GECTurk: Grammatical Error Correction and Detection Dataset for Turkish

Atakan Kara, Farrin Marouf Safian, Andrew Bond, Gözde Gül Şahin
Computer Engineering Department
Koç University, Istanbul, Turkey
https://gglab-ku.github.io/

Abstract

Grammatical Error Detection and Correction (GEC) tools have proven useful for native speakers and second language learners. Developing such tools requires a large amount of parallel, annotated data, which is unavailable for most languages. Synthetic data generation is a common practice to overcome the scarcity of such data. However, it is not straightforward for morphologically rich languages like Turkish due to complex writing rules that require phonological, morphological, and syntactic information. In this work, we present a flexible and extensible synthetic data generation pipeline for Turkish covering more than 20 expert-curated grammar and spelling rules (a.k.a., writing rules) implemented through complex transformation functions. Using the pipeline, we derive 130,000 high-quality parallel sentences from professionally edited articles. Additionally, we create a more realistic test set by manually annotating a set of movie reviews. We implement three baselines formulating the task as i) neural machine translation, ii) sequence tagging, and iii) few-shot learning with prefix tuning, achieving strong results. Then we perform a zero-shot evaluation of our pretrained models on on the coarse-grained "BOUN -de/-da" and fine-grained expert annotated dataset. Our results suggest that our corpus, GECTurk, is high-quality and allows knowledge transfer for the out-of-domain setting. To encourage further research on Turkish GEC, we release our dataset, baseline models, and synthetic data generation pipeline with https://anonymous.4open.science/r/tr-gec-17D6/.

1 Introduction

Grammatical Error Correction (GEC) is among the well-established NLP tasks with dedicated shared tasks (e.g., BEA (Bryant et al., 2019)), benchmarks, and even specific evaluation measures. With the increasing interest from the community, the field is in constant need of novel writing tools, and methods, and more importantly, extensions to other languages.

Recently, there has been an explosion of research about GEC for high-resource languages, especially English (Rothe et al., 2021; Omelianchuk et al., 2020; Bryant et al., 2019). These recent techniques have used two main approaches: formulating the task as i) neural machine translation, i.e., generation (Rothe et al., 2021) and ii) token classification to detect erroneous tokens (Omelianchuk et al., 2020). First set of approaches mainly utilized and engineered vanilla Transformers to generate the corrected text, while, the second set focused on engineering a set of errors and transformation rules. Nonetheless, both formulations require a large set of parallel corpora containing grammatically correct and incorrect sentence pairs. Furthermore, the latter approach additionally requires a highly curated dataset with annotations for correcting errors (i.e., location and type of the error). However, constructing such a parallel corpus with error annotations is nontrivial—especially for low-resourced languages with rich morphology like Turkish. The challenge is due to grammar rules, a.k.a., writing errors being entangled in several layers, such as phonology, morphology, syntax, and semantics. For this language, there are no spelling or grammar error datasets, as mentioned by Çağrı Çöltekin et al. (2022), with the exception of the dataset introduced by Arikan et al. (2019).

To address this, we focus on the Turkish language and utilize the official writing rules prepared by the Turkish Language Association. Then we implement corruption, a.k.a., transformation, functions to generate instances that violate a specific rule, which requires challenging analysis of sentences on several linguistic levels, as well as curation of specialized lexicons. Then, we generate a large, synthetic, high-quality annotated corpus by applying transformation functions to professionally edited, modern Turkish articles. In addition to the

transformation functions used for data generation, we implement and share the reverse-transformation functions for validating the generated datasets and developing sequence tagger models, which achieve state-of-the-art in English.

In addition, we have collected a corpus of movie reviews and manually annotated 300 sentences with the proposed error types to evaluate the models in a real-life setting. Furthermore, we design and implement several baselines using standard neural machine translation (NMT) and sequence tagging models. While NMT models are only trained to generate the corrected sentences, sequence tagging models are trained to tag the tokens with the error type (if any) and then perform the associated reverse transformation function on the detected error to generate the correct text. Finally, we perform prefix-tuning (Li and Liang, 2021) on large multilingual language models for both detection and correction tasks.

Our results show that sequence tagging and mGPT (Shliazhko et al., 2022) models are able to achieve high scores on the Turkish grammatical correction task. However, when it comes to detecting errors—especially rare ones—sequence tagging approach outperforms all proposed techniques despite its small size. This suggests that albeit their high performance in *generating* corrected text, large language models might be harder to adapt to capture the error types, due to their autoregressive nature. On the other hand, a small dedicated model outperforms LLMs by focusing on the smaller detection problem and addressing the easier generation problem with designed reverse transformation rules.

Our exhaustive experiments with dataset sizes show that the proposed dataset is indeed more challenging than existing ones, as the models demonstrate a steeper learning curve due to the larger number of error types. Finally, we measure the zero-shot and out-of-domain ability of the proposed baselines and find that; again, the sequence tagging model is the most capable despite its smaller size compared to the large language model. We make our datasets, baseline models, and synthetic data generation pipeline publicly available at `https://anonymous.4open.science/r/tr-gec-17D6/`.

