Corpora Generation for Urdu Grammatical Error Correction

Anonymous ACL submission

Abstract

Grammatical Error Correction (GEC) for Urdu remains an under-researched area due to the lack of annotated datasets. This paper addresses the challenge of generating a robust corpus for fine-tuning deep learning models aimed at Urdu GEC. We propose a method for synthesizing a large dataset by collecting errors from the Urdu WikiEdits history, learning from them and inserting similar errors in grammatically correct sentences to generate incorrect sentences with grammatical errors, hence creating a pair of grammatically correct and incorrect sentences. Furthermore, we have created a Gold Dataset by extracting errors from exam copies of students. After the generation of the dataset, we have also fine-tuned various models against synthetically generated data and evaluated them against gold data to show the quality of synthetic data generation.

1 Introduction

Urdu is a language with a rich heritage and a script influenced by Persian, Arabic, Hindi, Sanskrit, and Turkish. It is widely spoken in South Asian countries, including Pakistan and India, by a large population. Despite its complexity and the variety of grammatical components, unfortunately, there has been little work done in the field of Urdu grammatical error correction (GEC). An automated GEC system for Urdu would greatly enhance content creation, digital communication tools, and educational platforms. To build a state-of-the-art model that generalizes well to a wide range of human grammatical errors and outputs correct sentences for grammatically incorrect inputs, we need a large corpus of training data.

Although there isn't a vast corpus available for this task, there are proven methods (used for other languages) that can help generate such datasets. Our research mainly aims to investigate these techniques and apply them to the Urdu language to create a robust corpus of Urdu.

In our research, we are tackling two primary research questions: (1) "Generation of synthetic datasets for a low-resource language like Urdu," and (2) "Building a state-of-the-art model that can output correct sentences for input sentences that are grammatically incorrect but require a single edit (insertion, deletion, or substitution) of words for correction".

2 Related Work

Sonawane et al. (2020) generates a parallel corpus of synthetic errors by inserting errors into grammatically correct sentences using a rule-based process, focusing specifically on inflectional errors. They generate this artificial dataset for fine-tuning of models, and for evaluating those models, they scrape the edit history of Wikipedia and use that as a real-world data source to test the fine-tuned model on. This paper encourages that their strategies for Hindi can be applied to other Indic fusional languages such as Urdu, Bengali, etc.

Lichtarge et al. (2019) provides a strategy for corpora generation uses the Wiki edits dataset, which benefits from a more accurate distribution of natural grammatical errors made by humans. In their research, errors are introduced probabilistically in the source sequences randomly selecting deletion, insertion, replacement, or transposition of adjacent characters for each introduced error. The second approach they use for data generation is of round-trip machine translation through a high-resource bridge language. They further augment this data with the common errors found in the Wiki edit approach and introduce them in dataset. The authors pre-train the model on the above Wikipedia revision dataset and then fine-tune the model on a Lang-8 dataset, which is a parallel corpus for GEC but with high noise. Wikipedia derived data demonstrated to benefit significantly from finetuning on Lang-8 because the Wikipedia revision dataset contains corrections that are outside the scope of GEC.

Foster and Andersen's (2009) paper introduces

GenERRate, a probabilistic error-infliction tool that generates errors such as POS insertions, deletions, and substitutions. It performs well in some scenarios but has limitations, particularly in handling covert errors like sentence structure improvements. This synthetic data shows improved accuracy and recall, especially useful for small-to-medium corpora. Combining natural and synthetic data helps mitigate performance degradation, showing promise for augmenting training sets with artificial data.

3 Methodology of Dataset Generation

3.1 Synthetically Generated Dataset

Our approach for creating a synthetically generated dataset is a pipeline uses the established paradigm of "error annotation and infliction," a principled approach for creating synthetic data in low-resource settings (Alrehili and Alhothali, 2025). In the first phase of our pipeline (Annotation Phase), given a set of correct and grammatically incorrect sentence pairs, learns the different types of errors from those pairs, and in its second phase (Infliction Phase), given clean Urdu sentences, outputs corrupted sentences resulting in GEC pairs, where the corruption is the same as the one learned from the set of correct and grammatically incorrect sentence pairs.

