."*. The clause *"esklusibekin-eta"* is divided in three tokens and "-" is tagged as `<BEREIZ>`, which stands for *"bereizlea"* or "divisor".

```
/<gustura>/
      ("gustura" ADB ARR w16,L-A-ADB-ARR-11,lsfi13 @ADLG)
/<egingo>/
      ("egin" ADI SIN GERO DU w17,L-A-ADI-SIN-38,lsfi14 @-JADNAG)
/<nuen>/
      ("*edun" ADL B1 NR_HURA NK_NIK w18,L-A-ADL-42,lsfi15 @+JADLAG)
      ("*edun" ADL B1 NR_HURA NK_NIK MOS w18,L-A-ADL-44,lsfi16 @+JADLAG_MP_ADLG)
/<orain>/
    ("orain" ADB ARR w19,L-A-ADB-ARR-12,lsfi17 @ADLG)
/<.>/<PUNT_PUNT>/
/<jon>/
    ("Jon" IZE IZB PLU- ABS NUMS MUGM w68,L-A-IZE-IZB-8,lsfi82 @OBJ)
/<ez>/
    ("ez" PRT EGI w69,L-A-PRT-4,lsfi83 @PRT)
/<daki>/
    ("jakin" ADT A1 NR_HURA NK_HARK PNT w70,L-A-ADT-69,lsfi84 @+JADNAG)
/<ezer>/
    ("ezer" IOR IZGMGB ABS MG w71,L-A-IOR-IZGMGB-2,lsfi85 @OBJ)
    ("ezer" IOR IZGMGB ABS MG w71,L-A-IOR-IZGMGB-2,lsfi86 @SUBJ)
/<.>/<PUNT_PUNT>/
```

Figure 12: Output produced by Eustagger given the sentences *"*gustura egingo nuen orain."* and *"*jon ez daki ezer."*

Initially, looking at the format of the output analysis, we might think that each token in the original sentence corresponds to two lines in the analysis: the first line for the token itself and the second line for its morphological analysis. Therefore, if a sentence is 3 tokens long, its analysis should be given in 6 lines. However, the first sentence in our example already shows that this rule does not always apply: the word *"*nuen"* in the sentence *"*gustura egingo nuen orain."* is not completely disambiguated, which makes a single token in the original sentence have 3 corresponding lines in the analysis. Since we can not assume that doubling the number of tokens in the original sentences will give us the number of corresponding lines in the analyses, we have decided to iteratively match the tokens in the sentences with the lines in the analyses; we arrive at the end of a sentence every time a line in the analysis matches the last token of said sentence. Let us go back to our previous example to introduce this method step by step. The sentence *"*gustura egingo nuen orain."* is formed by five tokens (*"gustura"*, *"egingo"*, *"nuen'*, *"orain"* and *"."*) and in the analysis file, we can see that the tokens are marked between the symbols "/<" and ">/". That being the case, we start with the token *"gustura"* and iterate over the lines in the analysis until we find one that start with the sequence "/<gustura>/" (in this case we find it in the first line). Once we find it, we take the next token, *"egingo"*, and we keep iterating over the lines in the analysis (starting from the line where we found the previous token) until we find the sequence "/<egingo>/". We repeat this process until we find a line that starts with the sequence "/<.>/"; when we find it, we know that we have arrived to the end of the sentence and that the next line that starts with the symbols "/<" will be the beginning of the analysis of the next sentence.

This method is the reason why we need the tokenization of the sentences to match the tokenization in the analyses. If we consider that *"esklusibekin-eta"* is a single token but in the analysis it has been considered as three different ones, when we try to find a line that contains the sequence "/<esklusibekin-eta>/" we will not be able to find it

```
***\&\ <BEREIZ>***
***\#\ <BEREIZ>***
***\91\ <ZENB>***
***\;\ <BEREIZ>***
```

Figure 13: Output produced by Eustagger given the string *"["*.

```
/<ikus-entzunezko>/
      ("ikus-entzunezko"  ADJ ARR IZAUR+ w4,M-A-ADJ-ARR-5632 @IA>)
/<edukien>/
      ("eduki"  IZE ARR BIZ- GEN NUMP MUGM w5,M-A-IZE-ARR-15726 @IZLG>)
/<ekoizpenari>/
      ("ekoizpen"  IZE ARR BIZ- DAT NUMS MUGM w6,M-A-IZE-ARR-15728 @ZOBJ)
```

Figure 14: Output produced by Eustagger given the sentence *"ikus-entzunezko edukien ekoizpenari"*. The clause *"ikus-entzunezko"* is considered as a single token and tagged as an adjective.

and, consequently, we will not know how to identify the boundaries of the sentences in the analysis file.

In order to fix this tokenization issue, we have considered finding all the occurrences of the "-" character in the original sentences and separating them from the words they are attached to. Nevertheless, there are two main reasons why this solution is not plausible:

1. The dash is not the only character that causes tokenization mismatches. Throughout the texts, we have also found HTML entity codes such as "[", which is the code that represents the left square bracket ("["). While Moses interprets this code as a single token, Eustagger divides it in 4 tokens (see Figure 13). We could also try to define a pattern to match all HTML entities in order to find them in the sentences and separate them in different tokens, but considering that we have come across this issue with dashes and HTML codes, it is likely to happen with more punctuation or encoding related symbols. Performing a search of the token that causes the mismatch and dividing it appropriately every time tokenization issues arise is not optimal, specially because, due to the large size of the corpus, chances of it happening frequently are high.

2. Eustagger does not always follow the same pattern for tokenizing dashes. For instance, given the sentence *"ikus-entzunezko edukien ekoizpenari"*, Eustagger considers that *"ikus-entzunezko"* is a single token, instead of separating it in three (see Figure 14). Even if we tried to define a method to manually handle the tokenization mismatches caused by this symbol, we would not be able to do it with just a surface analysis; we would have to find heuristics and patterns to determine how the "-" symbol should be tokenized in each case and, by doing so, we would be creating our own tokenizer.

Since there is no easy way to determine how to manually separate these special cases so that both tokenizations match, we have come to the conclusion that the easiest thing to do is to use the same tokenizer Eustagger uses in the original sentences. To do so, first, we have added a special token that indicates that the sequence has finished at the end of each sentence in the original corpus. Then, we have applied the same tokenizer Eustagger uses to these sentences, so as to obtain an output file that follows the format and tokenization of the Eustagger analyses. It should be noted that this is also a considerably long process: in order to tokenize all the sentences in the original corpus around 5 or 6 days have been needed. To end with, we have lined up the tokenization files and the analyses of the sentences (a couple of days have been needed to complete lining up process), in such a way that the special end-of-sentence token gets added to the end of each analysis. Like so, we have achieved to delimit the analyses with the special token and, as a consequence, counting the number of tokens in a sentence to know where its analysis ends is no longer necessary.

Lastly, the amount of time needed to generate the actual errors should also be mentioned. In the end, over two months have been needed to create all the incorrect sentences that will be used to train the Error Correction models (the exact number of sentences will later be discussed in Section 4.2). This is due to the fact that, for each of the millions of sentences we have worked with, every token in the sentence is analysed to determine whether an error can be created from it and, if the possibility exists, several FST calls and replacement operations are performed to generate the new (incorrect) sentence.

3.3 Error type tagging

As mentioned in Chapter 1, apart from correcting grammatical errors, we also aim to build a system that is capable of, given any sentence written in Basque, detect whether it contains a grammatical error and, if so, identify its type and the fragment of the sentence where it occurs.

To do so, a corpus of incorrect sentences where the errors are tagged is needed and, as such, we have extended the error generation procedure mentioned in the section above with a feature that allows us to, not only create a grammatically incorrect sentence, but also tag said sentence with the type of error that has been applied to it.

In order to explain how we have achieved this, it is important to recall how our error generation process begins: by identifying from which words of a sentence an error can be generated. Every time we find a match between one of the patterns listed in Table 2 and the morphological analysis of a sentence, as well as knowing that an error can be generated, we also know exactly what the error that can be generated is. We have used this information to, after following the whole process explained in Section 3.2, concatenate the newly created word (the one that will make the sentence grammatically incorrect) and the error type. Then, instead of replacing the original word by the newly generated one, we have replaced it by the whole concatenation.

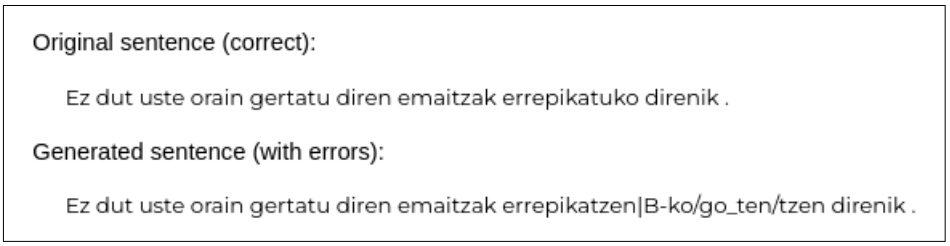

Figure 15: Example of how the sentences for error detection have been tagged.

As an example, the sentences generated following this method have been tagged as illustrated in Figure 15. The figure shows how the original sentence has been modified, by putting the word *"errepikatzen"* in place of the word *"errepikatuko"*, to create an error where the unfinished aspect is used when the future tense should be, as well as how the word *"errepikatzen"* is followed by the tag "B-ko/go_ten/tzen", which is the label we use to identify this kind of error.

Let us note that, in the previous example, the "B-" part of the "B-ko/go_ten/tzen" tag indicates that the token it refers to is the first token of the error. As for now, our approach only contemplates single word modifications for generating errors, but this label has been added in case future experiments lead us to creating multi-word errors or sentences with multiple different errors in them, in which case we would need to identify whether two consecutive tokens that make the sentence incorrect belong to the same error or not.

3.4 Overview of the Generated Corpora

A small sample of the results obtained with our error generation method can be seen in Table 5. Original sentences are shown in the first column, the generated sentences in the second column and which error has been applied in each case in the third column.

Among all the generated sentences, different degrees of errors can be found. One of the objectives of this work, to create grammatically incorrect sentences that are similar to real-life errors, has been achieved in cases like *"*bihar joaten gara erosketak egitera"*, *"*ez dut uste etorriko dela"*, *"*gustatu beharko zitzaizun"* or *"*abokatuak helegitea aurkeztu zuten"*, to mention a few. Even if some less realistic sentences have also been created, such as *"*niri orkestraren soinua lantzea gustatzen nau"* or *"*1995ean jaio litzatekeen"*, they still show errors that are much more plausible than those generated with simple token-operations such as random insertion, deletion or replacement of words.

It is important to mention that a small number of grammatically correct sentences have also been generated, those that are not marked with the symbol "*" in the second column of Table 5. For now, in order to delimit the scope of this project, we will assume that, out of the million of sentences we have created, only a minority are grammatically correct and we will not try to identify and remove those that are correct. We will discuss what could be done to treat these cases in Chapter 6.

Correct sentence	Generated sentence (with errors)	Applied change
bihar joango gara erosketak egitera	*bihar joaten gara erosketak egitera	ko/go –>ten/tzen
gustura egingo nuke orain	gustura egingo litzateke orain	NR-NK –>NR
atzo kalean ikusi zintudan	*atzo kalean ikusi nizun	NR-NK –>NR-NI-NK
afaltzera gonbidatu ninduen	afaltzera gonbidatzen ninduen	burutua –>burutugabea
gustatu beharko litzaizuke	*gustatu beharko zitzaizun	nuke –>nuen
3,6 milioi lagunek laguntza behar dute	*3,6 milioi lagun laguntza behar dute	erg –>abs
langileek lan handia egin dute	langileek lan handia egiten dute	burutua –>burutugabea
ez dut uste etorriko denik	*ez dut uste etorriko dela	denik –>dela
aholku kontrajarriak ematen ari zaigu	*aholku kontrajarriak ematen ari digu	NR-NI –>NR-NI-NK
niri orkestraren soinua lantzea gustatzen zait	*niri orkestraren soinua lantzea gustatzen nau	NR-NI –>NR-NK
1995ean jaio zen	*1995ean jaio litzatekeen	nuen –>nuke
kontzertu bakarra emango dugu beraiekin	*kontzertu bakarra ematen dugu beraiekin	burutua –>burutugabea
abokatuek helegitea aurkeztu zuten	*abokatuak helegitea aurkeztu zuten	erg –>abs

Table 5: Example of initial sentences and the sentences generated from them. The generated grammatical errors are marked in red.

Lastly, it should be noted that, although our aim has been to work with incorrect sentences that contain a single error, we have observed that in a few cases, specially in long sentences, multiple errors of the same type have been created. This is due to the fact that, the longer a sentence, the higher the probability of using a common word more than once. Consequently, if said word is the one we have randomly selected to modify and generate an error from, when making the final substitution and replacing the original word by the new word, it will be replaced in all its occurrences in the initial sentence. For example, let us suppose that our original sentence is *"Nahi duena egiteko libre da, baina etxeak su hartzen badu, arazoa ez da berea bakarrik izango"* and that we have chosen to modify the fifth word, *"da"*, to create a verbal paradigm related error. After applying our error generation algorithm, we will have created the word *"du"*, but since *"da"* appears twice in the input sentence, *"du"* will replace it in both cases, obtaining the sentence *"*Nahi duena egiteko libre du, baina etxeak su hartzen badu, arazoa ez du berea bakarrik izango"*. Improving our algorithm so that such cases are controlled is one of the tasks considered for the future steps in this work.

4 Error Correction

This chapter delves into the experimentation carried out for building a Neural Grammar Error Corrector and the results obtained when using the implemented systems for solving the GEC task. More specifically, the approach taken and the architecture chosen for training are briefly described, the proposed training settings are detailed and the quality of the developed models is discussed.

As previously stated, the present work aims to be the continuation of the work submitted as the Final Degree Project, where a first attempt at building a Neural Grammar Corrector was made. The main conclusion that was drawn from said work was that, in order to build a neural grammar checker capable of performing reasonably well, generating a training corpus by applying simple token replacement, insertion and deletion operations was not enough. Thus, the objective of this section is to analyse whether training a neural grammar checker with a corpus created using a more complex error generation process successfully improves the quality of the corrections.

4.1 Approach and Architecture

Since we want this work to be a continuation of the work done in the Final Degree Project, we have decided to stick to the approach taken in said project: considering Grammar Error Correction as a particular case of Machine Translation (MT). In MT tasks, a text written in a certain source-language is translated to a different target-language. If we view GEC as a specific case of MT, the source and target texts will be written in the same language, but the source will contain grammatical errors and the target will be, ideally, the corresponding correction of the source text. In other words, the incorrect text will be "translated" to a correct text.

Figure 16 showcases the parallelism between the two tasks. In the illustrated example, the source-language for the MT task is Spanish and a translation to the target-language — Basque — is obtained. Similarly, in the GEC example, taking an incorrect sentence as the starting point, a "translation" (correction) to a correct sentence is obtained.

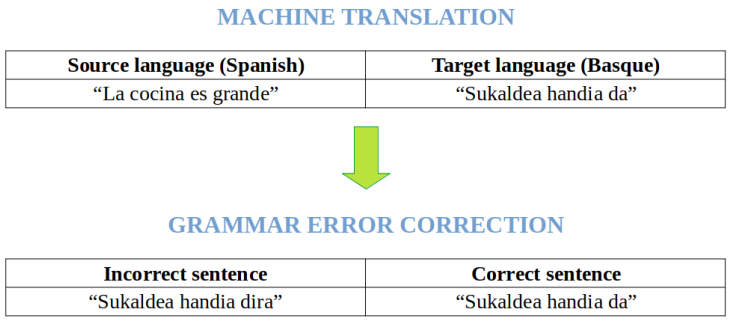

Figure 16: Parallelism between Machine Translation and Grammar Error Correction.

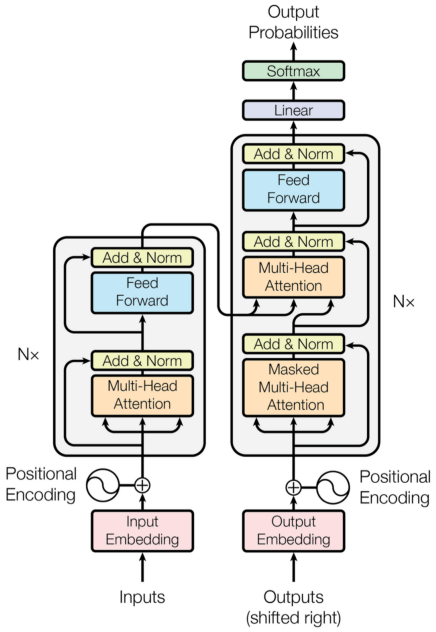

Figure 17: Architecture of the Transformer model.

