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

The field of Grammatical Error Correction (GEC) has witnessed a paradigm shift in recent years, moving from early rule-based and statistical classifiers to deep learning approaches. Specifically, treating GEC as a sequence-to-sequence (Seq2Seq) translation task has become the standard in the field, largely following the pioneering work of Yuan and Briscoe (2016), who demonstrated the efficacy of Neural Machine Translation (NMT) architectures for correcting grammatical errors. However, the success of these supervised deep learning models is heavily contingent on the availability of massive, high-quality parallel corpora of grammatically incorrect and correct sentence pairs. While such resources exist for high-resource languages like English, they are virtually nonexistent for low-resource languages, creating a significant bottleneck for development.

Urdu is a language with a rich heritage and a script influenced by Persian, Arabic, Hindi, Sanskrit, and Turkish. ((Butt and Ahmed, 2011), (Mangrio, 2016)) 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 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.

To address this data scarcity, this paper proposes a robust method for synthetic data generation. We propose a pipeline that mines naturally occurring error patterns from Urdu Wikipedia revision histories and re-inflicts them onto clean text. In doing so, we tackle 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."

Finally, given the substantial morphosyntactic similarities Urdu shares with other Indo-Aryan languages such as Hindi, Punjabi, and Sindhi (Butt and Ahmed, 2011; Mangrio et al., 2021), we hypothesize that our data generation pipeline is transferable to this broader linguistic family. Provided that prerequisite resources such as reliable POS taggers and dependency treebanks are available, this methodology can be effectively adapted to develop robust GEC systems for other low-resource Indic languages.

2 Related Work

Research in GEC has largely focused on methods to overcome the scarcity of annotated training data through synthetic error generation (Bryant et al., 2023). This section outlines the evolution of these techniques from random noise injection to linguistically grounded probabilistic modeling.

2.1 The Shift to Neural GEC and Data Scarcity

As established by Yuan and Briscoe (2016), modeling GEC as a monolingual machine translation task allows for powerful generalization, provided sufficient parallel data exists. In the absence of large annotated corpora (such as NUCLE or Lang-8 used for English), researchers have turned to synthetic data generation. Early approaches, such as GenERRate by Foster and Andersen (2009), introduced probabilistic error-infliction tools to generate errors like POS insertions, deletions, and substitutions. While useful for small-to-medium corpora, purely random or simple rule-based noise often fails to capture the complexity of human errors, leading to models that do not generalize well to real-world inputs.

2.2 Linguistically Grounded Error Generation

To improve the quality of synthetic data, recent work has emphasized the importance of linguistic context over random noise. Sidorov et al. (2013) argued that traditional lexical n-grams are insufficient for modeling syntactic relations and demonstrated that Syntactic N-grams (utilizing POS tags) significantly improve error detection and correction. This theoretical foundation supports the move toward "kernel-based" or pattern-based error generation.

Building on this, Ma et al. (2022) utilized a Linguistic Rules-Based Generation (CLG) approach. They explicitly defined grammatical rules (e.g., "Structural Confusion") to generate synthetic errors that mimic the structural complexity of real mistakes. Their findings highlight that synthetically generated errors must be linguistically plausible to be effective for training NMT systems. This contrasts with earlier methods by Foster and Andersen (2009), which relied on hard-coded rules, by demonstrating that preserving the "grammatical severity" of an error is crucial for downstream model performance.

2.3 Mining Error Distributions from Natural Data

Rather than manually crafting linguistic rules, modern approaches attempt to learn error distributions directly from noisy data. Felice and Yuan (2014) were among the first to move away from inserting random errors, instead calculating conditional probabilities of errors based on linguistic context (e.g., $P(error|POStag)$) derived from learner corpora. Similarly, Kasewa et al. (2018) argued against random noise injection, proposing a method to learn the probability distribution of errors from a seed corpus and project that distribution onto clean text. This "mining and grafting" strategy ensures that the synthetic data reflects the natural distribution of errors made by humans, a technique we adopt in our pipeline by downsampling synthetic data to match natural error frequencies.

2.4 GEC for Low-Resource and Indic Languages

In the context of utilizing web-scale resources, Lichtarge et al. (2019) proposed generating corpora using Wikipedia edit histories (WikiEdits). They employed a dual strategy of probabilistic error introduction and Round-Trip Translation (RTT) through a high-resource bridge language. Their work demonstrated that Wikipedia revision histories contain valuable "natural" errors that benefit models significantly when fine-tuned.