3.1.1 Extraction of Errors from Wikipedia Revision Histories

So, as the first phase of our pipeline to generate GEC pairs, we try and procure a source of natural grammatical errors. Similar to how Sonawane et al. (2020) uses Wikipedia revision histories as a real-world data source to test the fine-tuned model on, we use it as a natural source of grammatical errors (because it contains a huge amount of synthetic grammar errors, as the majority of text available in low-resource languages is translated from high-resource languages, which are then corrected by humans). The Urdu Wikipedia revision histories contain sentence pairs as prior-post edit sequences, and they consist of the history of all edits made by people reviewing and editing content on Wikipedia. We are only utilizing the last edit made for any sequence as correct (since it ensures that the last edit is in its grammatically correct form), and the second last edit as incorrect. After collection, the dataset has gone through a rigorous cleaning process to ensure high quality, which includes:

1. Removal of extra whitespaces and non-urdu characters.
2. Removal of the sequences that contain a high percentage of numeric or non-alphabetical characters, since these aren't of much use to us.
3. Removal of the extremely similar sentences. We removed these extra pairs since they are not very helpful to our task.
4. Translation of non-Urdu Unicode characters to the Unicode Urdu range, such as conversion of Arabic characters to Urdu, and translation of numeric characters from English to Urdu.

This resulted in the generation of our "Wiki-Edits Dataset" consisting of approximately 240,000 sentence pairs.

3.1.2 Annotation

The goal of the annotation phase is to build a structured, context-rich database of observed grammatical errors. This is achieved through a multi-step process for each incorrect-correct sentence pair in the "Wiki-Edits Dataset".

The first step for the annotation phase was to perform context-aware analysis on both sentences of the sentence pairs to determine the parts-of-speech (POS), lemma (root words), and morphological features in the context of their respective sentences. For this, we used Stanza (Qi et al., 2020), an NLP Toolkit that provides a complete pipeline for accurate text analysis on many languages, including Urdu. Stanza's Urdu model, fine-tuned on the UD 2.12 treebank dataset (Bhat et al.; Palmer et al., 2009), demonstrates high performance, with reported accuracy scores of 94.78% for lemmatization, 92.98% for UPOS tagging, and 84.06% for morphological features. This level of accuracy is crucial for the annotation phase, as it ensures that the grammatical context of each error is captured with high fidelity.

The second step was to establish an Urdu vocabulary to perform Out-of-Vocabulary (OOV) checks. For this purpose, we utilized the UD dataset (Bhat et al.; Palmer et al., 2009) which was structured in the CoNLL-U format, a standard for annotating data at both the sentence and word/token level. This format includes ten fields for each word, detailing its index, form, lemma, universal POS tag (UPOS), language-

specific POS tag (XPOS), morphological features (FEATS), and its head in the dependency tree, among other things. Since this dataset is quite vast and Stanza was also annotated on it, we decided to go with this dataset as our vocabulary; this could be potentially expanded upon later, but even this limited vocabulary resulted in a substantial amount of synthetic data being generated.

The third step was to determine the precise edits that transform an incorrect sentence into its correct version. This is accomplished using a custom sentence alignment algorithm inspired by the ERRANT(**ERR**or **AN**notation **T**oolkit) pipeline (Bryant et al., 2017), which classifies single-word edits as one of three types: 'Insertion' (I), 'Deletion' (D), or 'Substitution' (S). Figure 1 shows an example of this.