Source: A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.Ñ. Gomez, Ł. Kaiser,and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems

As for the architecture chosen to train a neural grammar checker, we have also decided to stick to the architecture used in the Final Degree Project: the Transformer, a Deep Learning model introduced in 2017 by Vaswani et al. (2017). We have selected this architecture for our task because Transformers have achieved state of the art results in Machine Translation across different languages (Takase and Kiyono, 2021; Liu et al., 2020).

Transformers are sequence-to-sequence architectures, that is, they take a sequence as input and transform it into another sequence. The main components of a Transformer model are an Encoder module (composed of Nx encoders), a Decoder module (composed of Nx decoders) and an attention mechanism. The Encoder takes the input sequence and maps it into a higher dimensional space. Then, that representation of the input sequence is fed into the Decoder, which turns it into an output sequence. The attention mechanism is in charge of deciding, in each step, what the most relevant segments of the input sentence are. The detailed architecture of the Transformer model is shown in Figure 17.

Transformers were proposed in order to overcome the weaknesses previous models for sequence processing had, and have since become the state of the art models in Natural Language Processing tasks (Wolf et al., 2020). For instance, one of the strengths of the Transformers is that it is capable of processing the entire input sequence at once, on the contrary to other sequential data processing architectures such as Recurrent Neural Networks, where the elements of the input sequences have to be read one by one. This allows for parallelization and therefore considerably reduces training times. In addition,

being able to process multiple input sequences at the same time also implies having the ability of training on larger datasets, which has led to the development of Transformer-based models such as BERT (Devlin et al., 2018) and GPT (Radford et al., 2018). These models are trained on huge amounts of data and can be fine-tuned for multiple tasks. Nowadays, these models can be found behind technologies we use on our every day lives, such as Google queries (Schwartz, 2020), or even in more complex scenarios like tools for automatic code search and generation (Feng et al., 2020).

4.2 Experiments

As an initial experiment, with the aim of ensuring that our implementation of the Transformer works properly and is able to learn, we have fed the raw training corpus presented by the Elhuyar Foundation (i.e., the "Train" corpus introduced in Table 1) into our own code and trained a model. This model has been able to obtain comparable results to those presented in the original paper (these results are discussed in detail in Section 4.3.2) and, thus, our implementation is proven to be appropriate for the task at hand.

Once we have made sure that our code is correctly implemented, we have trained two Error Correction systems with our own corpora: one that, for all sentences in the training corpus, contains at least an automatically generated grammatical error in the incorrect sentence of the pair and, a second system that, in addition to incorrect-correct pairs, also uses correct-correct pairs for training. Adding sentences that do not need a correction in the training corpus of an error correction model may seem contradictory, but it is important to do so because we want to ensure that the system does not propose any change if it is to receive a correct sentence as input. We do not want the system to learn that it always has to correct; we want it to be able to detect incorrect sentences and only suggest a correction for those.

For the first system, from now on referred to as *system1*, we have used 4.234.427 sentence pairs in the training corpus. For the second one, from now on referred to as *system2*, we have used over 9 million sentence pairs: the sentence pairs used in the first model plus 5 million correct-correct sentence pairs. The exact number of sentences is shown in Table 6.

The decision for choosing a total number of around 9M sentences comes from the paper by the Elhuyar Foundation. In their work, several systems are developed depending on the synthetic error generation method used for creating the incorrect sentences in the training

	Number of sentence pairs		
	Correct-Incorrect	Correct-Correct	Total
system1	4.234.427	0	4.234.427
system2	4.234.427	5.000.000	9.234.427

Table 6: Number of sentence pairs used for training the systems.

Iter.	Most freq. token pair	Combination	Sentence
0			t h e _ m a n _ i s _ i n _ t h e _ v a n
1	t h	th	th e _ m a n _ i s _ i n _ th e _ v a n
2	a n	an	th e _ m an _ i s _ i n _ th e _ v an
3	th e	the	the _ m an _ i s _ i n _ the _ v an

Figure 18: Example of the BPE algorithm.

corpus. With the aim of carrying out the fairest comparison of results possible, among all the presented systems, we have chosen the one where incorrect sentences in the training corpus were generated by applying a single, randomly selected error to each of them[7] (because we too have only applied a single error type per sentence when generating incorrect sentences). Said system presented by Elhuyar was trained with a total of 9.33M sentence pairs, 4.92M of them being correct-correct pairs and 4.41M of them being incorrect-correct pairs, which is why we have aimed for a similar balance in the training corpus of our model.

As for the actual implementation and training of the models, it has been done following the Transformer architecture previously described and setting the hyper-parameters listed below:

- Vocabulary size: 32.000
- Batch size: 4.096
- Number of epochs: 10
- Maximum length (number of words) of the sentences: 90

It should be noted that, although we set a fixed vocabulary size of 32.000 tokens, which seems small considering our data is formed by millions of sentences, none of the tokens of the corpus are discarded. This is achieved thanks to a technique called Byte Pair Enconding (BPE), which is used for fixing the vocabulary size while still maintaining less common tokens, as well as for other data compression tasks (Gage, 1994). The BPE algorithm works as follows: initially, each character of the corpus is considered to be a token and then, for each iteration, the most frequent token pair in the corpus is combined, forming a new single token. Iterations are performed until the number of tokens matches the fixed vocabulary size. At the end of the iteration process, the corpus will have as many tokens as defined in the vocabulary size; the most common words will be complete and those that are less frequent will be divided in sub-tokens. Figure 18 illustrates the first iterations of the BPE algorithm applied in a very simple sentence.

Due to the large-scale data used for the experiments, the training processes of all the systems mentioned in this section have been carried out making use of Graphic Processing

[7]Other systems presented in the paper were trained using different incorrect sentence generation methods such as random word replacement or application of several error types per sentence.

Units (GPUs) courtesy of the IXA Research Group. Around 20 hours have been needed in order to finish the training process for the smallest system (the one that is trained on 4.2M sentences pairs). As for the biggest system (the one trained in over 9M sentence pairs), training time has doubled, needing around 42 hours to be completed.

4.3 Evaluation and Results

4.3.1 Evaluation Metric: ERRANT

ERRANT (*ERRor ANnotation Toolkit*) is a tool for automatically annotating parallel sentences (Bryant et al., 2017; Felice et al., 2016). To be precise, it compares a sentence with errors and its corresponding corrected version and annotates the differences proposed in the correction. It does so by creating a file where, for every incorrect-correct sentence pair, the fields "S" and "A" are defined. Lines in the file beginning with the symbol "S" (short for "sentence") list the incorrect sentence. Lines beginning with the symbol "A" (short for "annotation") contain the corrections proposed for that incorrect sentence (obtained by annotating the differences between the incorrect sentence and its, presumably, correct version). Although the annotation is formed by multiple fields for each proposed correction, only the first one and the third one are relevant for our particular task [8]. These fields correspond to the position of the correction (i.e., in which token the correction begins and in which token it ends, represented by the token identifiers) and the proposed correction, respectively.

By way of illustration, let us use ERRANT for annotating the grammatically incorrect sentence *"*umeak triste dago"* and one of its possible corrections: *"umeak triste daude"*. Figure 19 shows the output obtained when using these two sentences as input for ERRANT. As can be seen, the first line, the one that starts with "S", contains the incorrect sentence and the second line, starting with "A", the annotation. The "2 3" indexes[9] we see in the first field of the annotation mean that the correction is made from the second token to the third token, that is, only the second token is replaced. The word *"daude"*, shown in the third field, is the word that appears in the correction in place of the word that is in the second position in the incorrect one. Thus, the correction consists in replacing the word *"dago"*, which is the token with index number 2 in the incorrect sentence, by the word *"daude"*.

[8] Other fields include information about the error types (only works for sentences written in English) and the annotators.

[9] Token index count starts from 0.

```
S umeak triste dago
A 2 3|||R:NOUN|||daude|||REQUIRED|||-NONE-|||0
```

Figure 19: ERRANT output given the incorrect sentence *"*umeak triste dago"* and *"umeak triste daude"* as its proposed correction.

```
S mahaia hurbil dira
A 2 3|||R:NOUN|||dago|||REQUIRED|||-NONE-|||0

S umeak triste dago
A 2 3|||R:NOUN|||daude|||REQUIRED|||-NONE-|||0

S ni oporretan joango da
A 3 4|||R:OTHER|||naiz|||REQUIRED|||-NONE-|||0
```

(a) Incorrect vs Reference

```
S mahaia hurbil dira
A 2 3|||R:NOUN|||dago|||REQUIRED|||-NONE-|||0

S umeak triste dago
A 1 2|||R:NOUN|||pozik|||REQUIRED|||-NONE-|||0
A 2 3|||R:NOUN|||daude|||REQUIRED|||-NONE-|||0

S ni oporretan joango da
A 0 1|||R:OTHER|||aulkia|||REQUIRED|||-NONE-|||0
A 1 2|||R:OTHER|||hurbil|||REQUIRED|||-NONE-|||0
A 2 4|||R:OTHER|||dago|||REQUIRED|||-NONE-|||0
```

(b) Incorrect vs Automatically corrected

Figure 20: Comparison of annotations made by ERRANT for two different sets of corrections proposed for the same incorrect sentences.

As well as automatic sentence pair annotation, ERRANT also includes a feature that can be useful for evaluating our error correction systems, where two annotated files are compared and measured in terms of Span-Based Correction. First, we can generate an annotation that compares the incorrect sentences and the original correct sentences (see Figure 20a) and an annotation that compares the incorrect sentences and the corrections proposed by our model (see Figure 20b)[10]. Then, we can evaluate the model by comparing both of these annotations, which showcase the differences between the (supposedly) ideal correction and the one our model proposes.

We can use Figure 21 so as to explain the Span-based Correction evaluation method. In the depicted scenario, "*I often look at TV" is the incorrect sentence, and we can assume that its correct version is "I often watch TV". By comparing these two sentences in ERRANT, we would obtain the annotation [2, 4, watch], which would indicate that the correction consists in replacing the second and third tokens ("look at") by the token "watch". In order to meet the Token-based Detection criterion, the annotation generated when comparing the incorrect sentence and the one proposed by our system would have to show that the index of the first token to be replaced is properly identified. To meet the Span-based Detection criterion, the complete position — first token and last token — of the replacement needs to be guessed. If we want to pass Span-based Correction, the criterion ERRANT uses to measure the quality of the corrections, apart from identifying the exact position of the replacement, the exact token to be put in place also needs to be guessed. In other words, in terms of Span-based Correction, an automatic error correction system is only rewarded if the reference correction and the correction proposed by the system are identical.

[10]The corrections proposed in Figure 20b correspond to a very simple model trained with very few and simple sentences in order to test ERRANT's functioning; these corrections were not suggested by the systems described in Section 4.2.

TP	FP	FN	Prec	Rec	F0.5
2	4	1	0.3333	0.6667	0.3704

Table 7: Measures computed by ERRANT for the files in Figure 20.

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 21: Comparison between Span-based Correction, Span-based Detection and Token-based Detection.

Source: `https://www.cl.cam.ac.uk/research/nl/bea2019st/#eval`

By contrasting the two annotations and following the Span-based Correction criterion, ERRANT computes the measures shown in Table 7 (the exact values correspond to the the files previously illustrated in Figure 20).

Firstly, the number of True Positives (TP), False Positives (FP) and False Negatives (FN) is shown. Following with the example of Figure 20, the TP correspond to the corrections the system has guessed, that is, substituting *"dira"* by *"dago"* in the first sentence and replacing *"dago"* by *"daude"* in the second one. The 4 cases of FP are the ones that the automatic system has proposed but are not present in the original corrections: using *"pozik"* instead of *"triste"* in the second sentence and using *"aulkia"* instead of *"ni"*, *"hurbil"* instead of *"oporretan"* and *"dago"* instead of *"joango da"* in the third sentence. FN correspond to the corrections that should have been made (according to the reference sentences) but our system has not suggested, in this case, using *"naiz"* instead of *"da"* in the third sentence.

Then, using TP, FP and FN values, Precision (Prec) and Recall (Rec) are computed as shown in Equations 1 and 2, respectively. Precision measures how many of the corrections proposed by the system are appropriate and recall measures how many corrections the system has proposed among all the corrections that should be made.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Finally, the F0.5 measure is calculated, which gives double the weight to precision than to recall and is computed as depicted in Equation 3. This is the score we will use to measure the quality of our automatic error correction system, because giving more weight to precision than to recall indicates that False Positives are considered worse than False Negatives. This is accurate for the task at hand since, when using a grammar checker, it is important that the proposed corrections are highly accurate in order to gain user acceptance. Neglecting to propose a correction is not as bad as proposing an erroneous correction.

$$F_{0,5} = (1 + 0,5^2) \cdot \frac{prec \cdot rec}{(0,5^2 \cdot prec) + rec} \quad (3)$$

4.3.2 Evaluation Corpora and Results

The evaluation of our Error Correction systems can be divided in two steps: first, we have performed an initial evaluation of the system trained using Elhuyar's raw training data, just to make sure our implementation is correct; then, we have evaluated the models trained with our own data.

For the initial evaluation, the main objective has been to obtain comparable results to those presented in the original paper, which would let us deem our implementation as appropriate. So as to make a fair comparison, we have evaluated the system using the same test sets that are used in the original paper, that is, the DeaSingle and DeaMulti datasets previously introduced in Section 3. Both datasets were automatically built and contain 2.000 correct-incorrect sentence pairs but, whereas the incorrect sentences of the DeaSingle dataset contain a single error, the incorrect sentences of the DeaMulti dataset contain multiple errors.

The F0.5 scores obtained by both systems with respect to these two datasets are shown in Table 8. We refer to the system presented in the original paper as "Reference System" and to the system we have trained as "Our System". It is worth to reiterate that both systems have been trained using the same dataset, a dataset in which incorrect sentences have multiple errors per sentence. As can be seen, for both test sets, the systems have obtained similar results, being ours a little bit higher. This proves that our Transformer implementation is valid for training systems that aim to solve the GEC task.

For evaluating our actual error correction systems (i.e. the ones we have trained using our own training datasets), we have also used the DeaSingle and DeaMulti sets, because we want to make a fair comparison between the reference model and the ones we have trained. Note that, in this scenario, making a fair comparison implies that we will be comparing against the model in the reference paper that has been trained using a single error per incorrect sentence, which is not the model that obtained the best overall score in said paper.

	DeaSingle	DeaMulti
Reference System (best performing)	0.83	0.88
Our System	**0.8638**	**0.8984**

Table 8: F0.5 scores obtained by the reference system (the best one out of all the systems presentenced in the reference paper) and our system with respect to the DeaSingle and DeaMulti datasets.

	TP	FP	FN	Prec	Rec	F0.5
system1	**1035**	900	**965**	0.5349	**0.5175**	0.5313
system2	919	**327**	1081	**0.7376**	0.4595	**0.6579**

Table 9: Performance of the systems with respect to the DeaSingle evaluation set.

	TP	FP	FN	Prec	Rec	F0.5
system1	1397	660	2950	0.6791	0.3214	0.5555
system2	**1500**	**514**	**2847**	**0.7448**	**0.3451**	**0.6047**

Table 10: Performance of the systems with respect to the DeaMulti evaluation set.

The ERRANT scores obtained when evaluating our two systems against the DeaSingle and DeaMulti sets are shown in Tables 9 and 10. As a reminder, system1 is the model that has been trained with over 4 million correct-incorrect sentence pairs and no correct-correct pairs. On its part, system2 has been trained trained with the same correct-incorrect sentence pairs plus 5 million correct-correct pairs. We can contrast the F0.5 scores we have obtained with the ones obtained by the reference model presented in the paper (the one trained with a single error per incorrect sentence) in Table 11.

As we can see, for the two test sets, both of our systems achieve a lower F0.5 score than the reference system. However, when correcting the sentences that form those test sets, our model is likely to obtain worse results than the model presented in the reference paper, because, at training time, the reference model has seen sentences generated with the same rules that have been used to create the test sets (and these rules differ from the rules we use in our error generating method). Therefore, we have created a third test set, from now on referred to as OwnDeaSingle, applying our own rules for error generation. This test set contains 2.000 correct-incorrect sentence pairs in which the incorrect sentences contain a single error. By also evaluating our models against this test set, we will be able to determine whether our systems are strongly dependent on our corpus generation method and are only able to correct incorrect sentences that follow those patterns or, on the contrary, are able to generalise and have the ability to correct grammar errors generated in multiple ways. The ERRANT scores obtained by our models with respect to the OwnDeaSingle test set are shown in Table 12.

This time, we can see that both of our systems obtain an F0.5 score of above 0.89, which is higher than any of the F0.5 scores seen in all the previous experiments and even higher than the scores obtained by the reference system with respect to the test sets that were created using the same set of rules that was used for training said model. This shows, on the one hand, that our models have learnt to correct incorrect sentences that contain the same type of errors that have been used for training them and, on the other hand, that the performance of these models is in fact a bit dependent on the type of corpus used for training them.