2 Related Work

English GEC Despite having a long history, with the BEA-2019 Shared Task on Grammatical Error Correction (Bryant et al., 2019), the GEC community started employing neural models and formulating GEC as a neural machine translation task (i.e., translate from grammatically incorrect to correct sentences), which has become the dominant approach. Another recent approach, GECToR (Omelianchuk et al., 2020), uses the idea of reverse transformations, which can be applied to a list of source tokens $[x_1, \ldots, x_n]$, in order to produce the desired correct grammar. Their model is a sequence tagger with a BERT encoder. Each tag corresponds to a transformation where transformations are applied after the sequence tagging finishes. In contrast, the gT5 model released by Xue et al. (2021) is a multilingual mT5 model fine-tuned on artificially corrupted sentences from the mC4 corpus and uses a span prediction and classification task to fix grammatical errors (Rothe et al., 2021). This does require a lot of additional training time, since the original mT5 model is not initially prepared for a similar task. Their model achieves SOTA results in 4 languages while only training once.

Turkish GEC Unlike English Grammatical Error Correction, data resources and existing solutions for Turkish Grammatical Error Correction are very scarce. Previously, Arikan et al. (2019) proposed a neural sequence tagger model and a synthetically generated dataset to correct "de/da" clitic errors, a common error made by Turkish native speakers. Öztürk et al. (2020) combined a contextual word embedding model, namely BERT, with a sequence tagger to correct "de/da" clitic errors. Although these errors are common, they constitute only a small portion of grammatical errors made by native speakers. In addition, while there are various forms of this error, the previous work only considers a few. Our data generation strategy considers multiple versions of the "de/da" clitic errors and many more common grammatical errors.

3 Synthetic Data Generation

The overall generation process is given in Fig. 1. First, we randomly sample from professionally edited Turkish corpora (§3.1). Then, sentences are corrupted—if possible—following the expert-curated transformation rules explained in §3.2, as

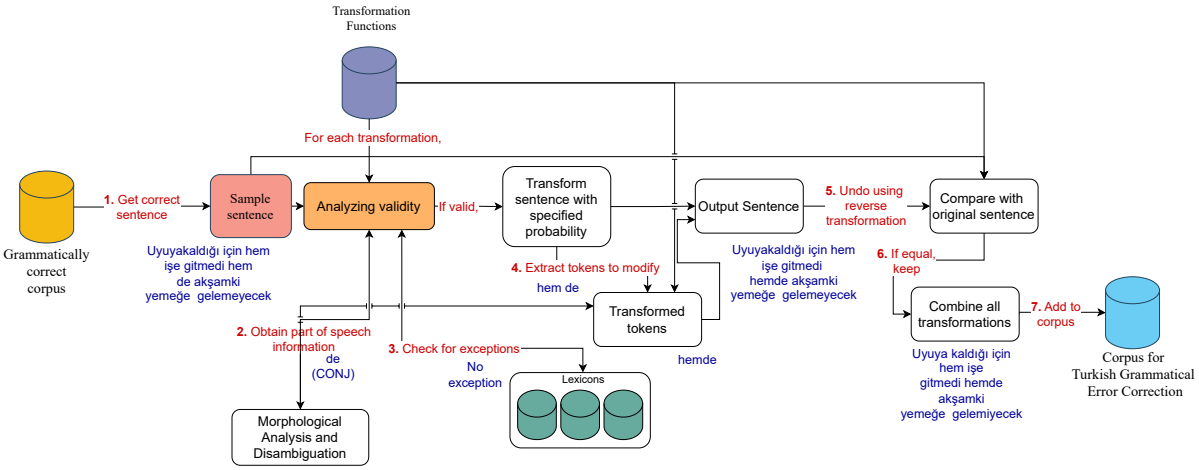

Figure 1: Data generation pipeline. 1) First, a correct sentence is obtained from the grammatically correct corpus. 2) Then, part of speech information is obtained and 3) checked against a lexicon of exceptions corresponding to each transformation function. If the sentence is valid, the necessary tokens are 5) extracted and modified with the transformation function. 6) This is then undone in order to avoid any edge cases from slipping through. Finally, 7) all the transformed sentences are combined, and the result is added to the corpus.

well as the use of a morphological analyzer. Finally, pairs of grammatically correct and corrupted sentences are added to the final Turkish GEC corpus following the M^2 standard.

3.1 Corpus

Our proposed data generation pipeline is built upon the assumption that all input sentences are grammatically correct. Hence, we base our study on previously compiled newspaper corpora (Diri and Amasyali, 2003; Amasyalı and Diri, 2006; Can and Amasyalı, 2016; Kemik NLP Group, 2022) that are proofread and went through a professional editing process. The articles are on various topics, including politics, sports, and medicine, and have been written by more than 95 authors for three different newspapers; in total, more than 7000 singly authored documents were collected between 2004-2012. Once we obtained grammatically correct source sentences, we performed several preprocessing steps, such as removing duplicates (2.9% of the combined dataset), ending with 138K unique sentences.

3.2 Transformation

The Turkish Language Association (TDK)[1], a government agency founded in 1932, is responsible for providing resources to conduct scientific research on written and oral sources of Turkish. Within this scope, they specify and maintain a comprehensive list of writing rules that is publicly available[2]. We rely on this expert-curated list to generate forward and backward transformation rules, which we refer to as f and f^{-1} respectively. However, the list is long, and some writing rules are intuitive to native speakers, so that any errors made on these rules sound abnormal to them. We selected the grammar rules that are most commonly used incorrectly by native speakers, determined by consulation with Turkish language experts and filtering the list from TDK using their feedback. We do not include any rules that are common for Turkish language learners but rarely made by native speakers. Table 1 provides the full list of the transformation rules produced by this work.