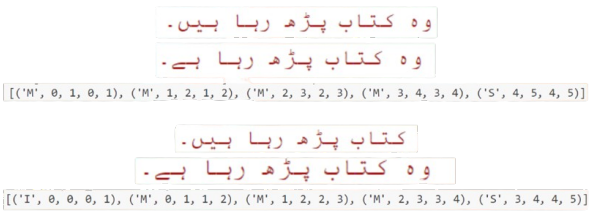

Figure 1: There are two example outputs of the alignment algorithm, both on the same grammatically incorrect input sentence (the top one in both examples), which has five words in it; "وہ", "کتاب", "پڑھ", "رہا", and "ہیں". The output of the first example (transition of type S) shows that the sentence has 5 words in total, 4 matching words "M", and 1 substituted word "S". The output of the second example (transition of type S and I) shows that there has been an insertion "I" at the first index and a substitution at the last index.

Unlike a standard Levenshtein distance, where all substitutions have a cost of 1, our alignment algorithm uses a nuanced linguistic cost function, get_sub_cost, to better model grammatical similarity. The substitution cost between an original token o and a corrected token c is calculated as:

$$\text{Cost}(o, c) = \text{Cost}_{\text{lemma}} + \text{Cost}_{\text{pos}} + \text{Cost}_{\text{char}}$$

Where:

- $\text{Cost}_{\text{lemma}}$ is 0 if the lemmas are identical and 0.499 otherwise. This penalizes substitutions that change the base meaning of the word.
- Cost_{pos} is 0 if the Universal Part-of-Speech (UPOS) tags are the same, 0.25 if both are open-class POS tags (e.g., Noun to Verb), and 0.5 otherwise. This reflects the grammatical severity of the change.
- $\text{Cost}_{\text{char}}$ is the normalized Indel distance between the token strings, accounting for spelling variations.

This cost function ensures that the alignment prefers linguistically plausible edits, such as changing an inflection, over substituting an unrelated word.

Kernel-Based Error Representation. With the alignment and context-aware features in hand, we capture the local context of each error using a three-word "kernel". The kernel's structure depends on the error type and is defined by the UPOS tags of the words immediately surrounding the edit. To handle cases where a context word does not exist, such as an edit at the start or end of a sentence, we use a placeholder tag, `'%'`.

- For **Substitution** and **Insertion** errors, the kernel is [UPOS_left, UPOS_middle, UPOS_right]. Either UPOS_left or UPOS_right can be the `'%'` placeholder if the edit occurs at a sentence boundary.
- For a **Deletion** error, the kernel represents the "gap" where the word was removed. Consequently, the middle element is always the placeholder, resulting in the structure [UPOS_left, `'%'`, UPOS_right].

This kernel serves as the primary key in our annotations database, allowing us to group errors that occur in similar grammatical contexts.

Annotation Generation and Storage. For each edit (S, D, I) identified by the alignment algorithm, a structured JSON annotation is created, the details of the structured annotation depend on the type of edit:

Substitution (S). An annotation is created if both the incorrect and correct words pass the OOV check. The annotation object contains:

- type: 'S'.
- kernel_upos: The list of three UPOS tags forming the kernel.
- incorrect_feats: A UPOSFeats object (UPOS and feature string) for the incorrect word, as analyzed by Stanza in the source sentence.

- correct_feats: A UPOSFeats object for the correct word, as analyzed in the target sentence.
- occurrence: A counter for this specific error pattern under this kernel.

Deletion (D). An annotation is created if the deleted word and its surrounding context words (if they exist) pass the OOV check. The annotation object contains:

- type: 'D'.
- kernel_upos: The three-element UPOS kernel with % in the middle.
- kernel_feats: A list containing the UPOSFeats of the context words.
- deleted_words: The text of the word that was deleted.
- occurrence: A counter for this specific error pattern under this kernel.

Insertion (I). An annotation is created if the inserted word and its left and right context words from the correct sentence all pass the OOV check. The annotation object contains:

- type: 'I'.
- kernel_upos: The list of three UPOS tags forming the kernel.
- kernel_feats: A list containing the UPOSFeats of all three words in the kernel.
- inserted_word: The text of the word that was inserted.
- occurrence: A counter for this specific error pattern under this kernel.