	DeaSingle	DeaMulti
Reference System (single error)	**0.81**	**0.79**
system1	0.5313	0.5555
system2	0.6579	0.6047

Table 11: F0.5 scores of the reference system (the system presented in the reference paper that is trained with a single error per incorrect sentence) and our systems with respect to the DeaSingle and DeaMulti evaluation sets.

	TP	FP	FN	Prec	Rec	F0.5
system1	**1857**	213	**216**	0.8971	**0.8958**	0.8968
system2	1761	**121**	312	**0.9357**	0.8495	**0.9171**

Table 12: Performance of the systems with respect to the OwnDeaSingle evaluation set.

Nevertheless, we can say that our models have been able to generalise up to certain extent and learn how to correct some patterns. Our best model (system2), when faced against sentences that have been created using a different approach to those seen at training time, obtains an F0.5 score of 0.66 points correcting sentences with a single error in them and an F0.5 score of 0.61 points correcting sentences with multiple errors in them. These scores are considerably higher than the 0.42 F0.5 score we were able to obtain in the Final Degree Project when we attempted to correct incorrect sentences that were automatically created, specially if we take into account that, in those experiments, sentences in the training and test sets were generated using the same rules. This further proves that sentence created with our new error generation method have a closer resemblance to more realistic grammar errors and that using them for training a GEC system improves the performance of the model.

Another observation that is important to make is that, for all test sets, the number of False Positives obtained by system2 is lower than those of system1. A lower number of False Positives implies that the system is less prone to trying to correct something that is not an error. This is due to the fact that, while all training pairs used in system1 contain at least an error in the incorrect sentence, system2 is also given correct-correct pairs for training. Thanks to those correct examples the model sees at training time, it is capable of learning that not all sentences need correction and it is able to identify, at least to a certain degree, when those corrections should be proposed.

Still, although it has been proven that adding correct examples in the training corpus does help, the number of correct instances to add can be hard to determine. This can be seen by comparing the number of False Negatives obtained by the systems in both scenarios where the sentences in the test sets contain a single error. In the case of the DeaSingle set, while system1 ignores an error and does not correct it 965 times, system2 does so 1.081 times. As for the OwnDeaSingle set, system1 incorrectly deems an error

Incorrect	Correct	system1	system2
ikertzaileak nitratoak zer ondorio izan ditzaketen aztertzen ari dira .	ikertzaileak nitratoek zer ondorio izan ditzaketen aztertzen ari dira .	ikertzaileak nitratoek zer ondorio izan ditzaketen aztertzen ari dira .	ikertzaileek nitratoak zer ondorio izan ditzaketen aztertzen ari dira .
metal astunak uretan disolbatutak dauden substantziak dira .	metal astunak uretan disolbatuta dauden substantziak dira .	metal astunak uretan disolbatutak dauden substantziak ditugu .	metal astunak uretan disolbatutak dauden substantziak dira .
nire gelako bati deitu egin nion ikusi niolako .	nire gelako bati dei egin nion ikusi nuelako .	nire gelako bati deitu egin nion ikusi nuelako .	nire gelako bati deitu egin nion ikusi nuelako .
baina pezetaren debaluazioa alde txarrak ere bazeukan .	baina pezetaren debaluazioak alde txarrak ere bazeuzkan .	baina pezetaren debaluazioak alde txarrak ere bazeukan .	baina pezetaren debaluazioak alde txarrak ere bazeukan .
hauteskundeen sistemaren arabera , bi buelta dago lehendakari aukeratzeko .	hauteskundeen sistemaren arabera , bi buelta daude lehendakaria aukeratzeko .	hauteskundeen sistemaren arabera , bi buelta dago lehendakari aukeratzekoek .	hauteskundeen sistemaren arabera , bi buelta dago lehendakari aukeratzeko .
anek gu etxera ekarri digu .	anek gu etxera ekarri gaitu .	anek guk etxera ekarri digu .	anek gu etxera ekarri gaitu .
gehiago ikasi izan balu , azterketa gaindituko luke , baina huts egin du .	gehiago ikasi izan balu , azterketa gaindituko zukeen , baina huts egin du .	gehiago ikasi izan balu , azterketa gaindituko luke , baina huts egin da .	gehiago ikasi izan balu , azterketa gaindituko luke , baina huts egin du .
ziur azterketaren emaitzak bihar esaten dituztela .	ziur azterketaren emaitzak bihar esango dituztela .	ziur azterketaren emaitzak bihar esango dituztela .	ziur azterketaren emaitzak bihar esango dituztela .
adimen naturala eboluzioa du eta artifiziala berriz ez .	adimen naturalak eboluzioa du eta artifizialak berriz ez .	adimen naturala eboluzioa da eta artifiziala berriz ez . ***	adimen naturala eboluzioa da eta artifiziala berriz ez . ***
egia da urte berri hau gauza txarrak ekarri dituela .	egia da urte berri honek gauza txarrak ekarri dituela .	egia da urte berri hau gauza txarrak ekarri direla .	egia da urte berri hau gauza txarrak ekarri dituela .

Table 13: Examples of grammatically incorrect sentences automatically corrected by our systems. The *** symbol indicates that the proposed correction is grammatically correct even if it does not match the human correction.

as correct 216 times, but system2 does so 312 times. This indicates that system2 is more likely to not propose a correction when it should, probably because said error resembles one of the correct-correct examples the model was trained with. Taking this into account, we can conclude that setting an appropriate number of correct-incorrect and correct-correct pairs when building the training corpus is important, and further experiments should be performed in order to find the perfect balance.

Finally, in order to show the actual performance of our models, Table 13 shows examples of sentences written by humans that contain some type of grammar error and the corrections proposed by our models, as well as what the actual correction should be[11]. In the first column, where the incorrect sentences are listed, the errors are highlighted in red. In the second column, where the corresponding human-proposed corrections appear, the word that should have been used (and makes the sentence correct) is highlighted in green. In the third and fourth columns, where we can see the corrections proposed by our systems, the words that have been properly identified as incorrect and corrected according to the reference (True Positives) are highlighted in green, those that have not been corrected (False Negatives) are highlighted in red and words for which a correction has been proposed but did not need any change (False Positives) are highlighted in purple.

[11]These reference corrections are proposed by people with advanced Basque knowledge.

It can be clearly seen that, while system1 has a tendency for proposing changes that are not needed, system2 is more conservative and almost never proposes changes that should not be made. It is also obvious that, in many cases, the systems are not able to properly correct the sentences. This is because these are real life examples and, although we have aimed for designing a more realistic error generation method, it is still artificial; many errors that we have synthetically created and used for training will not be found in real scenarios.

5 Error Detection

This chapter focuses on our approach towards building a Neural Grammar Error Detector. First, the chosen strategy is described, as well as the tools used for implementing it. Then, the experimentation process is explained, where several iterations have been performed throughout the training phase. Finally, the obtained results are discussed.

5.1 Approach

Error Detection involves, given a grammatically incorrect text, detecting which elements make it incorrect and determining the type of grammar error that occurs in the text. In this case, we have decided to view the task as a particular case of the Named Entity Recognition (NER) task. NER systems aim to locate named entities — real-world objects that can be referred to with a proper name — in a given text and classify them in pre-defined categories such as 'person', 'organization', 'location' or 'date', among others. We can see the parallelism between the GED and NER tasks if, instead of recognising named entities, we try to identify grammar errors and, instead of classifying them according to the type of entity they belong to, we classify them in different error categories.

A comparison between both tasks is shown in Figure 22. Given the sentence "*Amaiak bazkaria prestatu du*", a NER system will recognise the word "Amaiak" as a named entity that falls under the 'person' category. If we were to use a GED system to analyse the same sentence, none of the words would be tagged because it is a grammatically correct sentence and the objective of the system is to detect errors. Now, let us suppose we apply our error generation rules to that sentence and replace the ergative subject with the absolutive, obtaining the incorrect sentence "**Amaia bazkaria prestatu du*". If we were to analyse this new sentence with a GED system, the word "Amaia" should be detected as the word that makes the sentence incorrect and using the absolutive instead of the ergative should be identified as the type of error.

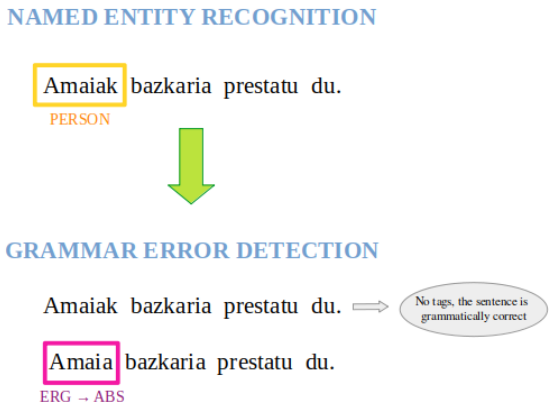

Figure 22: Parallelism between Named Entity Recognition and Grammar Error Detection.

5.2 Implementation

In order to build a system that is able to detect and classify grammar errors, we have decided to use Flair, a library for natural language processing that contains state of the art models but also facilitates new model training (Akbik et al., 2019). Flair allows us to train our own sequence labelling models by simply loading a training corpus and specifying the word embeddings[12] to use.

5.2.1 Corpora Adaptation

The training corpus we have used for this task is quite different to the one we have employed for GEC. In the previous task, we needed sentence pairs so that the model could learn which one was correct, which one was incorrect and how to "translate" from one to the other. In this case, we do not need sentence pairs; we need sentences in which the words that generate a grammar error are tagged. As aforementioned in Section 3.3, we have extended our error generation method so that obtaining sentences that contain those tags is possible.

For sequence tagging tasks, Flair requires the training data to follow a specific format:

- All the sentences in the training corpus are grouped in a single file.
- Each line of the training file contains a single word and its corresponding tag.
- A word and its tag are separated by a tab character.
- Sentences are delimited with a blank line.

As can be seen, one of the specifications indicates that all words must have a tag. However, we know that, for the GED task, only those words that generate errors need to be identified and, as such, only those should be tagged in the training corpus. This issue is easily fixed by a common practice for NER related tasks, which is to use the IOB2 format for tagging (IOB is short for inside, outside, beginning). The "I-" and "B-' prefixes are added before the tags: "B-' if the token that corresponds to the tag is the first one in the entity and "I-" if the token is part of an entity but is not the start of the entity. The "O" tag is used when a word does not have a corresponding tag, that is, when the word is not a named entity (or does not generate a grammar error in our case).

Taking all of the above into account, for building a training corpus that can be loaded in Flair, we have iterated over all the incorrect sentences (with tags) that we have generated, tokenized them and created a new file where we have a token per line. As for the corresponding tags, those words that generate a grammar error in the sentence will contain the string "|B-" in them, because our tagging method concatenated the generated incorrect word plus the error type (see the token "errepikatzen|B-ko/go_ten/tzen" in the example shown in Figure 15). Thus, if a token contains the substring "|B-", we have added what comes after the "|" symbol as the tag; otherwise, we have added the "O" tag.

[12]We define word embeddings as vector representations that encode the meaning of words. Words with a similar meaning are expected to be close in the vector space.

```
Europako        B-ORG
Banku           I-ORG
Zentralaren     I-ORG
egoitza         O
nagusia         O
Alemanian       B-LOC
dago            O
.               O

Carlos          B-PER
Garaikoetxea    I-PER
lider           O
bat             O
da              O
,               O
eta             O
gure            O
herriko         O
pertsona        O
historikoa      O
.               O
```

(a) NER tags.

```
Ikastolako      O
oporrak         O
hartu           O
berri           O
zaizkion        B-NR-NK_NR-NI
haurrek         O
,               O
aurten          O
,               O
olentzero       O
desberdin       O
bat             O
aurkituko       O
dute            O
.               O

Asierren        O
lana            O
bere            O
taldekideak     B-erg_abs
baliatu         O
zuten           O
hainbat         O
distantziatik   O
saskiratuz      O
.               O
```

(b) GED tags.

Figure 23: Comparison between annotated sentences for NER and GED.

In Figure 23 we show examples of what would be very small training files for the NER and GED tasks, respectively.

5.2.2 Training

Once our training corpus follows the appropriate format, training a model with Flair is simple[13]. First, we have loaded the corpus and specified the column properties: the first column contains the token list and the second column the tags. It is important to indicate Flair that this second column is the one the model should learn to predict. Then, we have defined which word embeddings to use. Several experiments have been carried out by combining different word embeddings. Next, we have defined the tagger (i.e., our GED model) and initialised it with the embeddings we are going to use and the list of tags that are present in the training corpus. Finally, we have initialised the trainer by specifying that we want to use our corpus for training. For all the experiments that are explained in this chapter, the models have been trained using the following hyper-parameters:

- Learning rate: 0.1
- Mini batch size: 32
- Number of epochs: 10

[13]The detailed procedure for model training is explained in the following site: `https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL_7_TRAINING_A_MODEL.md`

We have trained several models following this procedure and varying the training corpus or the embedding combinations in each case. All those experiments are detailed in Section 5.4. For all cases, we have trained the models using GPUs of the IXA Research Group.

5.3 Evaluation Metrics

In addition to all the possibilities it offers regarding model training, Flair is also useful for performing model evaluation. If, apart from the training set, a test set is also defined when initialising the corpora for an experiment, Flair automatically evaluates the model it just trained (with respect to said test set) according to F1 micro score, F1 macro score and Accuracy.

Whereas the F1 macro of a system is computed by first calculating the F1 score for each of the classes and then averaging all those scores, the F1 micro is computed by calculating the metrics globally, that is, counting the total True Positives, False Negatives and False Positives, using them to calculate the Precision and Recall (see Equations 1 and 2) and then computing the F1 as shown in Equation 4.

$$F_1 = 2 \cdot \frac{prec \cdot rec}{prec + rec} \tag{4}$$

Accuracy, on its part, is computed by dividing the number of correct predictions made by a model by the total number of samples, that is, by adding the number of True Positives and True Negatives and dividing them by the sum of True Positives, False Positives, True Negatives and False Negatives (see Equation 5). This metric needs to be used with care, because it can sometimes be deceiving: since True Negatives are also counted as right guesses for the model, it is easy to obtain a high Accuracy score. To give an example, let us imagine a sentence that is formed by 10 tokens and that one of those tokens is an error. If given that sentence the model predicts that it is correct and considers all tokens to be correct, it will still achieve a 90% Accuracy, because it has guessed that the other 9 tokens are not errors. Therefore, for tasks like the one we are working on where the vast majority of tokens are not errors, Accuracy is not the most reliable metric, so we will mostly be focusing on F1 scores.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

5.4 Experiments and Results

This section delves into the experimentation carried our for the Grammar Error Detection task. The section is divided in three subsections: initial experiments, error detection without error types and false positive analysis. In the first subsection, the earliest experiments are explained, including preliminar tries and more definite models. In the second subsection we talk about experiments performed in order to better understand the results

obtained in the first subsection. Finally, subsection three is used to define experiments that aim to fix issues found in the experiments carried out in subsection two.

5.4.1 Initial Experiments

As an initial experiment, in order to make sure our approach is appropriate for the task at hand, we have decided to take a random number of incorrect sentences generated by us, 3.247 in this case, and build a corpus that follows the guidelines explained in Section 5.2.1. Then, we have decided to try different combinations of embeddings for training multiple models with said corpus and compare the results. For evaluation, we have built a smaller corpus by generating 101 sentences that also contain grammar errors.

In this experimentation phase, we have opted for working with four embedding types:

- Character embeddings. They are representations of words built by looking at their character-level composition.
- Word embeddings. They are static embeddings, meaning that each word gets a pre-computed embedding and, no matter the context, the embedding for the word is always the same. Specifically, we have used the embeddings that have been pre-trained using the Basque Wikipedia.
- Flair embeddings (Akbik et al., 2018). They are contextual embeddings that capture syntactic and semantic information. They model words as sequences of characters because they are trained without any explicit notion of actual words. They are contextualised by their surrounding text, which means that the same word will have different embeddings depending on its context.
- Transformer embeddings. They are representations obtained by training transformer-based language models. Specifically, we have used the BERTeus embeddings, which have been trained on millions of sentences from Basque news articles and Basque Wikipedia (Agerri et al., 2020).

For our initial experiment, we have proposed five different combinations of those embeddings and trained a GED system with each of them, with the objective of determining which combination works best for the task at hand. In all cases, we have taken the Transformer embeddings as the basis, because of the state of the art performance of Tranformers across multiple tasks. The embedding combinations we have used are the following:

1. Transformer embeddings.
2. Transformer embeddings + Character embeddings.
3. Transformer embeddings + Word embeddings.
4. Transformer embeddings + Character embeddings + Word embeddings.
5. Transformer embeddings + Character embeddings + Word embeddings + Flair embeddings.