Specific to Indic languages, Sonawane et al. (2020) adapted these strategies for Hindi, a language linguistically similar to Urdu. They generated a parallel corpus of synthetic errors by focusing on inflectional errors via a rule-based process and validated their models against scraped Wikipedia edit histories. Our work extends these methodologies to Urdu; however, rather than relying solely on rule-based inflection or random noise, we employ a kernel-based extraction method inspired by the syntactic dependencies highlighted by Sidorov et al. (2013) and the probabilistic mining approaches of Felice and Yuan (2014), ensuring our synthetic corpus captures the specific morpho-syntactic nuances of Urdu.

3 Dataset Generation

3.1 Methodology for Synthetically Generated Dataset

Our approach for creating a synthetically generated dataset is a pipeline that 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

To capture natural grammatical errors for the Annotation phase, we mined Urdu Wikipedia revision histories, adopting the strategy used by Sonawane et al. (2020) for Hindi. Given that Wikipedia content in low-resource languages often originates from translation followed by human post-editing, the edit history provides a rich source of grammatical corrections. We extracted edit sequences, treating the final revision as the grammatically correct reference and the penultimate revision as the incorrect source. Following extraction, we applied a standard cleaning pipeline to filter noise, the details of which are provided in Appendix A, which is also based on (Sonawane et al., 2020), ensuring that the errors collected are grammatical only. This process yielded the **Wiki-Edits Dataset**, comprising approximately 240,000 parallel sentence pairs.

3.1.2 Error Annotation

The goal of the annotation phase is to build a structured, context-rich database of observed grammatical errors from the **Cleaned Wiki-Edits Dataset** using each of its incorrect-correct sentence pairs.

First, we performed context-aware analysis using **Stanza** (Qi et al., 2020) to extract POS tags, lemmas (root words), and morphological features for every word in each sentence pair in the **Cleaned Wiki-Edits Dataset**. Stanza was selected for its high accuracy on Urdu (Appendix B). To ensure that our pipeline focuses on morphological shifts rather than spelling corrections, we established a closed vocabulary for Out-of-Vocabulary (OOV) checks derived from the UD 2.12 treebank (Bhat et al.; Palmer et al., 2009). This ensures tokenization consistency, as Stanza and our vocabulary share the same underlying schema from the treebank.

Next, we identified the precise edits transforming incorrect sentences into correct ones using a custom alignment algorithm inspired by **ERRANT** (Bryant et al., 2017), which classifies edits as 'Insertion' (I), 'Deletion' (D), or 'Substitution' (S); Figure 1 shows an example. Unlike standard Levenshtein distance, our algorithm employs a linguistically motivated cost function to better model grammatical similarity (exact formulation in Appendix C).

Figure 1: Two example outputs of the alignment algorithm 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.

Kernel-Based Error Representation. To capture the grammatical environment of an error, we define a local context window, or "kernel," of size k. This consists of the token at the edit index (w_i) and its surrounding neighbors. For example, if $k = 3$, the kernel includes the immediate left and right neighbors (w_{i-1}, w_{i+1}). We treat k as a tunable hyperparameter to balance context-awareness against data sparsity. Rather than storing raw lexical forms, we abstract tokens into their **Morpho-Syntactic States** using Stanza. We observed that different error types require different levels of contextual granularity. Therefore, we defined distinct matching criteria for each operation:

- **Substitution Patterns:** We model this as a *Morphological Transition* of the target word constrained by a *Syntactic Frame*. The frame is defined strictly by the UPOS tags of the context window. The transition is defined by the change in morphological features of the middle word (e.g., Case: Nom → Case :cc). We do not constrain the morphological features of the surrounding tokens in the window for substitutions, allowing the pattern to generalize across different neighbor inflections.
- **Insertion and Deletion Patterns:** Since these operations involve the pres-

ence or absence of a word, we enforce a stricter context validation: the kernel is defined by both the UPOS tags *and* the exact morphological features of the surrounding words in the context window. For example, in the case of $k = 3$, deleting a postposition requires checking that the preceding noun is in the specific morphological state (e.g., Oblique) that licensed that postposition.

Annotation Storage. We aggregate these patterns into a database of observed errors, where each entry stores the matching criteria (Context UPOS and/or Features) and the transformation rules required to reproduce the error. For **Substitutions**, we record the specific mismatch in morphological features between the incorrect and correct words (e.g., a shift in Case). For **Deletions**, we store the exact surface word to be removed. For **Insertions**, we record the strict morphological context of the neighbors that indicates a missing word. Detailed schema definitions for these annotation objects are listed in Appendix D.

3.1.3 Error Infliction

In the final phase, we utilize the learned error patterns to corrupt the **Makhzan Corpus** (mak), a large collection of cleaned, grammatically correct corpus consisting of around 270,000 Urdu sentences.