Final Annotation Data Structure. The complete annotations database is stored as a dictionary, "Annotations Dictionary", where each key is the string representation of a kernel_upos list (e.g., "['PRON', 'NOUN', 'ADP']"). The corresponding value is a list of all unique annotation objects that share that same UPOS kernel. This structure allows multiple distinct error patterns, such as different feature-level substitutions or insertions, to be grouped under a single, high-level grammatical context. The Figure 2 shows an example of how the Annotations Dictionary looks like:

```
{
  "['PRON', 'NOUN', 'ADP']": [
    {
      "type": "S",
      "kernel_upos": ["PRON", "NOUN", "ADP"],
      "incorrect_feats": {
        "upos": "NOUN",
        "feats": "Case=Nom|Gender=Masc..."
      },
      "correct_feats": {
        "upos": "NOUN",
        "feats": "Case=Acc|Gender=Masc..."
      },
      "occurrence": 42
    },
    /*
    ... other substitution or insertion
    errors with the same UPOS kernel
     */
  ],
  "['NOUN', '%', 'VERB']": [
    {
      "type": "D",
      "kernel_upos": ["NOUN", "%", "VERB"],
      "kernel_feats": [
        {
          "upos": "NOUN",
          "feats": "Case=Acc..."
        },
        null,
        {
          "upos": "VERB",
          "feats": "Voice=Act..."
        }
      ],
      "deleted_words": ["نے"],
      "occurrence": 15
    }
  ]
}
```

Figure 2: Example of Final Annotation Data Structure.

3.1.3 Infliction

Once the structured database of error patterns is built, the second phase of the pipeline, Error Infliction, uses this knowledge to corrupt a large, clean corpus, thereby generating synthetic incorrect-correct sentence pairs. For this process, we used the Makhzan Corpus (mak), an effort towards curating a large collection of clean Urdu text for enabling research on Urdu NLP, ensuring that the baseline for error generation is grammatically sound. This corpus had about 272,296 clean sentences.

The infliction process for each clean sentence from the corpus follows a systematic, multi-step procedure to ensure that the generated errors are both contextually appropriate and

realistic.

Pattern Matching on Clean Sentences. The core of the infliction algorithm operates by sliding a three-word window across each clean sentence. The process begins with essential pre-processing steps.

1. **Normalization and Analysis:** Each input sentence is first standardized using character normalization to ensure consistency. It is then analyzed by the Stanza pipeline to acquire the context-aware UPOS tag, morphological features, and lemma for every word.
2. **Kernel Generation:** At each position in the sentence, a three-word UPOS kernel is generated from the current window (left word, middle word, right word). An Out-of-Vocabulary (OOV) check, using the same vocabulary as used in the annotation phase, is performed on the words in the window; if any are not in our dictionary, the window is skipped to prevent unreliable matches.
3. **Pattern Retrieval:** The generated UPOS kernel is used as a key to query the "Annotations Dictionary". If a match is found, the algorithm retrieves the list of associated error patterns (Substitution, Insertion, or Deletion) that were observed in that specific grammatical context.

Error Generation Mechanisms. If a kernel match is found, the algorithm attempts to inflict the corresponding error patterns. The logic for inflicting an error is specific to its type, involving rigorous context validation to ensure high fidelity.

Substitution Infliction. This is a multi-stage process designed to find a grammatically incorrect but morphologically plausible replacement for a word.