The results obtained by these 5 preliminar models are shown in Table 14. As can be seen, the highest scores are obtained by the model that has been trained using a combination of Character, Word and Transformer embeddings. In order to test the quality of this model, we have made up a small set of incorrect sentences and fed them to said model to see how capable of identifying errors it is.

Although for sentences such as *"*Gustatu beharko lizuke"*[14], *"*Eguraldi hobea egingo balu hondartzara joango nintzen"*[15] and *"*Ez dut uste emaitzak errepikatuko direla"*[16] the model has not been able to detect errors, we show successful error detection cases in Figure 24. Let us explain what the model has identified in each case:

- For the sentence *"*Nik Ariane deitzen naiz"*, we can see that the system identifies *"naiz"* as the word that should be corrected, and proposes that the error has occurred because the NOR element has been used instead of the NOR-NORK element. Although at first sight the obvious correction of this sentence would be to change the word *"Nik"* and use *"Ni"* in its place, so that the subject matches and the meaning of the sentence is "My name is Ariane", the modification proposed by the system is actually correct: if the NOR-NORK element is used instead of the NOR element, the sentence *"Nik Ariane deitzen dut"* is formed, which is grammatically correct and means "I call Ariane (with a certain periodicity)".
- For the sentence *"*Abiadura handiak balaztatze bortitzak eragiten zaizkio errepidean"*, the model identifies that the word *"zaizkio"* is the one that makes the sentence incorrect, the reason being the use of the NOR-NORI scheme instead of the NOR-NORK scheme. The prediction of the model is accurate: by using the NOR-NORK, we get the grammatically correct sentence *"Abiadura handiak balaztatze bortitzak eragiten ditu errepidean"*, which translates to "High speed causes violent braking on the road".
- For the sentence *"*Margolan hori gustatzen dit"*, the model is able to tell that the NOR-NORI element should be used instead of the NOR-NORI-NORK element, that is, that instead of the word *"dit"* the word *"zait"* should be employed, obtaining the correct sentence *"Margolan hori gustatzen zait"*, which translates to "I like that painting".

Even if the obtained results are somehow promising, we have realised that the systems could be learning features or grammatical types of certain words instead of learning how to identify errors and categorise them. For instance, if we have the word *"ikusten"* tagged as "ko/go_ten/tzen" in one of the sentences of the training corpus, it means that the correct sentence should use *"ikusiko"* in its place. However, instead of learning that another word

[14]The correction of the sentence *"*Gustatu beharko lizuke"* should be *"Gustatu beharko litzaizuke"*.

[15]The correction of the sentence *"*Eguraldi hobea egingo balu hondartzara joango nintzen"* should be *"Eguraldi hobea egingo balu hondartzara joango nintzateke"*.

[16]The correction of the sentence *"*Ez dut uste emaitzak errepikatuko direla"* should be *"Ez dut uste emaitzak errepikatuko direnik"*.

Train: 3247 sentences /// Test: 101 sentences			
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.7686	0.597	0.6286
Transformer + Character	0.7755	0.6726	0.6419
Transformer + Word	0.7607	0.6035	0.6181
Transformer + Character + Word	**0.805**	**0.7039**	**0.6783**
Transformer + Character + Word + Flair	0.7764	0.6595	0.6389

Table 14: F1 and Accuracy scores for preliminar error detection models using different embeddings. Only sentences with errors are used.

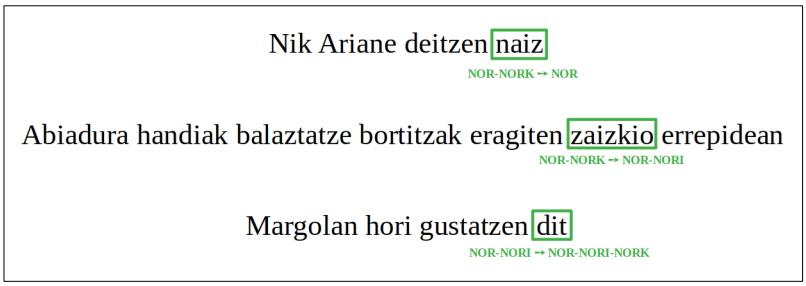

Figure 24: Examples of outputs obtained with the GED system that has been trained using BERTeus, word and character embeddings.

should be used in that case, the model might be learning that all words that end with the "-ten" suffix should be tagged with the "ko/go_ten/tzen" tag. In order to prevent this issue, we have decided to add correct sentences to the training corpus, that is, sentences where all tokens will have the "O" tag. In a similar manner to the GEC task, correct sentences will help the models learn that not all sentences contain errors and that not every instance should be corrected or tagged. In addition, having both correct and incorrect sentences in the training corpus, words that generate errors in some sentences are likely to appear in correct sentences without being tagged (because they do not generate an error in that case). For example, the word *"ikusten"* will be tagged in the sentence *"*gaur ikusten dut"* because *"ikusiko"* should be used in its place, but it is correctly employed in the sentence *"egunero ikusten dut"*, so it will not be tagged as an error in that scenario. The model will have to find patterns to understand which are the samples that need to be tagged and which are not and thus, will learn to detect actual errors.

Taking that into consideration, we have performed a new set of experiments where we have trained five new models — one for each of the embedding combinations previously mentioned — using the same 3.247 sentences with errors that we used in the previous training phase and adding 1.911 grammatically correct sentences. Table 15 shows the results obtained in this case when evaluating this model against a test set formed by the previous 101 sentences with errors and 81 new correct sentences. As we can see, by adding sentences without errors at training time, we have managed to improve the F1 micro and

Train: 5158 sentences // Test: 182 sentences			
1911 correct + 3247 with errors // 81 correct + 101 with errors			
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.786	0.6206	0.6475
Transformer + Character	0.7892	0.6125	0.6567
Transformer + Word	0.7748	0.6176	0.637
Transformer + Character + Word	**0.8182**	0.6181	**0.6977**
Transformer + Character + Word + Flair	0.8125	**0.6224**	0.6894

Table 15: F1 and Accuracy scores for error detection models using different embeddings. Sentences with no errors have been added in the training corpus.

Accuracy scores (both obtained with the model that combines transformer, character and word embeddings). However, the F1 macro is lower in 3 out of 5 models.

Considering that the training corpus used for our best GEC system contains more correct-correct pairs than correct-incorrect pairs and taking into account that adding correct sentences has helped improve F1 micro and Accuracy scores, we have decided to perform one last experiment: training GED models with a corpus that contains more correct sentences than incorrect ones.

Nevertheless, at the time of building such corpus, we have noticed that the incorrect sentences that the models have seen at training time up until this point might not be appropriate for the task. The reason behind this is that we have created multiple incorrect sentences taking the same original sentence as the basis, and therefore, most of the tokens of the original sentences appear several times, with the same tag, in the training corpus. Let us use the example previously shown in Table 3 to better illustrate this issue. In this case, from a single correct sentence, we can generate five incorrect sentences: *"*jaurlaritza gaur hasiko du urte politikoa, donostian"*, *"*jaurlaritzak gaur hasten du urte politikoa, donostian"*, *"*jaurlaritzak gaur hasiko dio urte politikoa, donostian"*, *"*jaurlaritzak gaur hasiko da urte politikoa, donostian"* and *"*jaurlaritzak gaur hasiko zaio urte politikoa, donostian"*. If we use these five sentences in the training corpus of a GED system, the word *"jaurlaritzak"* will always be given the "O" tag, that is, no errors will be related to it. However, the word *"jaurlaritzak"* itself is likely to cause an error related to ergative/absolutive subjects in other sentences. Even if the model should be able to determine when a word generates an error, if the number of times a single word is tagged as a non-error is considerably higher than the number of times it is tagged as an error, the system will be prone to always considering it a non-error, without even trying to determine whether an error exists. With the objective of mitigating this issue, we have decided to, first, build new corpora by creating a single incorrect sentence from each correct sentence and, then, repeat the whole training process.

For this final training phase, we have built five training sets and three test sets. The number of total sentences used in each of them and the balance between correct and

	Number of sentences		
	with errors	**correct**	**total**
train1	3250	0	3250
train2	3250	1910	5160
train3	3250	5165	8415
train4	3250	15000	18250
train5	3250	30000	33250

Table 16: Number of total sentences and distribution of correct and incorrect sentences throughout the different training sets for Error Detection.

	Number of sentences		
	with errors	**correct**	**total**
test1	100	0	100
test2	100	85	185
test3	100	825	925

Table 17: Number of total sentences and distribution of correct and incorrect sentences throughout the different test sets for evaluating the Error Detection systems.

incorrect sentences in each case is summarised in Table 16 (training sets) and Table 17 (test sets).

We have trained a model for each of these training sets and each of the embedding combinations mentioned before, which means that we have trained a total of 25 models. All the F1 and Accuracy scores that these models have obtained with respect to the three test sets we have built can be seen in Section B.1 of Appendix B.

Since we have a huge number of results to analyse, in order to simplify, we will consider that the third test set (test3) is the closest to a real-life scenario (because writing all — or most — sentences incorrectly is not realistic, at least for someone who has a certain degree of knowledge about a language) and we will use the results obtained with respect to that test set as our reference to draw conclusions from.

The first thing we can conclude by taking a look at the results is that adding correct sentences to the training corpus does help improve the performance, at least up to 15.000 correct sentences. For the train set with 30.000 correct sentences the F1 micro and Accuracy scores are a bit lower, but not significantly so to the point where we could affirm that adding 15.000 extra sentences makes the systems learn much less. Still, out of these two training sets, models trained with 15.000 correct sentences seem to perform better overall, even with respect to the test sets with less correct sentences (test1 and test2), so we show the results obtained by the models trained with this corpus in Table 18.

As for the embedding combination that performs best, we can see that, for all models trained with the train4 dataset, the setting that obtains the best results is using Trans-

Train: 18250 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
15000 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.7381	0.5555	0.5905	0.7006	0.5112	0.5439	0.6526	0.4778	0.4882
Transformer Character	0.764	0.5674	0.6355	0.0095	0.0088	0.0049	0.0025	0.0019	0.0013
Transformer Word	0.7349	0.5118	0.5865	0.7052	0.4932	0.5495	0.6354	0.4337	0.4692
Transformer Character Word	**0.7886**	**0.5859**	**0.6635**	**0.7541**	**0.5474**	**0.6161**	**0.6866**	**0.505**	**0.5308**
Transformer Character Word Flair	0.7485	0.5423	0.6095	0.7072	0.5081	0.5565	0.6305	0.4516	0.4672

Table 18: F1 micro, F1 macro and Accuracy scores for the models trained using 15.000 correct sentences and 3.250 sentences with errors.

former, Character and Word embeddings, which is the same embedding combination that obtained the best results in the preliminar experiments.

Taking all of the above into account, we can establish that using around 15.000 sentences without errors in the training corpus and combining Transformer, Character and Word embeddings for training the model seems to be the setting that achieves the best results for identifying tokens that generate an error in a sentence and tagging them appropriately according to the error type they create.

5.4.2 Error Detection Without Error Types

As we have just mentioned, all the results derived from the previous experiments show the performance of the models not only at detecting what token in a sentence makes it grammatically incorrect, but also the type of error it creates. If we focus on the error type classification part and count how many times errors of each type appear in our train and test sets (see Table 19), we can see that they are not balanced: in a set of 3.250 sentences, a certain error has been generated more than 600 times while other only occurs 4 times. This lack of balance is likely to make the system fail at classifying errors that are present very few times in the training corpus. In those cases, the system might be able to identify that a token does not fit within a sentence, but not the type of error that there is. If that occurs, even if the detection part of the task is accurately done, the system still receives a low score at evaluation time, because tagging the tokens appropriately is crucial for the NER approach we are following. So as to determine how often this occurs, we have decided to perform the same experiments as before without taking error types into account.

In order to evaluate how our systems work just for detecting incorrect tokens, without taking into account what type of error is made in each case, we have followed the next procedure:

1. Use each of the 25 models trained in the previous subsection to detect and classify errors in all the sentences of the three test sets.

	train	**test**
NR_NR-NK	645	22
NR-NK_NR-NI	564	17
NR-NK_NR-NI-NK	535	18
NR-NK_NR	501	10
buru_ezbu	359	7
erg_abs	264	15
nuen_nuke	192	5
ko/go_ten/tzen	131	4
NR-NI-NK_NR-NI	97	4
NR-NI_NR-NI-NK	72	1
NR-NI_NR-NK	56	0
denik-dena_dela	11	0
nuke_nuen	4	1
****exceptions**	3	0

Table 19: Number of errors of each type in the sentences with errors used in the Error Detection phase.

2. For each of the predictions, store the indexes of the tokens that the models identify as incorrect (if any).
3. For each of the sentences in the test sets, store the indexes of the tokens that are actually incorrect (if any).
4. For each sentence in the test sets and their corresponding prediction made by the models, compare the indexes of the tokens tagged as errors.
    - If the indexes match, count it as a True Positive
    - If we have stored an index in the prediction but not in the reference sentence, count it as a False Positive.
    - If we have not stored an index in the prediction but we have in the reference sentence, count it as a False Negative.
5. Compute the F1 score as shown in Equation 4.

All the TP, FP and FN counts obtained when testing our models, as well as the F1 scores, are shown in Section B.2 of Appendix B. Thanks to this evaluation, we are able to verify that the combination of Transformer, Character and Word embeddings plus a corpus with 15.000 correct sentences still seems to be the best option for this task, at least for the test scenario where the majority of the sentences do not have errors. In that case, the model trained in said setting obtains a F1 score of over 0.7 points (see Table 20), meaning that it is able to detect tokens that make a sentence incorrect 70% of the time.

Train: 18250 sentences	**Test: 100 sentences**				**Test: 185 sentences**				**Test: 925 sentences**			
15000 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer Character Word	71	0	33	0.8114285714285714	71	8	33	0.7759562841530053	71	26	33	0.7064676616915424

Table 20: True Positive, False Negative and False Positive counts and F1 scores for the model trained using a combination of Transformer, Character and Word embeddings, 15.000 correct sentences and 3.250 sentences with errors. Only detection of errors is evaluated, not classification according to error type.

Even if the best F1 score is obtained in that particular setting, the evolution that occurs when correct sentences are periodically added to the training corpus is noteworthy. For the systems trained using datasets train1, train2 and train3, where the maximum number of correct sentences used is just above 5.000, the models are capable of scoring a high number of True Positives, that is, they are able to correctly identify a lot of tokens that cause errors. However, the downside of those models is that they also score a high number of False Positives, which indicates that often times they mark as errors tokens that are actually correct. This is clearly an issue because, in practice, an error detection system that identifies elements that are correct as errors is not very useful, and would probably have a bad reception among users. It is safe to say that we would rather have a system that correctly identifies errors, even if not all of them, than a system that identifies all errors but also indicates that correct sentences should be modified. Often times, indicating that an error exists when it does not is directly connected to the fact that not that many correct sentences are used for training the models where this phenomenon happens. If the models, at training time, come across sentences that contain errors almost half of the time (or more, in the cases where train1 and train2 datasets are used for training), they are likely to learn that there should be an error to detect in most sentences.

The importance of showing correct sentences to the models is further proved when analysing the number of False Positives obtained by the systems trained with the train4 and train5 datasets: whereas the best model trained with few correct sentences wrongly identified 58 tokens as errors, the highest number of FPs obtained among the models trained with many more correct sentences is 33 (and it does not correspond to the model that obtains the best overall F1 score). Still, the perfect balance between correct and incorrect sentences to use in the training corpus remains unknown, because models trained using the train4 and train5 datasets show a decrease in the number of True Positives they obtain (compared to the models trained with the other three datasets).

Regarding this balance, since models trained with less correct sentences clearly have a better chance at learning more about the actual errors and, consequently, are better at guessing what tokens make a sentence incorrect, it would be interesting to analyse what the cause of the high number of False Positives is. If we are able to reduce the number of False Positives scored by these models, they will drastically improve and they will perform better than the models trained with more correct sentences. This is something we are

Train: 8415 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
5165 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer Word	84	2	20	0.8842105263157894	84	16	20	0.8235294117647058	84	70	20	0.6511627906976744

Table 21: True Positive, False Negative and False Positive counts and F1 scores for the model trained using a combination of Transformer and Word embeddings, 5.165 correct sentences and 3.250 sentences with errors. Only detection of errors is evaluated, not classification according to error type.

interested in because, although models trained with more correct sentences are less likely to say that there is an error when there is not, they are not as good at saying that there is an error when there actually is. The next subsection aims to explain experiments performed regarding the high number of False Positives obtained by some of the models.

5.4.3 False Positive Analysis

We have seen that models trained with less correct sentences are able to properly identify more than 80 errors out of test sets that contain a total of 104 errors. However, these models tend to also mark as errors words that are actually correct. For this set of experiments, we have decided to choose one of the best systems trained with less correct sentences and analyse the False Positives (i.e., the correct elements that are predicted as incorrect) it detects, so as to understand why this issue is happening and try to find a solution.