Applying f For each sampled sentence, first, we shuffle the list of fs. This ensures that mutually exclusive transformation functions are applied with desired frequencies. Then, we iteratively apply each f on the sentence given with the pseudo-code in Algorithm 1. Here, f gets an input sentence s, morphological analysis of the sentence M_s, an array of indicators for whether any transformation has been applied to the word— $flags$, and parameter $p \in (0,1)$. The algorithm, then, iterates over tokens (or pairs) and checks whether the token has been transformed. If not, it checks whether the token(s) are eligible for f. If eligible, we apply f with the probability p, since not all errors are made

[1] https://www.tdk.gov.tr/

[2] https://www.tdk.gov.tr/kategori/icerik/yazim-kurallari/

Algorithm 1 Apply f

Require: $s :=$ sentence, $M_s :=$ morphological analysis, flags, p
Ensure: tags
  $tags \leftarrow [\,]$
  $n \leftarrow$ Number of tokens in s
  for $i = 1 \rightarrow n$ **do**
    if flags$[i]$ **and** $is\_eligible(s, M_s)$ **and** $flipCoin(p)$ **then**
      tags.insert("A i $\{i+1\}$ ||| ruleID ||| sentence$[i]$ ||| REQUIRED ||| -NONE- ||| 0")
      sentence$[i] \leftarrow$ transformed token at index i
      flags$[i] \leftarrow$ **False**

with the same frequency by native speakers.[3]

Eligibility Check Some official writing rules require syntactic analysis at the token and sentence levels. For instance, to apply the transformation function CONJ_DE_SEP, one must perform morphological analysis and disambiguation to analyze the part-of-speech tags at the morpheme level. That is, CONJ_DE_SEP transformation can be applied **only if** a "-de/da" morpheme with a CONJUCTION part-of-speech tag is found. Additionally, a small set of rules requires specialized lexicons, e.g., a list of exceptional foreign words for FOREIGN_R2_EXC. To address the former, we use a state-of-the-art morphological analyzer Dayanik et al. (2018), and the lexicons are taken from the official lists provided by TDK.

Annotation Format We use the standard GEC annotation format following Ng et al. (2013) and Bryant et al. (2019). An example annotation is given in Fig. 3. Here, S and A refer to the ungrammatical sentence and edit annotations respectively. Each A contains starting and ending indices, the error type, the corrected phrase, and the id of the annotator.

Postprocessing Despite the use of professionally edited source sentences, there are still some grammatically incorrect sentences that slip through. We detect these cases by taking advantage of a key property of our reverse transformations: since each f is reversible, we should see $S = f^{-1}(f(S))$ for each sentence S. Therefore, at the end of the transformation process, we perform this check on every generated, grammatically incorrect sentence. If a sentence fails this check, then we know it is problematic, and we remove it from the corpus. The final sentences are thus properly modified in the desired way, with no unintentional side effects.

[3]We choose the probabilities intuitively after an initial analysis on web corpus and student essays.

3.3 Annotated Corpus

The annotated corpus includes more than 138K sentences, with 104K error annotations belonging to 25 error types given in §1. The generative pipeline controls the frequency of those error types, aiming to mimic the human error frequencies, given by Figure 6. As in the dataset of CoNLL-2014 shared task (Ng et al., 2014), some error types appear more frequently than others. These frequencies are by the probability parameters p; therefore, the difference between frequencies of error types is an intended result. Our dataset is finally split into a train/val/test set of 70%/15%/15%.

3.4 Curated Test Corpus

For a more realistic test setting, we collect movie reviews from a popular website [4] specifically focusing on top-rated movies. After performing sentence tokenization and deduplication, we shortlist sentences that potentially contain grammatical errors. To do so, we employed various techniques: for each rule, we utilized available dictionaries, incorporated regular expressions to identify specific morphemes that may exhibit errors, and employed models trained on a synthetic dataset. Then, our domain expert annotated the sentences for the proposed error types, following the standard GEC annotation format. By adhering to this procedure, we successfully produced a test dataset comprised of 300 sentences, wherein half of the sentences were grammatically correct and the other half contained errors.

4 Tasks and Models

4.1 Tasks

In this paper, we consider two tasks: Grammatical Error Correction (GEC) and Grammatical Error Detection (GED).