1. **Context Validation:** For a given substitution pattern from the annotations, the algorithm first validates that the UPOSFeats (the combination of UPOS and features) of the middle word in the clean sentence's window exactly match the correct_feats stored in the annotation. This critical step ensures that the grammatical context is precisely the one in which this error was originally observed.
2. **Candidate Search:** If the context is valid, the algorithm retrieves the lemma of the middle word. For that, it queries a pre-compiled word-to-lemma dictionary to find all other known morphological forms of that lemma. (We generated this lemma dictionary using the UD 2.12 treebank dataset (Bhat et al.; Palmer et al., 2009), because this dictionary had all its sentences annotated by professionals, having each word in the sentence and its lemma and morphological form is of great use to us.) Figure 3 shows how a lemma looks like as a key in the pre-compiled word-to-lemma dictionary and have value as a list of all of its derived forms.
3. **Feature Matching:** The algorithm iterates through other derived forms of the lemma of the middle word. For each candidate that passes an OOV check, it consults the UD 2.12 treebank dataset (Bhat et al.; Palmer et al., 2009) to see if any of its known grammatical usages in the thousands of sentences written there have UPOSFeats that exactly match the incorrect_feats from the annotation. Figure 4 shows how the lookup of the morphological features of a word in the UD 2.12 treebank looks like, among the many sentences it has been used in.
4. **Infliction:** If a candidate form with matching incorrect features is found, a new, incorrect sentence is generated by replacing the original middle word of the correct sentence with that candidate. (If more than one fit the description, only one candidate is randomly selected, and the corresponding generated correct–incorrect sentence pair is retained to ensure a balanced and non-redundant dataset.)

```
{
    "لانا": [
        "لائیں", "لائے", "لانے", "لایا",
        "لائی", "لانا",  "لاتے", "لائیں_گے",

    ]
}
```

Figure 3: Example of how the pre-compiled word-to-lemma dictionary formatting looks like.

Deletion Infliction (from an Insertion Pattern). The infliction of a deletion error is achieved by finding a context in the clean sentence that matches a learned *insertion* pattern.

```
{
    "ہیں": [
        {
            "id": "35",
            "text": "ہیں",
            "lemma": "ہے",
            "upos": "VERB",
            "xpos": "VM",
            "feats": "Mood=Ind|Number=Plur|
            Person=3|Tense=Pres|VerbForm=Fin|
            Voice=Act",
            "head": "9",
            "deprel": "acl:relcl",
            "deps": "_",
            "misc": "SpaceAfter=No|Vib=ہے|
            Tam=hE|ChunkId=VGF2|ChunkType=head|
            Stype=declarative|Translit=hīñ|
            LTranslit=he"
        },
        {
            "id": "15",
            "text": "ہیں",
            "lemma": "ہے",
            "upos": "AUX",
            "xpos": "VAUX",
            "feats": "Gender=Masc|Mood=Ind|
            Number=Plur|Person=3|Tense=Pres|
            VerbForm=Fin",
            "head": "13",
            "deprel": "aux",
            "deps": "_",
            "misc": "SpaceAfter=No|Vib=ہے|
            Tam=hE|ChunkId=VGF|ChunkType=child|
            Translit=hīñ|LTranslit=he"
        },
        // other occurrences of "ہیں" continue
        here
    ]
}
```


Figure 4: Example of how the UD 2.12 treebank dataset looks like in a dictionary format.

The logic is that if a word was commonly inserted in a specific three-word context, then the middle word of that same context in a clean sentence is a prime candidate for erroneous deletion.

1. **Context Validation:** The algorithm checks if the UPOSFeats of all three words in the clean sentence's current window exactly match the kernel_feats stored in a corresponding insertion annotation.
2. **Infliction:** If the context is a perfect match, the middle word is identified as a candidate for deletion. A new, incorrect sentence is generated by simply removing it.

Insertion Infliction (from a Deletion Pattern). Conversely, inflicting an insertion error corresponds to finding a context—a gap between two words—that matches a learned *deletion* pattern.

1. **Context Validation:** The algorithm considers the gap between two adjacent words in the clean sentence. It checks if the UPOSFeats of the left and right words exactly match the context features stored in a deletion annotation (which has a placeholder in the middle position).
2. **Infliction:** If the context matches, the algorithm retrieves the deleted_words from that annotation. After verifying the word is in-vocabulary, it is inserted between the two context words in the clean sentence.

Upon successful infliction, the system stores the generated correct–incorrect sentence pair, along with the associated error pattern object from the "Annotations Dictionary".