The model we have chosen is the one trained using 5.165 correct sentences and a combination of Transformer and Word embeddings. We have selected this system because, out of the systems trained with a smaller number of correct sentences, it achieves the best F1 score for test1 and test2, while also obtaining the second best F1 score for test3 (just 0.01 points below the best model). The exact F1 scores obtained by this system, as well as the TP, FP and FN counts, are shown in Table 21.

By analysing the sentences where this system believes that there is an error when actually there is not, we have realised that many cases are somehow ambiguous, that is, another word could be used instead of the word the system identifies as an error and the sentence would still be grammatically correct. For instance, this is the case of the sentence *"Sare sozialetan ere jarraitzaile kopurua goraka doa, eta urte osoan ia etenik gabe lan egiten jarraituko da"*. The system identifies the word *"da"* as incorrect, probably believing that the word *"du"* should be used in its place. It is true that, if we were to substitute *"da"* by *"du"*, we would obtain the grammatically correct sentence *"Sare sozialetan ere jarraitzaile kopurua goraka doa, eta urte osoan ia etenik gabe lan egiten jarraituko du"*. Still, the original sentence without modifications is grammatically correct, so the system saying that an error exists is not helpful for a potential user.

Our hypothesis for why the system considers cases like the one shown in the previous example as errors is that, because of the rules we have defined for error generation, the change between *"da"* and *"du"* frequently appears throughout the training corpus. Precisely, this occurs because of the rule where we find the NOR element and replace it by the NOR-NORK element (and vice versa). What we did not take into account when defining this rule is that *"da"* and *"du"* are extremely common words (which is why we assume that this modification can be found many times in the training corpus) and that, although substituting one by the other may generate errors in some cases, if the subject of the sentence is not specified, both can be used in a sentence and still have it be grammatically correct (but with a different meaning). In other words, it is quite possible that, by applying this rule, we have been generating sentences that are not actually grammatically incorrect and feeding them to the models as if they were.

In order to determine how often this has happened, as well as whether we have defined other rules that cause similar problems, we have taken the sentences where the 70 False Positives predicted by the system occur and we have looked at the actual error types the system assigns for each of those cases. What we have found further demonstrates our hypothesis: 27 out of the 70 cases the system wrongly predicts as grammatically incorrect are errors generated by replacing the NOR element with the NOR-NORK element and another 28 cases are errors generated by replacing the NOR-NORK element with the NOR element.

This clearly shows that part of the high amount of False Positives the systems produce is due to the fact that our error generation procedure can sometimes generate grammatically correct sentences. With the objective of fixing this issue, we have tried to see whether we can somehow discard those grammatically correct sentences that are generated from the training corpus. To do so, we have generated 50 new sentences by replacing the NOR element with the NOR-NORK element and another 50 new sentences where the NOR-NORK element is replaced by the NOR element. Then, we have used a Language Model trained with correct sentences written in Basque to calculate the probability of those 100 sentences belonging to correct Basque, as well as the probability of their corresponding 100 correct sentences (the 100 original sentences from which the new ones have been generated) of pertaining to the Basque language model. Next, for each sentence pair, we have computed the difference between the probability of the correct sentence and the probability of the generated sentence. Finally, by manually analysing the generated sentences, we have set a threshold to indicate that, if the difference between the probabilities of the original and the generated sentence is above said threshold, generally, it means that the generated sentence has clearer errors or is almost always incorrect without a doubt.

Once we have defined this threshold, we have applied it to the sentences in the training corpora, and re-trained the models shown in Table 20 and Table 21 using sentences that are above the threshold[17]. We show the comparison between the F1 scores and the number of TPs, FPs and FNs obtained by the models with and without applying the threshold

[17]So as to keep the same number of total sentences in the training corpus as in the previous experiments, every time a sentence has been discarded, a new one that scores above the threshold has been added.

	8415 sentences: 5165 correct + 3250 with errors // Transformer + Word embeddings							
	No threshold applied				Threshold: >2			
	TP	FP	FN	F1	TP	FP	FN	F1
test1 100 with errors	**84**	**2**	**20**	**0.8842105263157894**	71	5	33	0.7888888888888889
test2 85 correct + 100 with errors	**84**	16	**20**	**0.8235294117647058**	71	**13**	33	0.7553191489361704
test3 825 correct + 100 with errors	**84**	70	**20**	**0.6511627906976744**	71	**47**	33	0.6396396396396397

Table 22: Comparison between models trained using 5.165 correct sentences and a combination of Transformer and Word embeddings with and without applying a threshold to the sentences in the training corpus.

	18250 sentences: 15000 correct + 3250 with errors // Transformer + Word + Charcter embeddings							
	No threshold applied				Threshold: >2			
	TP	FP	FN	F1	TP	FP	FN	F1
test1 100 with errors	**71**	**0**	**33**	**0.8114285714285714**	56	**0**	48	0.7000000000000001
test2 85 correct + 100 with errors	**71**	8	**33**	**0.7759562841530053**	56	**1**	48	0.6956521739130436
test3 825 correct + 100 with errors	**71**	26	**33**	**0.7064676616915424**	56	**14**	48	0.6436781609195402

Table 23: Comparison between models trained using 15.000 correct sentences and a combination of Transformer, Character and Word embeddings with and without applying a threshold to the sentences in the training corpus.

in Table 22 (for the model trained with 5.165 correct sentences and a combination of Transformer and Word embeddings) and Table 23 (for the model trained with 15.000 correct sentences and a combination of Transformer, Character and Word embeddings).

If we analyse the results, we see that, in both cases and with respect to all test sets, training the models without applying the thresholds seems to work better. Even if, in almost all cases, the number of False Positives does decrease by using the threshold, the number of correct guesses made by the models is also considerably lower, which makes the overall F1 score be lower as well. This is strange because removing correct sentences that we were assuming to be incorrect, rather than making the models detect less errors, should help them have a clearer view of what actual errors are.

With the aim of identifying why this is happening, we have decided to analyse the sentences in the test sets and what the models are predicting for each of them. What we have realised by inspecting the incorrect sentences from the test set in greater detail is that many of them are actually correct. Since the supposedly incorrect sentences from the test set have also been automatically generated, we come across the same issue as before: sometimes correct sentences are generated. This might be the reason why the models trained with sentences that pass the threshold have achieved less True Positives: the systems have learnt to identify actual errors (those that are always errors, almost without a doubt) and have made their predictions accordingly, but in the test sets we have

tagged as errors cases that are not errors; even if the models rightly identify a sentence as correct, if it has been tagged as containing an error (when it should not have been), it will not count as a True Positive.

We believe that having an appropriate test set against which we can evaluate our models is crucial, because otherwise we will not be able to know how good at solving the task our models actually are. With that in mind, we have decided to manually revise the test set. To do so, we have read the 100 incorrect sentences in the test set an annotated them according to whether we thought the incorrect sentence of the pair was actually incorrect or not. Then, we have only kept those sentences that were considered incorrect by all annotators and we have generated new incorrect sentences (and repeated the annotation and discarding process) until we have built a new set of 100 incorrect sentences.

Once we have verified that the incorrect sentences are fitting for the task, we have re-built all our test sets using these 100 manually revised sentences (and maintaining the correct ones) and we have re-evaluated all our models (considering error types). The new scores can be found in Section B.3 of Appendix B. As we can see, for the most part, the F1 micro and Accuracy scores obtained when evaluating against the new test set are higher. This is a good sign, because we are getting better scores with a test set that is more appropriate for evaluating the task. As for the F1 macro, it does sometimes improve, but other times it is lower. Having a decrease in the macro score indicates that there might be certain error types that the models have trouble predicting. As future steps towards improving these systems, it could be interesting to identify which are the particular errors that the models have trouble guessing. Then, sentences with said errors could be generated and added to the training set, and whether the models are able to learn from those new instances and generalise with regards to that specific error type could be studied by comparing the F1 macro scores.

The model that obtains the best results against the third test set is the one trained with 30.000 correct sentences and just Transformer embeddings, achieving the highest F1 micro, F1 macro and Accuracy scores out of all the models and also higher scores than all of the scores obtained with the previous test set. However, the most constant model — the one that, for all three test sets, always obtains the highest or almost highest scores — is the one trained with 15.000 correct sentences and a combination of Transformer, Character and Word embeddings. Even if it does not score the absolute best F1 and Accuracy values in the third test scenario, it obtains just 0.0013 F1 micro points, 0.0402 F1 macro points and 0.0025 Accuracy points less than the best model, which we can say is a small difference for it to be considered significant. Altogether, we could conclude that, in our experience, the most successful setting for training a GED model is using 4 times more correct sentences than incorrect ones and combining Transformer, Character and Word embeddings.

6 Conclusions and Future Work

In this work we have completed our objective of continuing the line of research of our previous work. We have done so by exploring a methodology for creating more complex grammatical errors that aim to be more similar to human-made errors than those generated with surface operations. Said method has allowed us to generate over 4 million sentences that contain diverse types of grammar errors and that can be useful for training neural models for multiple tasks. Furthermore, we have been able to test this new error generation approach by training Grammar Error Correction systems and evaluating their performance across different settings. Results show that models trained with our synthetic errors achieve really good scores when correcting synthetic errors generated with the same method, as well as a considerably good performance when correcting synthetic errors that are generated using a different method. Correction of human-made incorrect sentences still needs improvement, but our models are capable of accurately correcting certain paradigm and subject related errors, demonstrating that the corpora created with the proposed error generation procedure does provide the models with a certain degree of knowledge.

Additionally, this work has also given us the chance to experiment towards building a Grammar Error Detection system that is capable of identifying grammar errors in a sentence and tagging them according to their type. In doing so, we have expanded our error generation method so as to create tagged synthetic data where error types are indicated next to the words that make the sentences incorrect. We have trained several detection models using these data, while also studying different embedding combinations for training and analysing which ones are more appropriate for the task. This experimentation phase has been specially insightful and has served to highlight the importance of using correct sentences at training time, due to the great impact they have at helping the models understand which cases do not contain errors. These experiments have also brought to light a downside of our error generation approach: generating grammatically correct sentences in certain occasions. This has proven to have a negative impact on the systems, because, if correct sentences that are tagged as incorrect are used for training, the information from which the models have to infer can end up being contradictory and thus, no real leaning can be made.

Overall, we have been able to complete the goals we set when we defined the scope of the project. The error generation method has been successfully improved, Neural Grammar Error Correction and Detection systems have been trained and tested and the developed systems have proven to have acquired a certain degree of knowledge about grammar rules in Basque.

As for future lines of investigation, there are several topics and tasks where we could continue our development. For instance, work still needs to be made regarding real-life incorrect sentences.

The first step towards improving the performance on real, human-made incorrect sentences, both for GEC and GED, would involve training a system with sentences that can

contain multiple errors. As seen in the real-life examples shown in Table 13, it is quite common to find sentences where two modifications need to be made in order to correct it. Therefore, it would be interesting to build a corpus combining correct sentences, sentences with a single error and sentences with multiple errors. This idea could also be extended and, instead of building a single corpus, multiple datasets could be constructed by varying the proportion of sentences according to the number of errors they have. In the case of GEC, it would be insightful to test models trained with these datasets against the DeaMulti dataset, because, if the performance with respect to that dataset improves (compared to the models presented in this work), the hypothesis that models do learn how to correct sentences that are very similar to those used for training would be further demonstrated. In addition, for both tasks, we would be able to analyse whether adding samples with more errors at training time improves the performance on real incorrect sentences.

Improvement could also be achieved by detecting correct sentences that are generated by our error generation method. In order to mitigate this issue, we could further experiment regarding the probabilities given by the language model to sentences that contain different types of errors, so as to see what the most problematic errors are. Then, we could try to define a new threshold that is fitting for all errors types and after applying it, re-train some models and evaluate them against our new manually revised test set without considering error types. This would show whether an appropriate threshold actually helps reduce the number of False Positives or, on the contrary, a different strategy should be used.

Regarding the particular task of error correction, we could try to extend the knowledge of our system and, not only provide it with the ability of correcting grammar errors but also those related to orthography. This could be useful because, when writing in Basque, words that contain *"ts"*, *"tx"* or *"tz"* are often incorrectly written and the use of the *"h"* letter also generates doubts frequently. So as to add such capability of correcting orthography errors, we could use the error generation method proposed in the Final Degree Project, the one that consisted on insertion, deletion and replacement operations. This time around, however, instead of performing those operations at token level, we could do so at character level. This way, we would generate sentences with a small amount of words where one or two characters should be corrected, and we could add them to the training corpus we have built in this work in order to train a new system that also considers orthography errors.

References

Rodrigo Agerri, Iñaki San Vicente, Jon Ander Campos, Ander Barrena, Xabier Saralegi, Aitor Soroa, and Eneko Agirre. Give your text representation models some love: the case for basque. arXiv preprint arXiv:2004.00033, 2020.

Eneko Agirre, Inaki Alegria, Xabier Arregi, Xabier Artola, Arantza Diaz de Ilarraza, Montse Maritxalar, Kepa Sarasola, and Miriam Urkia. Xuxen: A spelling checker/-corrector for basque based on two-level morphology. In Third Conference on Applied Natural Language Processing, pages 119–125, 1992.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In COLING 2018, 27th International Conference on Computational Linguistics, pages 1638–1649, 2018.

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), pages 54–59, 2019.

Izaskun Aldezabal, Olatz Ansa, Bertol Arrieta, Xabier Artola, Aitzol Ezeiza, G Hernández, and Mikel Lersundi. Edbl: A general lexical basis for the automatic processing of basque. In Proceedings of the IRCS Workshop on linguistic databases. IRCS Workshop on linguistic databases., 2006.

Dimitrios Alikaniotis and Vipul Raheja. The unreasonable effectiveness of transformer language models in grammatical error correction. arXiv preprint arXiv:1906.01733, 2019.

Samuel Bell, Helen Yannakoudakis, and Marek Rei. Context is key: Grammatical error detection with contextual word representations. arXiv preprint arXiv:1906.06593, 2019.

Zuhaitz Beloki, Xabier Saralegi, Klara Ceberio, and Ander Corral. Grammatical error correction for basque through a seq2seq neural architecture and synthetic examples. Procesamiento del Lenguaje Natural, 65:13–20, 2020.

Adriane Boyd. Using wikipedia edits in low resource grammatical error correction. In Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text, pages 79–84, 2018.

Christopher Bryant, Mariano Felice, and Edward Briscoe. Automatic annotation and evaluation of error types for grammatical error correction. Association for Computational Linguistics, 2017.

Christopher Bryant, Mariano Felice, Øistein E Andersen, and Ted Briscoe. The bea-2019 shared task on grammatical error correction. In Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 52–75, 2019.

Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. A neural grammatical error correction system built on better pre-training and sequential transfer learning. arXiv preprint arXiv:1907.01256, 2019.

Rachele De Felice and Stephen Pulman. A classifier-based approach to preposition and determiner error correction in l2 english. In Proceedings of the 22nd international conference on computational linguistics (Coling 2008), pages 169–176, 2008.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

Nava Ehsan and Heshaam Faili. Grammatical and context-sensitive error correction using a statistical machine translation framework. Software: Practice and Experience, 43(2): 187–206, 2013.

Nerea Ezeiza, Iñaki Alegria, Jose Mari Arriola, Rubén Urizar, and Itziar Aduriz. Combining stochastic and rule-based methods for disambiguation in agglutinative languages. In COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics, 1998.

Mariano Felice, Christopher Bryant, and Ted Briscoe. Automatic extraction of learner errors in esl sentences using linguistically enhanced alignments. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 825–835, 2016.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155, 2020.

Philip Gage. A new algorithm for data compression. C Users Journal, 12(2):23–38, 1994.

Tao Ge, Furu Wei, and Ming Zhou. Fluency boost learning and inference for neural grammatical error correction. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1055–1065, 2018.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 252–263, 2019.

Mans Hulden. Foma: a finite-state compiler and library. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pages 29–32. Association for Computational Linguistics, 2009.

Emi Izumi, Kiyotaka Uchimoto, Toyomi Saiga, Thepchai Supnithi, and Hitoshi Isahara. Automatic error detection in the japanese learners' english spoken data. In The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics, pages 145–148, 2003.

Eusko Jaurlaritza, Nafarroako Gobernua, and Euskararen Erakunde Publikoa. Vi. inkesta soziolinguistikoa. euskararen eremu osoa. linean],¡ https://www. irekia. euskadi. eus/uploads/attachments/9954/VI_INK_SOZLG-EH_eus. pdf, 1499236557, 2016.