[4]www.beyazperde.com

Category	Rule ID	Description	f	Frequency
-DE/-DA	1. CONJ_DE_SEP	Conjunction "-de/-da" is written separately.	Durumu [oğluna da -> oğlunada] bildirdi.	12962
	2. CONJ_DE_VH	Conjunction "-de/-da" must follow the vowel harmony	Çok [da -> de] iyi olmuş.	101
	3. CONJ_DE_AR	Conjunction "-de/-da" does not follow phonetic assimilation rules.	Sınıf [da -> ta] temizlendi.	99
	4. YADA	"-de/-da" written together with the word "ya" is always written separately.	Sen [ya da -> yada] o buradan gidecek.	472
	5. CONJ_DE_APOS	Conjunction "-de/-da" cannot be used with an apostrophe.	[Ayşe de -> Ayşe'de] geldi.	10859
	6. CASE_DE	Suffix "-de/-da" is written adjacent.	[Evde -> Ev de] hiç süt kalmamıştı.	37462
-KI	7. CONJ_KI_SEP	Conjunction "-ki" is written separately.	Bugün öyle çok [yorulmuş ki -> yorulmuşki] hemen yattı.	1817
	8. CONJ_KI_EXC	On some exceptional instances, by convention, the conjunction "-ki" is written adjacent.	[Belki -> Bel ki], [oysaki -> oysa ki], [çünkü -> çünki]	1395
FOREIGN	9. FOREIGN_R1	Words that start with double consonants of foreign origin are written without adding an "-i" between the letters.	[gram -> gıram]	307
	15. FOREIGN_R2	Some foreign origin words undergo consonant assimilation	[sebebi -> sebepi]	327
	16. FOREIGN_R2_EXC	Exceptions to the FOREIGN_R2 rule	[evrakı -> evrağı]	422
BISYL	13. BISYLL_HAPL_VOW	Some bisyllabic words undergo haplology when they get a suffix starting with a vowel.	[ağzı -> ağızı]	1567
	14. BISYLL_HAPL_VOW_EXC	Exception to previous rule	[içeride -> içerde]	866
LIGHT VERB	17. LIGHT_VERB_SEP	Light verbs such as "etmek, edilmek, eylemek, olmak, olunmak" are written separately in case of no phonological assimilation	[arz etmek -> arzetmek]	460
	18. LIGHT_VERB_ADJ	Light verbs are written adjacent in case of phonological assimilation e.g., liaison	[emretti -> emir etti]	547
COMPOUND	20. COMP_VERB_ADJ	Compound words formed by knowing, giving, staying, stopping, coming, and writing are written adjacent if they have a suffix starting with -a, -e, -ı, -i, -u, -ü.	[uyuyakalma -> uyuya kalmak], [gidedurmak -> gide durmak], [çıka gelmek -> çıka gelmek]	7840
SINGLE	22. PRONOUN_EXC	Traditionally, some pronouns are written adjacent.	[hiçbir -> hiç bir], [herhangi -> her hangi]	3867
	23. SENT_CAP	The first letter of the sentence is capitalized.	[Onlar -> onlar] geldi.	2613
	24. CAPPED	Some Arabic and Persian originated words are written with capped letters.	[kâğıt -> kağıt], [karargâh -> karargah]	834
	25. ABBREV	Grammatical rules for abbreviations, such as adding suffixes to abbreviations, punctuations with abbreviations etc.	[Alm. -> Alm], [THY'de -> THY'da], [cm'yi -> cm'ye]	359
	12. PRONOUNC_EXC	Unlike its pronunciation, verbs ending with "-a/-e" do not mutate when they get a suffix other than "-yor"	[başlayacağım -> başlıyacağım]	12750

Table 1: Selected list of writing rules introduced by Turkish Language Association. $f[arg1 -> arg2]$ refers to the transformation function where the correct and corrupted surface forms are given with $arg1$ and $arg2$ respectively.

Grammatical Error Correction (GEC) takes as input a grammatically incorrect sentence and outputs the corrected version of the sentence. Formally, given an input sentence $\mathbf{x} = (x_1, \cdots, x_T)$ which contains some grammar mistakes, the aim is to produce an output sentence $\mathbf{y} = (y_1, \cdots, y_{T'})$ which contains no grammatical errors. Conditions are not imposed on how the model produces grammatically correct sentences.

Grammatical Error Detection (GED) takes a slightly different approach to this problem, with the goal of producing detailed information about the errors in the source sentence. This includes details about the type of error and the location of the error in the sentence. Formally, given an input sentence $\mathbf{x} = (x_1, \cdots, x_T)$, we can represent the problem as a token-level classification task, where the output is $\mathbf{c} = (c_1, \cdots, c_T)$, and c_i represents the error type of token i. Given the knowledge that an error of type c_k occurred at the location from m to n, it is then possible to apply the corresponding reverse transformation f^{-1}, and fix the error.

4.2 Models

We introduce three models to evaluate the performance of GECTurk: An NMT baseline, a sequence tagger using BERT (Devlin et al., 2019) pretrained on Turkish, and mGPT using prefix-tuning. All models are trained using 1 Nvidia V100 GPU.

NMT Baseline: We train a vanilla transformer model (Vaswani et al., 2017) for GEC. This choice is inspired by the most recent shared task on grammatical correction (Bryant et al., 2019), where many of the winning teams used transformer-based models and modeled the problem as a Neural Machine Translation (NMT).

The training dataset consists of triples $\{(x_i, y_i, a_i)\}_{i=1}^{N}$, where x_i is the i-th input sentence, y_i is the corresponding ground truth corrected sentence, and a_i are the annotations. During training, the model receives x_i as input.

Due to the nature of the formulation, NMT is only used for correction.

Sequence Tagger: Similar to recent work (Omelianchuk et al., 2020), we train a sequence tagging model using a cased BERT encoder, pretrained on Turkish text (Schweter, 2020) with default configurations and additional linear and softmax layers on the top. The BERT model uses the WordPiece tokenizer (Wu et al., 2016) that segments tokens into subwords. Therefore, each sentence in the dataset is first tokenized into subwords and passed into the BERT encoder. We only hold the first subword's representation for words with multiple subword tokens. Then, the encoder's representations are linearly transformed and passed to the softmax layer to classify into possible error types described in Table 1 or no error. The model is finetuned for token classification objective using cross-entropy loss. The loss is similar to Eq. 1, except ranging over the number of possible error types. The advantage of this model is the ability to perform error detection easily, as opposed to simply error correction. Correction is simply performed with reverse transformations as described previously.