Aligning Synthetic Data Distribution with Naturally Occurring Errors. Our infliction algorithm, by design, could generate errors uniformly, creating a distribution that does not match our natural source, the "Wiki-Edits Dataset". To rectify this, we implemented a probabilistic down-sampling method. We first calculated the relative frequency of each unique error pattern observed in the "Wiki-Edits Dataset". We then sampled our synthetically generated sentence pairs according to this natural distribution, ensuring the final error frequencies mirrored the source data. This alignment process yields our final large-scale synthetic corpus of 1.27 million sentence pairs.

4 Experimental Setup

To empirically validate the effectiveness of our data generation pipeline, we designed an experimental setup to measure the GEC capabilities imparted to various state-of-the-art language models. Our evaluation framework is intentionally designed to test for robust, real-world generalization.

4.1 Datasets and Evaluation Strategy

Our evaluation relies on a clear and deliberate separation between our large-scale synthetic training data and a high-quality, manually-curated test set.

- **Training Corpus (Synthetic):** The primary training data for all fine-tuning experiments is our **1.27 million synthetically generated sentence pairs**. The creation of this corpus is detailed in Section 3.
- **Test Corpus (Gold):** For evaluation, we use a held-out, high-quality "**Gold**

Dataset" of naturally occurring errors. This dataset was meticulously curated from real-world sources. We reached out to institutions to get exam papers from students studying Urdu above grade 8. Those exam papers had student errors that were highlighted by Urdu instructors. We manually extracted those errors and created pairs of clean and corrupted sentences. This includes mistakes made by native and non-native speakers and people at different levels of Urdu proficiency. Furthermore, we also extracted errors from some exam exercises available online, which were used to correct grammar errors in Urdu. We applied the same process to some books that had similar activities.

Justification of Evaluation Strategy: Our decision to use this Gold Dataset *exclusively* for testing is a deliberate methodological choice designed to provide the most stringent and realistic test of our system's generalization capabilities. By fine-tuning models solely on our synthetic corpus and evaluating them on this completely unseen, out-of-domain, human-verified data, we measure not just the memorization of synthetic patterns, but the model's ability to learn and apply genuine grammatical rules to general, real-world errors. This makes it a clean, unbiased benchmark for the Urdu GEC community.

Both our Synthetic and Gold datasets will be made publicly available to encourage future research.

4.2 Model Selection

It is a common practice that GEC tasks are framed as sequence-to-sequence (seq2seq) translation tasks, a paradigm renowned for its efficacy. We evaluated on multiple unique neural architectures of this type.

- **NLLB:** We use Meta AI's No Language Left Behind (Team et al., 2022) (1.3B and 3.3B parameter variants), a powerful multilingual translation model designed to support over 200 languages.
- **mT0:** Google's Multitask T5 (Muennighoff et al., 2022) (base and large variants), an instruction-tuned model with strong performance on GEC tasks as compared to its predecessor, MT5.
- **ByT5:** Google's tokenizer-free ByT5 (Xue et al., 2022) (base variant), which processes raw bytes and is theoretically more sensitive to morphological and spelling errors.

4.3 Training and Optimization

To assess different training paradigms, we employed both full fine-tuning and parameter-efficient methods.

- **Full Fine-Tuning:** All models were fine-tuned on the entire 1.27 million pair synthetic dataset for one full epoch. The **Adafactor** optimizer was used to manage memory constraints, which is critical for training these large-scale models.
- **Parameter-Efficient Fine-Tuning (PEFT):** For the **mT0-base** and **NLLB-1.3B** models, we also experimented with Low-Rank Adaptation (LoRA). The reduced memory footprint of PEFT allowed for the use of the **AdamW** optimizer.

All training was conducted in a distributed environment utilizing NVIDIA RTX 3060, RTX 4090, TITAN RTX, T4, and A100 GPUs.