Masahiro Kaneko and Mamoru Komachi. Multi-head multi-layer attention to deep language representations for grammatical error detection. Computación y Sistemas, 23(3): 883–891, 2019.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions, pages 177–180, 2007.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very deep transformers for neural machine translation. arXiv preprint arXiv:2008.07772, 2020.

Ariane Méndez. Euskarazko zuzentzaile gramatikal neuronala. 2020.

Daniel Naber et al. A rule-based style and grammar checker. 2003.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. Gector–grammatical error correction: tag, not rewrite. arXiv preprint arXiv:2005.12592, 2020.

Maite Oronoz. Euskarazko errore sintaktikoak detektatzeko eta zuzentzeko baliabideen garapena: datak, postposizio-lokuzioak eta komunztadura. PhD thesis, 2008.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Marek Rei and Helen Yannakoudakis. Compositional sequence labeling models for error detection in learner writing. arXiv preprint arXiv:1607.06153, 2016.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. A simple recipe for multilingual grammatical error correction. arXiv preprint arXiv:2106.03830, 2021.

Barry Schwartz. Google: Bert now used on almost every english query. Search Engine Land, https://searchengineland. com/google-bert-used-on-almost-every-english-query-342193 (October 15, 2020), 2020.

Felix Stahlberg and Shankar Kumar. Synthetic data generation for grammatical error correction with tagged corruption models. arXiv preprint arXiv:2105.13318, 2021.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. arXiv preprint arXiv:2104.06022, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, pages 38–45, 2020.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. arXiv preprint arXiv:2010.11934, 2020.

Zheng Yuan and Mariano Felice. Constrained grammatical error correction using statistical machine translation. In Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task, pages 52–61, 2013.

A Comparisons of analyses of correct and incorrect sentences

This appendix displays tables that contain all the comparisons performed during the error analysis phase described in Section 3.1. For each table, the left column shows a grammatically correct sentence and the analyses obtained when passing it as input to Eustagger; the right column contains an incorrect version of the other sentence and its analysis. In all pairs, for the particular analysis of the word that differs between the two sentences, the patterns that change from one analysis to the other are marked in red. In addition, if disambiguation issues (i.e. the analyser has not been able to completely disambiguate a word) occur in the words that vary between the sentences, they are underlined in orange. Some annotations made during the development stages of the error analysis and generation approaches are shown under the tables.

Ziur bihar jakingo dugula.

```
/<Ziur>/<HAS_MAI>/
   ("ziur" ADJ ARR w1,L-A-ADJ-ARR-10,lsfi1 @<IA)
/<bihar>/
   ("bihar" ADB ARR w2,L-A-ADB-ARR-10,lsfi2 @ADLG)
/<jakingo>/
   ("jakin" ADI SIN GERO DU w3,L-A-ADI-SIN-34,lsfi3 @-JADNAG)
/<dugula>/
   ("*edun" ADL A1 NR_HURA NK_GUK MOD/DENB w4,L-A-ADL-39,lsfi4
      @+JADLAG_MP_ADLG)
/<.>/<PUNT_PUNT>/
```

Ziur bihar jakiten dugula.

```
/<Ziur>/<HAS_MAI>/
   ("ziur" ADJ ARR w6,L-A-ADJ-ARR-10,lsfi5 @<IA)
/<bihar>/
   ("bihar" ADB ARR w7,L-A-ADB-ARR-10,lsfi6 @ADLG)
/<jakiten>/
   ("jakin" ADI SIN EZBU DU w8,L-A-ADI-SIN-37,lsfi7 @-JADNAG)
/<dugula>/
   ("*edun" ADL A1 NR_HURA NK_GUK MOD/DENB w9,L-A-ADL-39,lsfi8
      @+JADLAG_MP_ADLG)
/<.>/<PUNT_PUNT>/
```

- ➔ Jakingo → Geroaldia (future tense)
- ➔ Jakiten → Burutugabea (imperfective)

Gustura egingo nuke orain.

```
/<Gustura>/<HAS_MAI>/
   ("gustura" ADB ARR w11,L-A-ADB-ARR-11,lsfi9 @ADLG)
/<egingo>/
   ("egin" ADI SIN GERO DU w12,L-A-ADI-SIN-38,lsfi10 @-JADNAG)
/<nuke>/
   ("*edun" ADL B2 NR_HURA NK_NIK w13,L-A-ADL-41,lsfi11 @+JADLAG)
/<orain>/
   ("orain" ADB ARR w14,L-A-ADB-ARR-12,lsfi12 @ADLG)
/<.>/<PUNT_PUNT>/
```

Gustura egingo nuen orain.

```
/<Gustura>/<HAS_MAI>/
   ("gustura" ADB ARR w16,L-A-ADB-ARR-11,lsfi13 @ADLG)
/<egingo>/
   ("egin" ADI SIN GERO DU w17,L-A-ADI-SIN-38,lsfi14 @-JADNAG)
/<nuen>/
   ("*edun" ADL B1 NR_HURA NK_NIK w18,L-A-ADL-42,lsfi15 @+JADLAG)
   ("*edun" ADL B1 NR_HURA NK_NIK MOS w18,L-A-ADL-44,lsfi16
      @+JADLAG_MP_ADLG)
/<orain>/
   ("orain" ADB ARR w19,L-A-ADB-ARR-12,lsfi17 @ADLG)
/<.>/<PUNT_PUNT>/
```

- ➔ Nuke → Indik. baldintza (conditional indicative) → adb: nintzateke
- ➔ Nuen → Indikatibozko lehenaldia (past indicative → adb: nintzen

Gauza bat falta zait esateko.

```
/<Gauza>/<HAS_MAI>/
   ("Gauza" IZE LIB PLU- w21,L-A-IZE-LIB-3,lsfi18 @KM>)
/<bat>/
   ("bat" DET DZH NMGS w22,L-A-DET-DZH-3,lsfi19 @ID>)
/<falta>/
   ("falta" IZE ARR BIZ- ABS MG w23,L-A-IZE-ARR-72,lsfi21 @PRED)
   ("falta" IZE ARR BIZ- ABS NUMS MUGM w23,L-A-IZE-ARR-73,lsfi20 @PRED)
/<zait>/
   ("izan" ADT A1 NR_HURA NI_NIRI PNT w24,L-A-ADT-48,lsfi22 @-JADNAG)
/<esateko>/
   ("esateko" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-75,lsfi23 @PRED)
   ("esateko" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-75,lsfi24 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-77,lsfi25 @PRED)
   ("esate" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-77,lsfi26 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-80,lsfi27 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w25,L-A-IZE-ARR-80,lsfi28 @PRED)
/<.>/<PUNT_PUNT>/
```

Gauza bat faltatzen zait esateko.

```
/<Gauza>/<HAS_MAI>/
   ("gauza" IZE ARR BIZ- w27,L-A-IZE-ARR-68,lsfi29 @KM>)
/<bat>/
   ("bat" DET DZH NMGS ABS MG w28,L-A-DET-DZH-4,lsfi30 @SUBJ)
/<faltatzen>/
   ("faltatu" ADI SIN EZBU DA-DU w29,L-A-ADI-SIN-47,lsfi31 @-JADNAG)
/<zait>/
   ("izan" ADL A1 NR_HURA NI_NIRI w30,L-A-ADL-46,lsfi32 @+JADLAG)
/<esateko>/
   ("esateko" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-75,lsfi33 @PRED)
   ("esateko" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-75,lsfi34 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-77,lsfi35 @PRED)
   ("esate" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-77,lsfi36 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-80,lsfi37 @SUBJ)
   ("esate" IZE ARR BIZ- ABS MG w31,L-A-IZE-ARR-80,lsfi38 @PRED)
/<.>/<PUNT_PUNT>/
```

➔ We have not been able to obtain the needed disambiguation because the option of *"falta"* being a verb has been discarded by the tagger. We will try with the *"bururatzen"* verb in the next example to see if we get the disambiguation we need.

Gaur gauza bat bururatu zait.

```
/<Gaur>/<HAS_MAI>/
   ("gaur" ADB ARR w1,L-A-ADB-ARR-2,lsfi1 @ADLG)
/<gauza>/
   ("gauza" IZE ARR BIZ- w2,L-A-IZE-ARR-4,lsfi2 @KM>)
/<bat>/
   ("bat" DET DZH NMGS ABS MG w2,L-A-DET-DZH-4,lsfi2 @SUBJ)
/<bururatu>/
   ("bururatu" ADI SIN BURU DU-ZAIO w3,L-A-ADI-SIN-11,lsfi3 @-JADNAG)
/<zait>/
   ("izan" ADL A1 NR_HURA NI_NIRI w4,L-A-ADL-2,lsfi4 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

Gaur gauza bat bururatzen zait.

```
/<Gaur>/<HAS_MAI>/
   ("gaur" ADB ARR w1,L-A-ADB-ARR-2,lsfi1 @ADLG)
/<gauza>/
   ("gauza" IZE ARR BIZ- w2,L-A-IZE-ARR-4,lsfi2 @KM>)
/<bat>/
   ("bat" DET DZH NMGS ABS MG w7,L-A-DET-DZH-4,lsfi6 @SUBJ)
/<bururatzen>/
   ("bururatu" ADI SIN EZBU DU-ZAIO w8,L-A-ADI-SIN-15,lsfi7 @-JADNAG)
/<zait>/
   ("izan" ADL A1 NR_HURA NI_NIRI w9,L-A-ADL-2,lsfi8 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

Afaltzera gonbidatu ninduen.

```
/<Afaltzera>/<HAS_MAI>/
   ("afaltze" IZE ARR ALA NUMS MUGM w33,L-A-IZE-ARR-81,lsfi39 @ADLG)
/<gonbidatu>/
   ("gonbidatu" ADI SIN BURU DU w34,L-A-ADI-SIN-50,lsfi40 @-JADNAG)
/<ninduen>/
   ("*edun" ADL B1 NR_NI NK_HARK w35,L-A-ADL-47,lsfi41 @+JADLAG)
   ("*edun" ADL B1 NR_NI NK_HARK MOS w35,L-A-ADL-49,lsfi43
     @+JADLAG_MP_ADLG)
   ("*edun" ADL B1 NR_NI NK_HARK ERLT w35,L-A-ADL-50,lsfi42
     @+JADLAG_MP_IZLG>)
/<.>/<PUNT_PUNT>/
```

Afaltzera gonbidatu zidan.

```
/<Afaltzera>/<HAS_MAI>/
   ("afaltze" IZE ARR ALA NUMS MUGM w37,L-A-IZE-ARR-81,lsfi44 @ADLG)
/<gonbidatu>/
   ("gonbidatu" ADI SIN BURU DU w38,L-A-ADI-SIN-50,lsfi45 @-JADNAG)
/<zidan>/
   ("*edun" ADL B1 NR_HURA NI_NIRI NK_HARK w39,L-A-ADL-52,lsfi46
     @+JADLAG)
   ("*edun" ADL B1 NR_HURA NI_NIRI NK_HARK MOS w39,L-A-ADL-54,
     lsfi48 @+JADLAG_MP_ADLG)
   ("*edun" ADL B1 NR_HURA NI_NIRI NK_HARK ERLT w39,L-A-ADL-55,
     lsfi47 @+JADLAG_MP_IZLG>)
/<.>/<PUNT_PUNT>/
```

➔ Ninduen → Nor: ni (nork: hark)
➔ Zidan → Nor: hura, Nori: niri (nork: hark)

Jonek ez daki ezer.

```
/<Jonek>/<HAS_MAI>/
   ("Jone" IZE IZB PLU- ERG NUMS MUGM w63,L-A-IZE-IZB-5,lsfi76 @SUBJ)
   ("Jon" IZE IZB PLU- ERG NUMS MUGM w63,L-A-IZE-IZB-6,lsfi77 @SUBJ)
/<ez>/
   ("ez" ADB ARR w64,L-A-ADB-ARR-14,lsfi78 @ADLG)
/<daki>/
   ("jakin" ADT A1 NR_HURA NK_HARK PNT w65,L-A-ADT-69,lsfi79
     @+JADNAG)
/<ezer>/
   ("ezer" IOR IZGMGB ABS MG w66,L-A-IOR-IZGMGB-2,lsfi80 @OBJ)
   ("ezer" IOR IZGMGB ABS MG w66,L-A-IOR-IZGMGB-2,lsfi81 @SUBJ)
/<.>/<PUNT_PUNT>/
```

Jon ez daki ezer.

```
/<Jon>/<HAS_MAI>/
   ("Jon" IZE IZB PLU- ABS NUMS MUGM w68,L-A-IZE-IZB-8,lsfi82 @OBJ)
/<ez>/
   ("ez" PRT EGI w69,L-A-PRT-4,lsfi83 @PRT)
/<daki>/
   ("jakin" ADT A1 NR_HURA NK_HARK PNT w70,L-A-ADT-69,lsfi84
     @+JADNAG)
/<ezer>/
   ("ezer" IOR IZGMGB ABS MG w71,L-A-IOR-IZGMGB-2,lsfi85 @OBJ)
   ("ezer" IOR IZGMGB ABS MG w71,L-A-IOR-IZGMGB-2,lsfi86 @SUBJ)
/<.>/<PUNT_PUNT>/
```

➔ Jonek → Ergatiboa (ergative)
➔ Jon → Absolutiboa (absolutive)

Bidaiek atsedena hartzeko balio dute.

```
/<Bidaiek>/<HAS_MAI>/
    ("bidaia" IZE ARR BIZ- ERG NUMP MUGM w73,L-A-IZE-ARR-93,lsfi87 @SUBJ)
/<atsedena>/
   ("atseden" ADI SIN ABS NUMS MUGM PART w74,L-A-ADI-SIN-62,lsfi88
     @-JADNAG_MP_OBJ)
   ("atseden" ADI SIN ABS NUMS MUGM PART w74,L-A-ADI-SIN-62,lsfi89
     @-JADNAG_MP_SUBJ)
/<hartzeko>/
   ("hartze" IZE ARR GEL NUMS MUGM w75,L-A-IZE-ARR-97,lsfi91 @IZLG>)
   ("hartz" IZE ARR BIZ+ GEL NUMS MUGM w75,L-A-IZE-ARR-102,lsfi90 @IZLG>)
/<balio>/
   ("baliotu" ADI SIN DU ADOIN LEX w76,L-A-ADI-SIN-66,lsfi92 @-JADNAG)
/<dute>/
   ("ukan" ADT A1 NR_HURA NK_HAIEK-K PNT w77,L-A-ADT-73,lsfi93
     @+JADNAG)
/<.>/<PUNT_PUNT>/
```

Bidaiak atsedena hartzeko balio dute.

```
/<Bidaiak>/<HAS_MAI>/
   ("bidaia" IZE ARR BIZ- ABS NUMP MUGM w79,L-A-IZE-ARR-112,lsfi94 @SUBJ)
/<atsedena>/
   ("atseden" ADI SIN ABS NUMS MUGM PART w80,L-A-ADI-SIN-62,lsfi95
     @-JADNAG_MP_OBJ)
   ("atseden" ADI SIN ABS NUMS MUGM PART w80,L-A-ADI-SIN-62,lsfi96
     @-JADNAG_MP_SUBJ)
/<hartzeko>/
   ("hartze" IZE ARR GEL NUMS MUGM w81,L-A-IZE-ARR-97,lsfi98 @IZLG>)
   ("hartz" IZE ARR BIZ+ GEL NUMS MUGM w81,L-A-IZE-ARR-102,lsfi97 @IZLG>)
/<balio>/
   ("baliotu" ADI SIN DU ADOIN LEX w82,L-A-ADI-SIN-66,lsfi99 @-JADNAG)
/<dute>/
   ("ukan" ADT A1 NR_HURA NK_HAIEK-K PNT w83,L-A-ADT-73,lsfi100
     @+JADNAG)
/<.>/<PUNT_PUNT>/
```

➔ Bidaiek → Ergatiboa
➔ Bidaiak → Absolutiboa

Jende askok uste du.

```
/<Jende>/<HAS_MAI>/
   ("jende" IZE ARR w85,L-A-IZE-ARR-114,lsfi101 @KM>)
/<askok>/
   ("asko" DET DZG MG ERG MG w86,L-A-DET-DZG-4,lsfi102 @SUBJ)
/<uste>/
   ("uste" IZE ARR BIZ- ABS MG w87,L-A-IZE-ARR-119,lsfi103 @OBJ)
/<du>/
   ("ukan" ADT A1 NR_HURA NK_HARK PNT w88,L-A-ADT-74,lsfi104
     @-JADNAG)
/<.>/<PUNT_PUNT>/
```

Jende askok uste dute.

```
/<Jende>/<HAS_MAI>/
   ("jende" IZE ARR w90,L-A-IZE-ARR-114,lsfi105 @KM>)
/<askok>/
   ("asko" DET DZG MG ERG MG w91,L-A-DET-DZG-4,lsfi106 @SUBJ)
/<uste>/
   ("uste" IZE ARR BIZ- ABS MG w92,L-A-IZE-ARR-119,lsfi107 @OBJ)
/<dute>/
   ("ukan" ADT A1 NR_HURA NK_HAIEK-K PNT w93,L-A-ADT-73,lsfi108
     @-JADNAG)
/<.>/<PUNT_PUNT>/
```