Prefix Tuning: Inspired by the recent successes of prefix tuning (Li and Liang, 2021) as an alternative to model fine-tuning, we use OpenPrompt (Ding et al., 2022) to perform prefix tuning on mGPT (Shliazhko et al., 2022). Despite being multilingual and primarily focused on other languages, mGPT achieves encouraging results on morphologically rich languages (Acikgoz et al., 2022).

In prefix tuning, we append N trainable (soft) tokens to the front of each input. Therefore, given input $\mathbf{x} = (x_1, \cdots, x_T)$, the new input becomes $\mathbf{x} = (s_1, \cdots, s_N, x_1, \cdots, x_T)$, where the s_i's are the added artificial tokens. We then optimize only these tokens during training, while leaving the original model frozen.

We primarily utilize mGPT for sequence generation, where the model simply outputs the grammatically correct sentence. We use the standard sequence generation prompt provided by OpenPrompt, due to its recent success (Acikgoz et al., 2022), and use teacher forcing during training. When the model detects multiple types of errors in the same sentence, all the detection information is generated on the same line.

Here, we model both correction and detection tasks in the same sequence generation approach, where the corrected sentence is first generated, and then information about the violated rule, and the location of this error is generated at the end of the sentence. This allows for one trained model to output both results. In order to train this correctly, the target sentence was appended with the details of the error type and location, and used for loss calculations. An example is provided in Fig 2.

Dataset	#s	#a	%e	#e
GECTurk	138K	104K	49.7%	25
BOUN	10K	6K	50.0%	2
BOUN complex	102	105	100.0%	2

Table 2: Datasets used for training and evaluation. #s: sentences, #a: annotations, %e: percentage of erroneous sentences, #e: number of errors types

5 Experimental Setup

5.1 Datasets

GECTurk consists of 138K sentences, of which 70% (97K) are used as the training set, and 15% (21K) each was used for the validation and test sets. GECTurk consists of 25 different rules, with varying levels of frequency 6.

The BOUN dataset (Arikan et al., 2019) consists of 15K training, 3.5K validation and 2K test sentences. However, the errors in the dataset correspond to only 2 out of the 25 errors in GECTurk. It also includes a complex split, a list of 100 sentences that are supposed to be extra challenging. Details of the datasetes are given in Table 2.

5.2 Evaluation

5.2.1 Grammatical Error Correction

Following the evaluation methods of Omelianchuk et al. (2020) and Bryant et al. (2019), we evaluate our correction results using Precision (P), Recall (R), and $F_{0.5}$ score.

This evaluation is done using the M^2 scorer (Dahlmeier and Ng, 2012) (MaxMatch), following the guidelines from the CoNLL-2014 Shared Task.

5.2.2 Grammatical Error Detection

To allow for a fair comparison with the BOUN dataset from (Arikan et al., 2019), we use the same metrics, namely Precision (P), Recall (R), and F_1 score. Since the task is modeled as a sequence tagging problem, this aligns with the standard evaluation for sequence tagging, such as in Huang et al. (2015).

We compute both macro and micro F_1. Macro F_1 computes the unweighted average of the F_1 scores for each class. We include macro because of the imbalance of samples for different classes, where an easy grammatical rule that appears very often will inflate the total scores a lot. However, micro is still important to understand the overall performance on the entire dataset, is the one used by Arikan et al. (2019).

To calculate these scores, we use SeqEval (Nakayama, 2018), a common library for evaluating sequence tagging tasks. We use one tag for each error type, and convert from each model's output format to the necessary

6 Experiments and Results

We conduct a set of experiments to investigate the performance of the proposed baselines and the difficulty level of the introduced dataset: GECTurk. In our main experiment, we train the baseline models using the experimental setup explained in §5. Each model is trained using three different fixed seeds. The mean and standard deviation of their performances on both GEC and GED (if applicable) are given in Table 3. Both SeqTag and mGPT provide exceptionally strong results over 0.94 $F_{0.5}$ score for the GEC task, compared to the NMT baseline model. On the other hand, the detection task is performed more competently by SeqTag—as expected—than mGPT, which again achieves around 0.99 $F1_{micro}$ and 0.90 $F1_{macro}$ score. When we use the macro F1 measure, where rare rules are equally important, mGPT's F1 detection performance drops to around 0.28, while SeqTag still provides high scores.

We, then, used the weights that performed highest on our synthetic test set, and evaluated without any additional training on the hand-annotated corpus as given in Table 3. For detection, SeqTag performs similarly to synthetic test data, while mGPT's performance increases dramatically, proving the importance of being exposed to real-life data during pretraining. However, SeqTag still outperforms mGPT by a large margin. On the other hand, both models perform significantly worse on the correction task for the movie-review data, suggesting a larger room for improvement on the more challenging test set.

6.1 Out-of-Domain

Next, we investigate the our-of-domain ability of the proposed models on unseen datasets. We first evaluate our pretrained models on the BOUN (Arikan et al., 2019) standard and complex test splits to gain insights into their zero-shot ability. Our results for the standard split and complex split are given in Table 4. SeqTag achieves an F1 score that is on-par with state-of-the-art even

	Detection			Correction		
GECTurk						
	P	**R**	**F1**	**P**	**R**	**$F_{0.5}$**
NMT	-	-	-	0.50 ± 0.01	0.84 ± 0.01	0.55 ± 0.01
SeqTag	0.90 ± 0.003	0.90 ± 0.013	0.90 ± 0.006	0.98 ± 0.001	0.98 ± 0.001	0.98 ± 0.001
mGPT	0.52 ± 0.02	0.38 ± 0.004	0.41 ± 0.01	0.95 ± 0.01	0.92 ± 0.03	0.94 ± 0.01
Movie-Review Test Data						
NMT	-	-	-	0.31	0.62	0.35
SeqTag	0.94	0.87	0.89	0.85	0.80	0.84
mGPT	0.73	0.52	0.59	0.75	0.61	0.72

Table 3: Detection and Correction results of the baselines on a) GECTurk (in-domain) and b) manually annotated movie-reviews dataset.