4.4 Evaluation Metrics

Following standard GEC evaluation practices, we benchmark all models on the held-out Gold Dataset. We report scores for **GLEU**, **BLEU**, and **BERTScore**. GLEU (Napoles et al., 2016) is specifically designed for GEC tasks, balancing precision and recall of n-gram matches between system outputs and references, making it well-suited for measuring correction quality. These metrics have been widely adopted in low-resource GEC research (Náplava and Straka, 2019), demonstrating their effectiveness for both lexical and semantic evaluation in morphologically rich languages such as Urdu.

5 Results and Analysis

This section presents the empirical results of our experiments. We first report the main quantitative outcomes, demonstrating the performance boost imparted by our synthetic dataset. We then provide a deeper analysis of the results across different model architectures and conclude with a qualitative look at the types of errors our fine-tuned models learn to correct.

Table 1: Evaluation results on the held-out gold test set. Baseline = pretrained checkpoint; LoRA = adapter-finetuned. BLEU and GLEU are reported in percent; BERTScore is F1 (0–1). Num_epochs = 1 for all

Model	Variant	BLEU (%)	GLEU (%)	BERTScore (F1)
ByT5	ByT5-base (baseline)	14.94	15.95	0.702
	ByT5-base (finetuned)	70.21	70.38	0.951
MT0	mt0-base (baseline)	49.47	49.15	0.898
	mt0-large (baseline)	46.28	45.28	0.892
	mt0-base + LoRA (finetuned)	66.37	66.76	0.948
	mt0-large (finetuned)	68.54	68.60	0.951
NLLB	nllb-1.3B (baseline)	11.61	12.38	0.752
	nllb-3.3B (baseline)	22.35	23.51	0.778
	nllb-1.3B + LoRA (finetuned)	66.86	67.16	0.949
	nllb-3.3B (finetuned)	65.55	65.87	0.939

5.1 Main Quantitative Results

To measure the impact of our synthetic corpus, we evaluated each model on the held-out Gold Dataset in two settings: a zero-shot baseline (before fine-tuning) and after fine-tuning on our 1.27 million sentence pairs. The results are summarized in Table 1.

The results in Table 1 reveal a substantial and consistent improvement across all metrics for every model after being fine-tuned on our synthetic corpus. The low baseline scores, particularly for ByT5 and NLLB, confirm that even powerful multilingual models lack specialized GEC capabilities for Urdu out-of-the-box. Fine-tuning provides a dramatic performance boost; for instance, the ByT5-base model's GLEU score increases by over 54 points, from 15.95% to 70.38%. The fully fine-tuned mT0-large model achieves one of the highest overall scores with a GLEU of 68.60% and a BERTScore of 0.951, demonstrating that our data generation method effectively equips state-of-the-art architectures with robust GEC capabilities for Urdu. Furthermore, the parameter-efficient LoRA fine-tuning proves highly effective, with **mT0-base + LoRA** and **NLLB-1.3B + LORA** achieving competitive scores of 66.37 and 66.86 BLEU, respectively, demonstrating that significant GEC capability can be achieved with minimal parameter updates.

6 Conclusion

This paper addressed the critical scarcity of resources for Urdu Grammatical Error Correction (GEC) by introducing a principled, two-phase pipeline for large-scale synthetic corpus generation. Our approach learns fine-grained, context-aware error patterns from Wikipedia revision histories and systematically inflicts them onto a clean monolingual corpus, resulting in two key contributions: a replicable generation methodology and the release of a 1.27 million-pair synthetic corpus and a "Gold" test set. The profound effectiveness of this approach is confirmed by our experiments, which show dramatic performance gains across state-of-the-art models like mT0, NLLB, and ByT5 after fine-tuning. These results validate our method and provide the essential contributions to spur the development of practical GEC tools for millions of Urdu speakers.