➔ Du → Nork: hark
➔ Dute → Nork: haiek

Atzo kalean ikusi zintudan.

```
/<Atzo>/<HAS_MAI>/
   ("atzo" ADB ARR w41,L-A-ADB-ARR-13,lsfi49 @ADLG)
/<kalean>/
   ("kale" IZE ARR BIZ- INE NUMS MUGM w42,L-A-IZE-ARR-82,lsfi50 @ADLG)
/<ikusi>/
   ("ikusi" ADI SIN BURU DU w43,L-A-ADI-SIN-53,lsfi51 @-JADNAG)
/<zintudan>/
   ("*edun" ADL B1 NR_ZU NK_NIK w44,L-A-ADL-56,lsfi52 @+JADLAG)
   ("*edun" ADL B1 NR_ZU NK_NIK MOS w44,L-A-ADL-58,lsfi54
     @+JADLAG_MP_ADLG)
   ("*edun" ADL B1 NR_ZU NK_NIK ERLT w44,L-A-ADL-59,lsfi53
     @+JADLAG_MP_IZLG>)
/<.>/<PUNT_PUNT>/
```

Atzo kalean ikusi nizun.

```
/<Atzo>/<HAS_MAI>/
   ("atzo" ADB ARR w46,L-A-ADB-ARR-13,lsfi55 @ADLG)
/<kalean>/
   ("kale" IZE ARR BIZ- INE NUMS MUGM w47,L-A-IZE-ARR-82,lsfi56 @ADLG)
/<ikusi>/
   ("ikusi" ADI SIN BURU DU w48,L-A-ADI-SIN-53,lsfi57 @-JADNAG)
 /<nizun>/
   ("*edun" ADL B1 NR_HURA NI_ZURI NK_NIK w49,L-A-ADL-60,lsfi58
     @+JADLAG)
   ("*edun" ADL B1 NR_HURA NI_ZURI NK_NIK MOS w49,L-A-ADL-62,lsfi60
     @+JADLAG_MP_ADLG)
   ("*edun" ADL B1 NR_HURA NI_ZURI NK_NIK  ERLT w49,L-A-ADL-63,lsfi59
     @+JADLAG_MP_IZLG>)
/<.>/<PUNT_PUNT>/
```

➔ Zintudan → Nor: zu (nork: nik)
➔ Nizun → Nor: hura, nori: zuri (nork: nik)

- Note: If we use the command “up zintudan” in the FST, several outputs are obtained. Still, for all those choices, we can get “nizun” in the downwards direction by adding nori and modifying nor.

Paisaia asko gustatzen zait.

```
/<Paisaia>/<HAS_MAI>/
   ("paisaia" IZE ARR w41,L-A-IZE-ARR-67,lsfi41 @KM>)
/<asko>/
   ("asko" DET DZG MG ABS MG w42,L-A-DET-DZG-4,lsfi42 @SUBJ)
/<guztatzen>/
   ("gustatu" ADI SIN EZBU ZAIO w43,L-T1-ADI-SIN-61,lsfi43 @-JADNAG)
/<zait>/
   ("izan" ADL A1 NR_HURA NI_NIRI w44,L-A-ADL-45,lsfi44 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

Paisaia asko gustatzen nau.

```
/<Paisaia>/<HAS_MAI>/
   ("paisaia" IZE ARR w46,L-A-IZE-ARR-67,lsfi45 @KM>)
/<asko>/
   ("asko" DET DZG MG ABS MG w47,L-A-DET-DZG-4,lsfi46 @OBJ)
/<gustatzen>/
   ("gustatu" ADI SIN EZBU ZAIO w48,L-A-ADI-SIN-66,lsfi47 @-JADNAG)
/<nau>/
  ("*edun" ADL A1 NR_NI NK_HARK w49,L-A-ADL-46,lsfi48 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

➔ Zait → nor: hura. nori: niri
➔ Nau → nor: ni, nork: hark

Utzi behar diogu negar egiteari.

```
/<Utzi>/<HAS_MAI>/
   ("utzi" ADI SIN DU PART w51,L-A-ADI-SIN-55,lsfi61 @-JADNAG)
   ("utzi" ADI SIN ABS MG PART w51,L-A-ADI-SIN-57,lsfi62 @-JADNAG_MP_OBJ)
   ("utzi" ADI SIN ABS MG PART w51,L-A-ADI-SIN-57,lsfi63 @-JADNAG_MP_SUBJ)
/<behar>/
   ("behar" IZE ARR BIZ- ABS MG w52,L-A-IZE-ARR-86,lsfi64 @OBJ)
/<diogu>/
   ("ukan" ADT A1 NR_HURA NI_HARI NK_GUK PNT w53,L-A-ADT-66,lsfi65
    @-JADNAG)
   ("esan" ADT A1 NR_HURA NK_GUK PNT w53,L-A-ADT-67,lsfi66 @-JADNAG)
/<negar>/
   ("negar" IZE ARR BIZ- ABS MG w54,L-A-IZE-ARR-88,lsfi67 @OBJ)
/<egiteari>/
   ("egin" ADI SIN DAT NUMS MUGM ADIZE w55,L-A-ADI-SIN-60,lsfi68
    @-JADNAG_MP_ZOBJ)
/<.>/<PUNT_PUNT>/
```

Utzi behar dugu negar egitea.

```
/<Utzi>/<HAS_MAI>/
   ("utzi" ADI SIN DU PART w57,L-A-ADI-SIN-55,lsfi69 @-JADNAG)
   ("utzi" ADI SIN ABS MG PART w57,L-A-ADI-SIN-57,lsfi70 @-JADNAG_MP_OBJ)
   ("utzi" ADI SIN ABS MG PART w57,L-A-ADI-SIN-57,lsfi71 @-JADNAG_MP_SUBJ)
/<behar>/
   ("behar" IZE ARR BIZ- ABS MG w58,L-A-IZE-ARR-86,lsfi72 @OBJ)
/<dugu>/
   ("ukan" ADT A1 NR_HURA NK_GUK PNT w59,L-A-ADT-68,lsfi73 @-JADNAG)
/<negar>/
   ("negar" IZE ARR BIZ- w60,L-A-IZE-ARR-87,lsfi74 @KM>)
/<egitea>/
   ("egin" ADI SIN ABS ADIZE KONPL w61,L-A-ADI-SIN-61,lsfi75
    @-JADNAG_MP_OBJ)
/<.>/<PUNT_PUNT>/
```

- ➔ Egiteari → Datiboa (nori), singularra, mugatua (dative, singular, definite)
- ➔ Egitea → Absolutiboa (nor), konpletiboa (absolutive, completive)

Azkenaldian asko argaldu du.

```
/<Azkenaldian>/<HAS_MAI>/
   ("azkenaldi" IZE ARR BIZ- INE NUMS MUGM w31,L-A-IZE-ARR-66,lsfi33
    @ADLG)
/<asko>/
   ("asko" ADB ARR GRAD w32,L-A-ADB-ARR-8,lsfi34 @ADLG)
/<argaldu>/
   ("argaldu" ADI SIN BURU DA-DU w33,L-A-ADI-SIN-59,lsfi35 @-JADNAG)
/<du>/
   ("*edun" ADL A1 NR_HURA NK_HARK w34,L-A-ADL-43,lsfi36 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

Azkenaldian asko argaldu da.

```
/<Azkenaldian>/<HAS_MAI>/
   ("azkenaldi" IZE ARR BIZ- INE NUMS MUGM w36,L-A-IZE-ARR-66,lsfi37
    @ADLG)
/<asko>/
   ("asko" ADB ARR GRAD w37,L-A-ADB-ARR-8,lsfi38 @ADLG)
/<argaldu>/
   ("argaldu" ADI SIN BURU DA-DU w38,L-A-ADI-SIN-59,lsfi39 @-JADNAG)
/<da>/
   ("izan" ADL A1 NR_HURA w39,L-A-ADL-44,lsfi40 @+JADLAG)
/<.>/<PUNT_PUNT>/
```

- ➔ Du → nork: hark (nor: hura)
- ➔ Da → X (nor: hura)

Aholkuak ematen ari zaigu.

```
/<Aholkuak>/<HAS_MAI>/
   ("aholku" IZE ARR BIZ- ABS NUMP MUGM w1,L-A-IZE-ARR-5,lsfi1 @PRED)
/<ematen>/
   ("eman" ADI SIN INE ADIZE KONPL w2,L-A-ADI-SIN-5,lsfi2 @-JADNAG_MP_OBJ)
/<ari>/
   ("ari" HAOS w3,L-A-HAOS-2,lsfi3 @HAOS)
/<zaigu>/
   ("izan" ADT A1 NR_HURA NI_GURI PNT w4,L-A-ADT-2,lsfi4 @+JADNAG)
/<.>/<PUNT_PUNT>/
```

Aholkuak ematen ari digu.

```
/<Aholkuak>/<HAS_MAI>/
   ("aholku" IZE ARR BIZ- ABS NUMP MUGM w1,L-A-IZE-ARR-5,lsfi1 @SUBJ)
/<ematen>/
   ("eman" ADI SIN INE ADIZE KONPL w2,L-A-ADI-SIN-5,lsfi2 @-JADNAG_MP_OBJ)
/<ari>/
   ("ari" HAOS w3,L-A-HAOS-2,lsfi3 @HAOS)
/<digu>/
   ("ukan" ADT A1 NR_HURA NI_GURI NK_HARK PNT w4,L-A-ADT-2,lsfi4
    @+JADNAG)
/<.>/<PUNT_PUNT>/
```

➔ Zaigu → X (nor: hura, nori: guri)
➔ Digu → nork: hark (nor: hura, nori: guri)

Ez dut uste hori egia denik.

```
/<Ez>/<HAS_MAI>/
   ("ez" ADB ARR w95,L-A-ADB-ARR-15,lsfi109 @ADLG)
/<dut>/
   ("ukan" ADT A1 NR_HURA NK_NIK PNT w96,L-A-ADT-75,lsfi110 @+JADNAG)
/<uste>/
   ("uste" IZE ARR BIZ- w97,L-A-IZE-ARR-118,lsfi111 @KM>)
/<hori>/
   ("hori" DET ERKARR ABS NUMS MUGM w98,L-A-DET-ERKARR-2,lsfi112
    @SUBJ)
/<egia>/
   ("egia" IZE ARR BIZ- ABS NUMS MUGM w99,L-A-IZE-ARR-124,lsfi113 @PRED)
/<denik>/
   ("izan" ADT A1 NR_HURA PNT KONPL w100,L-A-ADT-76,lsfi114
    @+JADNAG_MP_SUBJ)
   ("izan" ADT A1 NR_HURA PNT KONPL w100,L-A-ADT-76,lsfi115
    @+JADNAG_MP_OBJ)
/<.>/<PUNT_PUNT>/
```

Ez dut uste hori egia dela.

```
/<Ez>/<HAS_MAI>/
   ("ez" ADB ARR w102,L-A-ADB-ARR-15,lsfi116 @ADLG)
/<dut>/
   ("ukan" ADT A1 NR_HURA NK_NIK PNT w103,L-A-ADT-75,lsfi117 @+JADNAG)
/<uste>/
   ("uste" IZE ARR BIZ- w104,L-A-IZE-ARR-118,lsfi118 @KM>)
/<hori>/
   ("hori" DET ERKARR ABS NUMS MUGM w105,L-A-DET-ERKARR-2,lsfi119
    @SUBJ)
/<egia>/
   ("egia" IZE ARR BIZ- ABS NUMS MUGM w106,L-A-IZE-ARR-124,lsfi120 @PRED)
/<dela>/
   ("izan" ADT A1 NR_HURA PNT MOD/DENB w107,L-A-ADT-78,lsfi121
    @+JADNAG_MP_ADLG)
/<.>/<PUNT_PUNT>/
```

➔ Denik →Konpletiboa (completive)
➔ Dela → Moduzkoa/Denborazkoa (adverb type: manner/time)

Nire ustez, hori horrela da.

```
/<Nire>/<HAS_MAI>/
   ("ni" IOR PERARR NI GEN NUMS MUGM w109,L-A-IOR-PERARR-3,lsfi122 @IZLG>)
/<ustez>/
   ("uste" IZE ARR BIZ- INS MG w110,L-A-IZE-ARR-125,lsfi123 @ADLG)
/<,>/<PUNT_KOMA>/
/<hori>/
   ("hori" DET ERKARR ABS NUMS MUGM w112,L-A-DET-ERKARR-2,lsfi124 @SUBJ)
/<horrela>/
   ("horrela" ADB ARR w113,L-A-ADB-ARR-17,lsfi125 @ADLG)
/<da>/
   ("izan" ADT A1 NR_HURA PNT w114,L-A-ADT-80,lsfi126 @+JADNAG)
/<.>/<PUNT_PUNT>/
```

Nire ustez, hori horrela dela.

```
/<Nire>/<HAS_MAI>/
   ("ni" IOR PERARR NI GEN NUMS MUGM w116,L-A-IOR-PERARR-3,lsfi127 @IZLG>)
/<ustez>/
   ("uste" IZE ARR BIZ- INS MG w117,L-A-IZE-ARR-125,lsfi128 @ADLG)
/<,>/<PUNT_KOMA>/
/<hori>/
   ("hori" DET ERKARR ABS NUMS MUGM w119,L-A-DET-ERKARR-2,lsfi129 @SUBJ)
/<horrela>/
   ("horrela" ADB ARR w120,L-A-ADB-ARR-17,lsfi130 @ADLG)
/<dela>/
   ("izan" ADT A1 NR_HURA PNT MOD/DENB w121,L-A-ADT-78,lsfi131
      @+JADNAG_MP_ADLG)
/<.>/<PUNT_PUNT>/
```

➔ Da → X

➔ Dela → Moduzkoa/Denborazkoa (adverb type: manner/time)

Badago beste kutsadura bat dena nuklearra.

```
/<Badago>/<HAS_MAI>/
   ("egon" ADT A1 NR_HURA PNT BALD w123,L-A-ADT-82,lsfi132 @+JADNAG_MP_ADLG)
   ("egon" ADT A1 NR_HURA PNT EGI w123,L-A-ADT-83,lsfi133 @+JADNAG)
/<beste>/
   ("beste" DET DZG w124,L-A-DET-DZG-5,lsfi134 @ID>)
/<kutsadura>/
   ("kutsadura" IZE ARR BIZ- w125,L-A-IZE-ARR-127,lsfi135 @KM>)
/<bat>/
   ("bat" DET DZH NMGS ABS MG w126,L-A-DET-DZH-4,lsfi136 @SUBJ)
   ("bat" DET DZH NMGS ABS MG w126,L-A-DET-DZH-4,lsfi137 @PRED)
/<dena>/
   ("dena" DET ORO ABS NUMS MUGM w127,L-A-DET-ORO-2,lsfi138 @SUBJ)
   ("dena" DET ORO ABS NUMS MUGM w127,L-A-DET-ORO-2,lsfi139 @PRED)
/<nuklearra>/
   ("nuklear" ADJ ARR ABS NUMS MUGM w128,L-A-ADJ-ARR-18,lsfi140 @SUBJ)
   ("nuklear" ADJ ARR ABS NUMS MUGM w128,L-A-ADJ-ARR-18,lsfi141 @PRED)
/<.>/<PUNT_PUNT>/
```

Badago beste kutsadura bat dela nuklearra.