	Detection				Correction		
BOUN Zero-Shot							
	Prec.	Rec.	F1	Accuracy	Prec.	Rec.	$F_{0.5}$
NMT	-	-	-	-	0.19	0.56	0.22
SeqTag	0.91	0.72	0.80	0.99	0.81	0.63	0.77
mGPT	0.60	0.56	0.58	0.97	0.75	0.67	0.73
BOUN Results	0.92	0.82	0.87	-	-	-	-
BOUN Complex Split Zero-Shot							
NMT	-	-	-	-	0.71	0.73	0.72
SeqTag	0.99	0.93	0.96	0.99	0.99	0.93	0.98
mGPT	0.61	0.96	0.90	0.93	0.79	0.75	0.78
BOUN Results	-	-	-	0.71	-	-	-
BOUN Full Training							
NMT	-	-	-	-	0	0	0
SeqTag	0.97	0.86	0.91	0.99	0.97	0.86	0.94
mGPT	0.61	0.48	0.53	0.97	0.85	0.73	0.82
BOUN Results	0.92	0.82	0.87	-	-	-	-

Table 4: Model results for BOUN dataset. The first section uses models trained on the GECTurk dataset, and evaluated zero-shot on the BOUN test split. The second section uses the same zero-shot models on the BOUN complex split. The third section is for our models only trained on the BOUN dataset, and evaluated on the test split. All precision and recall values displayed are micro scores.

in the zero-shot setting for the standard split. It surpasses state-of-the-art accuracy scores on the complex split by a large margin together with the mGPT model. We note that despite the claims made by the authors of the BOUN dataset, our experiments found no additional complexity in the "complex" split.

We, then, perform additional experiments via training our proposed model from scratch on the BOUN (Arikan et al., 2019) training split to compare their performance against the state-of-the-art. The results are given in Table 4. For this setup, the NMT model was not able to fully converge, and just produced unhelpful noise. Therefore, we list the results of this model as 0. Following our previous results, SeqTag achieves and F1 score of 0.91, this time surpassing the state-of-the-art by 0.05 pp.

7 Conclusion and Future Work

In this work, we have presented an annotated dataset for Grammatical Error Correction (GEC) and Detection (GED) containing more than 20 Turkish writing error types proposed by language experts. We have also introduced a flexible and extensible data generation pipeline that can be used to create a synthetic dataset from grammatically correct sentences. We used this pipeline to create a large-scale dataset using multiple opinion columns from Turkish newspapers. In addition, we have manually constructed a more challenging test set by annotating the movie-reviews with the proposed error types.

Finally, we implemented a diverse set of strong baseline models, by training from scratch, fine-tuning, or if applicable, using prefix tuning. Our results show that simpler models focusing on the smaller problem of detecting the error types outperform large pretrained models on both the synthetic and real-life datasets, especially for the detection task. On the other hand, we observe that pretraining helps the models to handle more realistic cases, even though they still lag behind the simpler models. Our out-of-domain results suggest that training on the synthetic data gives a strong prior to both small and larger models.

As a future work, we plan to extend the pipeline with more error types based on spelling correction and enlarge the GECTurk using the proposed pipeline.

Acknowledgements

This work has been supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) as part of the project "Automatic Learning of Procedural Language from Natural Language Instructions for Intelligent Assistance" with the number 121C132. We also gratefully acknowledge KUIS AI Lab for providing computational support. We thank our anonymous reviewers and the members of GGLab who helped us improve this paper.

Limitations

There are a few key limitations to our work. One key issue is that the data generation pipeline is very time-consuming as it requires the use of a morphological analyzer. This prevents the pipeline from being used for very large-scale datasets. Additionally, the requirements for hand-crafted rules and reverse transformations slows the speed at which new rules can be added, and which rules can even be added.

Another important limitation is the necessity of dictionaries to handle exceptions to grammatical rules. Words in Turkish that have been borrowed from other languages (notably Persian, French, and Arabic) tend to not align with the normal grammar rules, and thus require special lists of exceptions. While we aimed to include as many as possible, it is definitely possible that we missed some, which can lead to rare edge cases where our pipeline fails.

Regarding our models, mGPT is very computationally intensive to work with, even when only doing prefix tuning. This prevented us from training until fully converged, and instead only opted for 1 epoch.

Ethical Considerations

Transformer-based large language models such as (Vaswani et al., 2017; Shliazhko et al., 2022; Rothe et al., 2021) are very successful, but they also have some ethical concerns. First, the model is highly dependent on the dataset it was pre-trained on. It is possible that the dataset contained certain biases, such as racism or sexism, which will later be passed to the model outputs. While this is an important detail to focus on, our work does not focus much on actually training these models. We perform training using GECTurk, which is collected from opinion columns, as well as the BOUN dataset, which is already publicly available. Due to the publication of these columns in reputable Turkish newspapers, they contain less bias than the average document collected online. While the pre-training procedure itself may have introduced some biases, we are unable to handle these in our work. While these problems are inherent to all deep learning models, we emphasize transformer models here due to our model choices.