6.1 Limitations

While our work establishes a strong baseline, we acknowledge several limitations that offer avenues for future research:

- **Source Data Domain:** Although the WikiEdits corpus provides a rich source of naturalistic errors, its encyclopedic style may not fully capture the nuances and error patterns common in other domains, such as conversational text or creative writing. The error distribution is primarily reflective of Wikipedia contributors, which may differ from that of language learners.
- **Scope of Error Correction:** Our kernel-based approach, which relies on a three-word local context, is highly effective for single-token edits (substitutions, insertions, deletions) but may not adequately capture more complex, non-local grammatical errors, such as incorrect sentence structure or discourse-level phenomena.
- **Evaluation on Limited Gold Data:** Due to the scarcity of annotated resources for Urdu, our "Gold" test set, while high-quality, is of a limited size. A larger, more

diverse evaluation benchmark would be needed to make more definitive claims about real-world performance across different demographics of Urdu speakers.

- **Computational Constraints:** Fine-tuning large models required significant computational resources, which constrained our ability to conduct more extensive hyperparameter searches or train for extended epochs. This may have prevented some models from reaching their full potential on our dataset.

7 Ethical Statement

We have considered the ethical implications of our work throughout the research process.

- **Data Sources and Privacy:** The primary source for our synthetic data generation, Wikipedia, is a public resource with content licensed under Creative Commons (CC BY-SA 3.0). The text is encyclopedic and generally non-personal. While we did not implement an explicit PII removal step, a manual review of a sample of the data did not reveal any instances of sensitive personal information. The usernames associated with edits are pseudonymous and were not used in any part of our analysis. The "Gold Dataset" was derived from anonymized student exam papers, collected with institutional consent for research purposes. All personally identifiable information was removed during the manual extraction process.

- **Potential for Bias:** Our synthetic dataset inherits the biases present in its source corpora. The "Wiki-Edits Dataset" reflects the language and error patterns of its contributors, while the "Gold Dataset" reflects those of students in a specific educational context. The models trained on this data will consequently learn these biases. We believe that releasing both datasets allows future researchers to study and mitigate these biases, but users of models trained on this data should be aware of its origins.

- **Intended Use and Impact:** The intended use of our work is to advance NLP research and tool development for Urdu, a low-resource language. An automated GEC system has a significant positive impact by enhancing content creation, improving digital communication tools, and providing valuable feedback in educational platforms. We are not aware of any direct potential for malicious use of this technology. Both the code and the datasets will be made publicly available to ensure transparency, reproducibility, and to foster further research for the benefit of the Urdu-speaking community.

References

Ma zan. https://github.com/zeerakahmed/makhzan/.

Ahlam Alrehili and Areej Alhothali. 2025. Towards the development of balanced synthetic data for correcting grammatical errors in arabic: An approach based on error tagging model and synthetic data generating model. *Preprint*, arXiv:2502.05312.

Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, et al. The hindi/urdu treebank project. In *Handbook of Linguistic Annotation*. Springer Press.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada. Association for Computational Linguistics.

Jennifer Foster and Oistein Andersen. 2009. GenERRate: Generating errors for use in grammatical error detection. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 82–90, Boulder, Colorado. Association for Computational Linguistics.

Jared Lichtarge, Chris Alberti, Shankar Kumar, Noam Shazeer, Niki Parmar, and Simon Tong. 2019. Corpora generation for grammatical error correction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3291–3301, Minneapolis, Minnesota. Association for Computational Linguistics.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

Jakub Náplava and Milan Straka. 2019. Grammatical error correction in low-resource scenarios. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 346–356, Hong Kong, China. Association for Computational Linguistics.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2016. Gleu without tuning. *Preprint*, arXiv:1605.02592.

Martha Palmer, Rajesh Bhatt, Bhuvana Narasimhan, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Ankur Sonawane, Sujeet Kumar Vishwakarma, Bhavana Srivastava, and Anil Kumar Singh. 2020. Generating inflectional errors for grammatical error correction in Hindi. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 165–171, Suzhou, China. Association for Computational Linguistics.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. No language left behind: Scaling human-centered machine translation. *Preprint*, arXiv:2207.04672.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.