```
/<Badago>/<HAS_MAI>/
   ("egon" ADT A1 NR_HURA PNT BALD w130,L-A-ADT-82,lsfi142
      @+JADNAG_MP_ADLG)
   ("egon" ADT A1 NR_HURA PNT EGI w130,L-A-ADT-83,lsfi143 @+JADNAG)
/<beste>/
   ("beste" DET DZG w131,L-A-DET-DZG-5,lsfi144 @ID>)
/<kutsadura>/
   ("kutsadura" IZE ARR BIZ- w132,L-A-IZE-ARR-127,lsfi145 @KM>)
/<bat>/
   ("bat" DET DZH NMGS ABS MG w133,L-A-DET-DZH-4,lsfi146 @PRED)
/<dela>/
   ("izan" ADT A1 NR_HURA PNT MOD/DENB w134,L-A-ADT-78,lsfi147
      @+JADNAG_MP_ADLG)
/<nuklearra>/
   ("nuklear" ADJ ARR ABS NUMS MUGM w135,L-A-ADJ-ARR-18,lsfi148 @SUBJ)
   ("nuklear" ADJ ARR ABS NUMS MUGM w135,L-A-ADJ-ARR-18,lsfi149 @PRED)
/<.>/<PUNT_PUNT>/
```

➔ We have not been able to find an example where the word “dena” is not disambiguated as a synonym for “all”.

B Grammar Error Detection Evaluation

B.1 Considering error types

This section shows F1 micro, F1 macro and Accuracy scores of models trained using different embedding combinations and different corpora sizes, evaluated against three different test sets. The evaluation considers error types, that is, the capability of models to detect tokens that make a sentence incorrect AND determine what type of error they create is measured.

Train: 3250 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.819	0.6849	0.7107	0.702	0.6027	0.5513	0.3221	0.3325	0.1933
Transformer + Char	**0.8462**	**0.7396**	**0.7521**	0.7154	0.6371	0.5677	0.3372	0.3637	0.2042
Transformer + Word	0.8438	0.7389	0.75	**0.7714**	**0.6796**	**0.6429**	**0.4615**	**0.4037**	**0.3034**
Transformer + Char + Word	0.8205	0.6932	0.7207	0.7273	0.6103	0.5882	0.4156	0.3874	0.2658
Transformer + Char + Word + Flair	0.8406	0.665	0.7436	0.7373	0.597	0.5959	0.3686	0.3602	0.2277

Table 24: GED with error types. Setting1. Training corpus contains 3.250 grammatically incorrect sentences.

Train: 5160 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
1910 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.8108	**0.7014**	0.7009	0.75	**0.6452**	0.6148	0.566	**0.4823**	0.4011
Transformer + Char	0.8066	0.6079	0.6887	0.7487	0.5513	0.6083	0.5794	0.4442	0.4124
Transformer + Word	0.8045	0.5869	0.6857	0.7423	0.526	0.6	**0.5878**	0.4387	**0.4211**
Transformer + Char + Word	**0.8342**	0.631	**0.7411**	**0.7685**	0.576	**0.6434**	0.5108	0.4178	0.3487
Transformer + Char + Word + Flair	0.7876	0.588	0.6726	0.707	0.5305	0.563	0.4097	0.3676	0.2612

Table 25: GED with error types. Setting2. Training corpus contains 1.910 correct sentences and 3.250 grammatically incorrect sentences.

Train: 8415 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
5165 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.8197	0.5907	0.7075	0.7614	0.5346	0.625	0.6198	0.4507	0.4545
Transformer + Char	0.8432	0.724	0.7429	0.7959	0.6755	0.6724	**0.6446**	0.5158	**0.4815**
Transformer + Word	**0.8737**	**0.7597**	**0.783**	**0.8137**	**0.7039**	**0.6917**	0.6434	**0.5587**	0.477
Transformer + Char + Word	0.8415	0.7347	0.7333	0.77	0.6655	0.6311	0.59	0.5363	0.4208
Transformer + Char + Word + Flair	0.8442	0.653	0.7434	0.7778	0.5978	0.6462	0.5793	0.4682	0.4118

Table 26: GED with error types. Setting3. Training corpus contains 5.165 correct sentences and 3.250 grammatically incorrect sentences.

Train: 18250 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
15000 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.7381	0.5555	0.5905	0.7006	0.5112	0.5439	0.6526	0.4778	0.4882
Transformer + Character	0.764	0.5674	0.6355	0.0095	0.0088	0.0049	0.0025	0.0019	0.0013
Transformer + Word	0.7349	0.5118	0.5865	0.7052	0.4932	0.5495	0.6354	0.4337	0.4692
Transformer + Character + Word	**0.7886**	**0.5859**	**0.6635**	**0.7541**	**0.5474**	**0.6161**	**0.6866**	**0.505**	**0.5308**
Transformer + Character + Word + Flair	0.7485	0.5423	0.6095	0.7072	0.5081	0.5565	0.6305	0.4516	0.4672

Table 27: GED with error types. Setting4. Training corpus contains 15.000 correct sentences and 3.250 grammatically incorrect sentences.

Train: 33250 sentences	Test: 100 sentences			Test: 185 sentences			Test: 925 sentences		
30000 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.6994	0.5126	0.5481	0.6706	0.4817	0.5135	0.6404	0.4627	0.479
Transformer + Char	**0.7602**	0.5463	**0.625**	0.0022	0.0021	0.0011	0.0005	0.0007	0.0003
Transformer + Word	0.7558	0.5864	**0.625**	**0.7182**	0.5591	**0.5752**	0.6404	0.4636	0.4815
Transformer + Char + Word	0.7456	**0.5925**	0.6058	0.7119	**0.5723**	0.5625	**0.6597**	**0.5068**	**0.5**
Transformer + Char + Word + Flair	0.6951	0.4868	0.5481	0.6514	0.4519	0.4957	0.6064	0.4214	0.4453

Table 28: GED with error types. Setting5. Training corpus contains 30.000 correct sentences and 3.250 grammatically incorrect sentences.

B.2 Without considering error types

This section shows True Positive, False Positive and False Negative counts and F1 score of models trained using different embedding combinations and different corpora sizes, evaluated against three different test sets. The evaluation ONLY considers the capability of models to detect tokens that make a sentence incorrect. The error type the models assign to each token is not taken into account.

Train: 3250 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer	89	17	15	0.8476190476190476	89	52	15	0.726530612244898	89	341	15	0.3333333333333333
Transformer + Char	91	13	13	0.875	91	51	13	0.7398373983739838	91	327	13	0.3486590038314176
Transformer + Word	84	2	20	0.875	84	22	20	0.7999999999999999	84	163	20	0.4786324786324865
Transformer + Char + Word	84	7	20	0.8615384615384616	84	32	20	0.7636363636363636	84	197	20	0.4363636363636364
Transformer + Char + Word + Flair	90	13	14	0.8695652173913043	90	42	14	0.7627118644067796	90	278	14	0.3813559322033898

Table 29: GED without error types. Setting1. Training corpus contains 3.250 grammatically incorrect sentences.

Train: 5160 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
1910 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer	78	3	26	0.8432432432432433	78	18	26	0.78	78	83	26	0.5886792452830188
Transformer + Char	75	2	29	0.8287292817679558	75	16	29	0.7692307692307693	75	73	29	0.5952380952380953
Transformer + Word	74	1	30	0.8268156424581006	74	16	30	0.7628865979381444	74	67	30	0.6040816326530611
Transformer + Char + Word	87	8	17	0.8743718592964824	87	25	17	0.8055555555555556	87	134	17	0.5353846153846153
Transformer + Char + Word + Flair	80	9	24	0.8290155440414508	80	31	24	0.7441860465116278	80	187	24	0.431266846361186

Table 30: GED without error types. Setting2. Training corpus contains 1.910 correct sentences and 3.250 grammatically incorrect sentences.

Train: 8415 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
5165 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer	77	2	27	0.8415300546448088	77	16	27	0.7817258883248731	77	61	27	0.6363636363636365
Transformer + Char	80	1	24	0.864864864864865	80	12	24	0.8163265306122449	80	58	24	0.6611570247933884
Transformer + Word	84	2	20	0.8842105263157894	84	16	20	0.8235294117647058	84	70	20	0.6511627906976744
Transformer + Char + Word	78	1	26	0.8524590163934427	78	18	26	0.78	78	79	26	0.5977011494252874
Transformer + Char + Word + Flair	86	9	18	0.8643216080402011	86	26	18	0.7962962962962962	86	100	18	0.593103448275862

Table 31: GED without error types. Setting3. Training corpus contains 5.165 correct sentences and 3.250 grammatically incorrect sentences.

Train: 18250 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
15000 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer	63	1	41	0.7499999999999999	63	10	41	0.7118644067796609	63	23	41	0.663157894736842
Transformer + Char	-	-	-	-	-	-	-	-	-	-	-	-
Transformer + Word	62	0	42	0.7469879518072289	62	7	42	0.7167630057803468	62	26	42	0.6458333333333334
Transformer + Char + Word	71	0	33	0.8114285714285714	71	8	33	0.7759562841530053	71	26	33	0.7064676616915424
Transformer + Char + Word + Flair	66	1	38	0.7719298245614035	66	11	38	0.729281767955801	66	33	38	0.6502463054187192

Table 32: GED without error types. Setting4. Training corpus contains 15.000 correct sentences and 3.250 grammatically incorrect sentences.

Train: 33250 sentences	Test: 100 sentences				Test: 185 sentences				Test: 925 sentences			
30000 correct + 3250 with errors	100 with errors				85 correct + 100 with errors				825 correct + 100 with errors			
Embeddings	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**	**TP**	**FP**	**FN**	**F1**
Transformer	59	0	45	0.7239263803680982	59	7	45	0.6941176470588236	59	15	45	0.6629213483146067
Transformer + Char	-	-	-	-	-	-	-	-	-	-	-	-
Transformer + Word	68	0	36	0.7906976744186047	68	9	36	0.7513812154696132	68	31	36	0.6699507389162562
Transformer + Char + Word	65	0	39	0.7692307692307693	65	8	39	0.7344632768361581	65	22	39	0.6806282722513088
Transformer + Char + Word + Flair	60	0	44	0.7317073170731707	60	11	44	0.6857142857142857	60	24	44	0.6382978723404256

Table 33: GED without error types. Setting5. Training corpus contains 30.000 correct sentences and 3.250 grammatically incorrect sentences.

B.3 Manually revised test set

This section shows F1 micro, F1 macro and Accuracy scores of models trained using different embedding combinations and different corpora sizes, evaluated against three MANUALLY REVISED test sets. The evaluation considers error types, that is, the capability of models to detect tokens that make a sentence incorrect AND determine what type of error they create is measured.

Train: 3250 sentences	Test: 100 sentences			Test: 185 sentences			Test: 925 sentences		
3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.8612	0.6659	0.9842	0.7377	0.5938	0.98	0.3377	0.3387	0.9764
Transformer + Character	**0.9048**	**0.7257**	**0.9897**	0.7661	0.5856	0.982	0.3626	0.3696	0.9777
Transformer + Word	0.8776	0.6808	0.9879	**0.8037**	0.6453	**0.9875**	**0.4845**	0.4046	**0.9879**
Transformer + Character + Word	0.8731	0.6767	0.986	0.7748	0.5702	0.9843	0.4444	**0.4157**	0.9856
Transformer + Character + Word + Flair	0.8942	0.7121	0.9879	0.7848	**0.6464**	0.9839	0.3932	0.3924	0.9808

Table 34: GED with error types. Setting1. Training corpus contains 3.250 grammatically incorrect sentences. Evaluation with respect to manually revised test sets.

Train: 5160 sentences	Test: 100 sentences			Test: 185 sentences			Test: 925 sentences		
1910 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	**0.8601**	0.6561	**0.9854**	**0.7981**	**0.622**	**0.9872**	**0.6081**	**0.4455**	0.993
Transformer + Character	0.828	0.6014	0.9818	0.77	0.5706	0.9856	0.5992	0.4232	**0.9932**
Transformer + Word	0.8021	0.5763	0.7009	0.7426	0.5424	0.6148	0.5929	0.4153	0.4335
Transformer + Character + Word	0.86	0.6514	0.7818	0.7926	0.6185	0.6772	0.5276	0.4056	0.3644
Transformer + Character + Word + Flair	0.8458	**0.6562**	0.983	0.7623	0.6034	0.9836	0.4485	0.3819	0.9861

Table 35: GED with error types. Setting2. Training corpus contains 1.910 correct sentences and 3.250 grammatically incorrect sentences. Evaluation with respect to manually revised test sets.

Train: 8415 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
5165 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.8085	0.5594	0.9812	0.7525	0.5363	0.9852	0.6154	0.4036	0.9939
Transformer + Character	0.8513	0.6342	0.9854	0.8058	0.6116	**0.9885**	**0.6587**	0.4742	**0.9945**
Transformer + Word	0.8776	0.6752	0.9867	0.819	0.6423	0.9882	0.6515	**0.4902**	0.9939
Transformer + Character + Word	0.8449	0.6403	0.9836	0.7745	0.6011	0.9856	0.5962	0.4524	0.9929
Transformer + Character + Word + Flair	**0.8966**	**0.7012**	**0.9885**	**0.8273**	**0.6655**	0.9882	0.619	0.4734	0.9926

Table 36: GED with error types. Setting3. Training corpus contains 5.165 correct sentences and 3.250 grammatically incorrect sentences. Evaluation with respect to manually revised test sets.

Train: 18250 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
15000 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.7841	0.556	0.6571	0.7459	0.5372	0.6053	0.697	**0.5108**	0.5433
Transformer + Character	0.0133	0.0099	0.0069	0.0089	0.0065	0.0046	0.0025	0.0018	0.0012
Transformer + Word	0.7978	0.6252	0.6762	0.7676	0.5482	0.6339	0.6961	0.4807	0.542
Transformer + Character + Word	**0.8242**	0.6073	**0.7143**	**0.7895**	**0.5943**	**0.6637**	**0.7212**	0.5021	**0.5725**
Transformer + Character + Word + Flair	0.7956	**0.6267**	0.6792	0.7539	0.5463	0.6207	0.6761	0.4888	0.5217

Table 37: GED with error types. Setting4. Training corpus contains 15.000 correct sentences and 3.250 grammatically incorrect sentences. Evaluation with respect to manually revised test sets.

Train: 33250 sentences	**Test: 100 sentences**			**Test: 185 sentences**			**Test: 925 sentences**		
30000 correct + 3250 with errors	100 with errors			85 correct + 100 with errors			825 correct + 100 with errors		
Embeddings	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**	**F1 (micro)**	**F1 (macro)**	**Accuracy**
Transformer	0.7841	0.5676	0.6571	0.7541	**0.5577**	0.6161	**0.7225**	**0.5423**	**0.575**
Transformer + Character	0.7821	0.5568	0.6667	0.7447	0.5409	0.614	0.6796	0.4536	0.5303
Transformer + Word	0.7912	0.5301	**0.6857**	0.7539	0.5143	0.6316	0.6761	0.4093	0.5294
Transformer + Character + Word	**0.809**	**0.6291**	**0.6857**	**0.7742**	0.5498	**0.6372**	0.72	0.465	0.5669
Transformer + Character + Word + Flair	0.7889	0.603	0.6762	0.7435	0.5211	0.6121	0.6961	0.489	0.5504

Table 38: GED with error types. Setting5. Training corpus contains 30.000 correct sentences and 3.250 grammatically incorrect sentences. Evaluation with respect to manually revised test sets.

C Translations

We provide the English translations of the sentences in Basque used throughout this document. Note that only those examples that are grammatically correct have been translated.

Basque	**English**
Joxan Goikoetxea ariko da Lasarte-Oriako Akordeoi Jaialdian	Joxan Goikoetxea will play in the Accordion Festival in Lasarte-Oria
Nire ustez, hori horrela da	I think that is right
Ziur bihar jakingo dugula	I am sure we will know tomorrow
Gauza bat falta zait esateko	I have one thing left to say
Gaur gauza bat bururatu zait	Today I have had an idea
etxean	at home
jaurlaritzak gaur hasiko du urte politikoa, donostian	the government will start the political year today, in donostia
Gustura egingo nuke	I would gladly do it
Zerbait falta zait esateko	I have something left to say
Afaltzera gonbidatu ninduen	He/She invited me to have dinner
Afaltzera gonbidatzen ninduen	He/She used to invite me to have dinner
Nik ez dut nahi	I do not want to / I do not want it
langileak egin du	the worker has done it
langileak jatorrak dira	the workers are nice
Ez dut uste hori egia denik	I do not think that is true
Badago beste aukera bat hobea dena	A better choice exists
Atzo ikusi zintudan	I saw you yesterday
Azkenaldian asko argaldu du	He/She has lost a lot of weight lately
Aholku kontrajarriak ematen ari zaigu	He/She is giving us contradictory advice
Paisaia asko gustatzen zait	I really like the view
ikus-entzunezko edukien ekoizpenari	to the production of audio-visual content
Ez dut uste orain gertatu diren emaitzak errepikatuko direnik	I do not think the results that have happened now will be repeated
Nahi duena egiteko libre da, baina etxeak su hartzen badu, arazoa ez da berea bakarrik izango	He/She is free to do whatever he/she wants, but if the house catches fire, the problem will not only be his/hers
umeak triste daude	the children are sad
Amaiak bazkaria prestatu du	Amaia has prepared lunch
Gustatu beharko litzaizuke	You should like it
Eguraldi hobea egingo balu hondartzara joango nintzateke	If the weather was better I would go to the beach
egunero ikusten dut	I see it every day
Sare sozialetan ere jarraitzaile kopurua goraka doa, eta urte osoan ia etenik gabe lan egiten jarraituko da	The number of followers in social media is also increasing, and work will continue to be done almost without interruption all year long
Sare sozialetan ere jarraitzaile kopurua goraka doa, eta urte osoan ia etenik gabe lan egiten jarraituko du	The number of followers in social media is also increasing, and he/she will continue to work almost without interruption all year long