Another possible ethical issue is the misuse of grammatical correction models for cheating. By having a model that can automatically detect and correct grammatical errors, students can more easily use these to score better than normal on assignments and exams, with less time invested. Not only is this bad for the student's learning, but it also affects others who can be negatively impacted by the student's artificial success.

Despite the concerns about misuse or inherent biases in the model, we believe that grammatical error correction models are more beneficial than harmful. Many people, from authors and writers, to language learners, can benefit from having grammar corrections. By introducing a dataset and demonstrating models on Turkish, an under-served language in the NLP community, more people will be able to take advantage of this, similar to the many existing tools for English.

References

Emre Can Acikgoz, Tilek Chubakov, Müge Kural, Gözde Gül Sahin, and Deniz Yuret. 2022. Transformers on multilingual clause-level morphology. *CoRR*, abs/2211.01736.

M. Fatih Amasyalı and Banu Diri. 2006. Automatic turkish text categorization in terms of author, genre and gender. In *Natural Language Processing and Information Systems*, pages 221–226, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ugurcan Arikan, Onur Güngör, and Suzan Uskudarli. 2019. Detecting clitics related orthographic errors in turkish.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. 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, Florence, Italy. Association for Computational Linguistics.

Ender Can and Mehmet Fatih Amasyalı. 2016. Text2arff: A text representation library. In *2016 24th Signal Processing and Communication Application Conference (SIU)*, pages 197–200.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572, Montréal, Canada. Association for Computational Linguistics.

Erenay Dayanik, Ekin Akyürek, and Deniz Yuret. 2018. Morphnet: A sequence-to-sequence model that combines morphological analysis and disambiguation. *CoRR*, abs/1805.07946.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Haitao Zheng, and Maosong Sun. 2022. Openprompt: An open-source framework for prompt-learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022 - System Demonstrations, Dublin, Ireland, May 22-27, 2022*, pages 105–113. Association for Computational Linguistics.

Banu Diri and Mehmet Fatih Amasyali. 2003. Automatic author detection for turkish texts.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *ArXiv*, abs/1508.01991.

Kemik NLP Group. 2022. Our datasets. `http://www.kemik.yildiz.edu.tr/data/File/2500koseyazisi.rar`. Online; accessed 26-November-2022]".

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Hiroki Nakayama. 2018. seqeval: A python framework for sequence labeling evaluation. Software available from https://github.com/chakki-works/seqeval.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland. Association for Computational Linguistics.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The CoNLL-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–12, Sofia, Bulgaria. Association for Computational Linguistics.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 702–707, Online. Association for Computational Linguistics.

Stefan Schweter. 2020. Berturk - bert models for turkish.

Oleh Shliazhko, Alena Fenogenova, Maria Tikhonova, Vladislav Mikhailov, Anastasia Kozlova, and Tatiana Shavrina. 2022. mgpt: Few-shot learners go multilingual. *CoRR*, abs/2204.07580.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings*

of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 483–498, Online. Association for Computational Linguistics.

Çağrı Çöltekin, A. Seza Doğruöz, and Özlem Çetinoğlu. 2022. Resources for turkish natural language processing: A critical survey.

Hasan Öztürk, Alperen Değirmenci, Onur Güngör, and Suzan Uskudarli. 2020. The role of contextual word embeddings in correcting the 'de/da' clitic errors in turkish. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

A Model Details

A.1 NMT Baseline

For tokenization, we used BerTurk-cased (Schweter, 2020) tokenizer, passed into the NMT model. The transformer model has 6 encoders with embedding size 512, 6 decoder layers, and 8 heads. A dropout of 0.1 is used directly after the positional embeddings. For training, an Adam (Kingma and Ba, 2014) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\varepsilon = 1e-9$, and a learning rate of $1e-4$. We used batch size of 32, and trained the model for 100 epochs on a single V100.

We use a standard cross-entropy loss during training, as follows:

$$\mathcal{L}_{GEC}(\hat{y}, y) = -\sum_{n=1}^{N}\sum_{c=1}^{V} log\frac{exp(\hat{y}_{n,c})y_{n,c}}{\sum_{i=1}^{V} exp(\hat{y}_{n,i})} \quad (1)$$

Here, N is the batch size, V is the number of error classes, x is the model output, and y is the target.

For the data size experiments, we used the same architecture but with slightly different hyperparameters. For both the 75% and 100% experiments, the model was trained for 100 epochs. For the 50% experiment, we only trained for 50 epochs. When training 10% and 25%, the Adam optimizer is used with the same β values, a learning rate of $5e-4$, and a weight decay of $1e-4$, for 100 epochs.

The zero-shot testing on the BOUN (Arikan et al., 2019) dataset is tokenized with the same tokenizer, and the best pre-trained model from GECTurk is used for evaluation.

A.2 Sequence Tagger

For training, we used the AdamW(Loshchilov and Hutter, 2019) optimizer for 3 epochs, using batch size 16, learning rate $2e-5$, weight decay 0.01, $\beta_1 = 0.9$, and $\beta_2 = 0.999$.

A.3 Prefix Tuning

We used the standard mGPT tokenizer and the OpenPrompt prefix tuning template. All experiments use 5 soft tokens at the beginning. Teacher forcing is used during training, and both the correction and detection tasks are formulated as a sequence generation problem.

Following the settings from (Acikgoz et al., 2022), we don't use weight decay for the bias and LayerNorm weights. The AdamW optimizer is used, with an initial learning rate of $5e-5$, linearly decaying to 0 over the entire training. We clip the norm of the gradient at 1.0. Due to the computational requirements of mGPT, we only train on GECTurk for a single epoch on all experiments. However, on the smaller BOUN dataset, we train for 5 epochs.

For inference, we also follow the hyperparameters from (Acikgoz et al., 2022), using a temperature of 1.0, top p of 0.9, no repetition penalty, and a beam search of 5 beams.

For all experiments, a batch size of 3 was used. The max sequence length, including soft tokens, is set to 512.

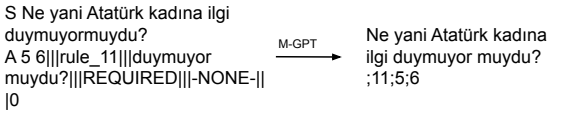

Figure 2: Example output of the mGPT model. The first line performs the grammatical error correction, and subsequent lines allow for detection. The annotations are included for comparison with model outputs, but are not actually provided to the model.

Uyuyakaldığı için hem işe gitmedi hem de akşamki yemeğe gelemeyecek .

(Because they overslept, they didn't go to work and won't be able to come to dinner tonight.)

(a)

Uyuya kaldığı için hem işe gitmedi hemde akşamki yemeğe gelemiyecek .

(b)

S Uyuya kaldığı için hem işe gitmedi hemde akşamki yemeğe gelemiyecek.
A 0 2|||COMP_VERB_ADJ|||Uyuyakaldığı|||REQUIRED|||-NONE-|||0
A 6 7|||CONJ_DE_SEP|||hem de|||REQUIRED|||-NONE-|||0
A 9 10|||PRONOUNC_EXC|||gelemeyecek|||REQUIRED|||-NONE-|||0

(c)

Figure 3: The grammatically correct sentence is given in (a), the transformed version is given in (b), and the annotation format is given in (c).

B Effect of Dataset Size

In order to obtain a better understanding of how important the dataset size is for this task, we conducted training on $1, 10, 25, 50, 75$, and 100 percent of GECTurk and evaluated each model using the same evaluation measures. Fig 4 shows how the performance of the models vary with more training data. NMT reaches its top point with around 75% of the training data, while SeqTag and mGPT achieve high $F_{0.5}$ scores with 25% of the training split. However, as discussed before, correction scores can be misleadingly high, since high fre-

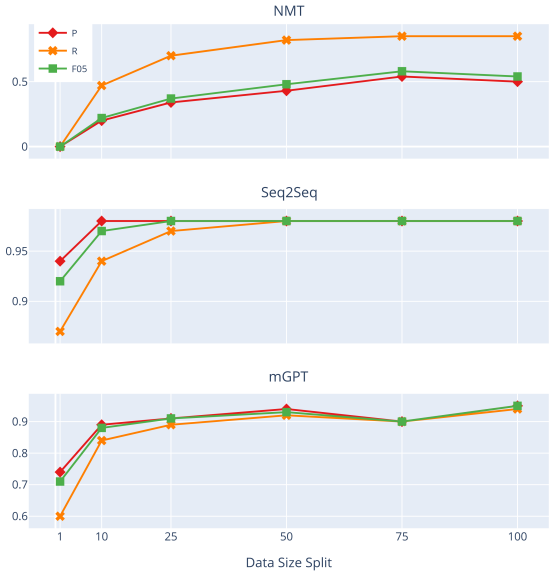

Figure 4: Correction performance of each model trained on varying sizes of GECTurk

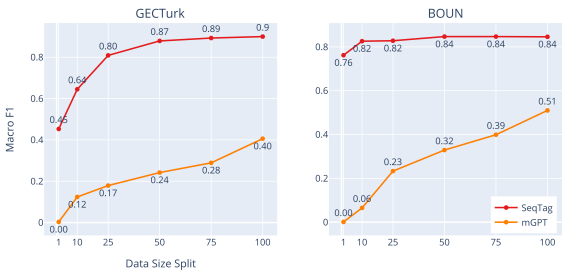

Figure 5: Macro-F1 performance of SeqTag and mGPT on GECTurk and Boun training splits.

quency and easy to correct errors will push the results much higher. Hence we also plot the macro F1 scores for the detection task both on TurkishGEC and BOUN datasets in Fig 5. The plot shows that the GECTurk dataset is richer than the BOUN, since SeqTag and mGPT macro F1 performances are much steeper on the former.

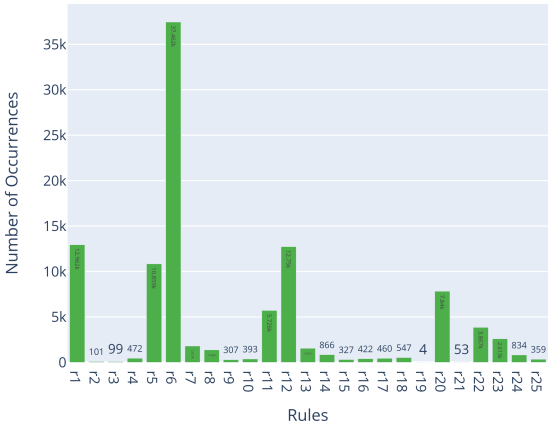

Figure 6: Number of sentences with each writing rule type.