The infliction process operates by sliding a window across every sentence in the clean text, analyzing every possible local context window of size k with Stanza to extract UPOS tags and morphological features. A context window in the clean sentence is identified as a candidate for corruption if its local morpho-syntactic context exactly matches the *target* (correct) side of an error pattern stored in our Annotations Database.
To ensure high-quality generation, we apply specific linguistic constraints for each error type:

Substitution Infliction (The Lemma Constraint). Blindly substituting words based solely on POS tags would likely result in semantic drift. To prevent this, we generate substitution errors only when two conditions are met:

1. The word in the clean corpus must have morphological features matching the correct_feats of a retrieved error pattern.
2. There must exist a derived form from the **lemma** of the clean word matching the incorrect_feats of the pattern (verified via our **Lemma Dictionary**, see Appendix E.1).

These conditions ensure that the generated error is a valid morphological variant of the original word, preserving the sentence's semantic core.

Insertion and Deletion Infliction. For these operations, we apply the logic of "reversing the correction" with strict context validation (described in Appendix E.2):

- **Deletions:** To simulate a deletion error, we identify a context in the clean sentence that matches a learned *Insertion* pattern. If the clean words match the full morphological profile of an inserted word and its neighbors, the middle word is removed to generate the erroneous sentence of the pair.
- **Insertions:** To simulate an insertion error, we look for a context matching a learned *Deletion* pattern. If a sequence of clean words matches the context defined by a Deletion pattern (where the 'gap' represents the token to be inserted), we inject the missing word as stored in the annotation database to generate the erroneous sentence of the pair.

3.2 Creation of the Gold Test Dataset

While synthetic data is essential for training in low-resource settings, evaluating models solely on synthetic data often leads to inflated performance metrics that fail to reflect real-world generalization. To establish a rigorous, unbiased benchmark for Urdu GEC, we curated a high-quality **Gold Test Dataset** of naturally occurring errors.

Data Sources and Curation. We collaborated with local educational institutions to procure work from Urdu exam papers of students (Grade 8 and above). The selected schools were chosen so that the student demographic there comprises both L1 and L2 learners to add diversity. Additionally, we extracted exercises from Urdu grammar workbooks designed for error correction tasks.

Scope and Filtering. We used help from expert linguists to manually filter out purely orthographic errors (spelling mistakes) or stylistic fluency edits that did not violate grammatical rules. This resulted in a dataset of 1,613 high-quality parallel pairs.

Error Distribution Statistics. To provide transparency regarding the types of errors covered, we provide statistics of the Gold-Dataset using the standardized ERRANT-type metrics (Bryant et al., 2017) to illustrate the distribution of error types. Table 1 details the breakdown of error types.

Error Category	Count	%	Example (Incorrect → Correct)
Verb & Aux	1,131	33.2	کرتی → کرتا (Gender/Number)
Noun & Pron	733	21.5	لڑکا → لڑکے (Oblique Case)
Adpos	560	16.4	کا → کی (Genitive Agreement)
Mod & Misc	618	18.2	اچھا → اچھے (Adj Agreement)
Ortho	362	10.6	مسئلہ → مسلہ (Spelling)
Total Edits	**3,404**	**100.0**	**Avg. 2.1 Errors / Sentence**

Table 1: Distribution of error types in the Gold Test Set (1,600 sentences), grouped by macro-grammatical categories.

To foster reproducibility and further research in Urdu NLP, we intend to release the **Gold Test Corpus** (1,613 pairs).
The Gold corpus is, to our knowledge, the first expert-annotated GEC benchmark for Urdu derived from diverse educational sources.

Evaluation Protocol. This dataset is treated exclusively as a **Held-Out Test Set**. It is never seen by the model during any training experiment. By fine-tuning on synthetic data and evaluating on this completely different distribution of human errors, we ensure that our reported metrics reflect genuine grammatical generalization rather than pattern memorization.

4 Optimization of Synthetic Data Generation

The methodology described in Section 3.1 allows for flexible generation of errors based on three configurable parameters:

1. **Context Window Size (k):** Defines the granularity of the morpho-syntactic environment required to trigger an error injection.

2. **Sampling Strategy:** Determines the probability distribution used to select which error pattern to inflict when multiple matching patterns exist.

3. **Error Density:** Controls the number of distinct errors injected into a single sentence.

To determine the optimal configuration for generating the final synthetic corpus, we conducted a series of ablation studies.

4.1 Optimization Protocol and Metrics

For these optimization experiments, we utilized the **mT0-large** (1.2B parameters) (Muennighoff et al., 2022) model as a proxy. We fine-tuned the model on different synthetic variations and evaluated performance on our Gold Dataset. All ablation runs used identical hyperparameters (detailed in Appendix **??**) to ensure that performance differences were solely attributable to data characteristics.

To quantify performance, we utilize established metrics (Yuan and Briscoe, 2016) for Grammatical Error Correction:

- **MaxMatch (M^2):** We use the M^2 scorer (Dahlmeier and Ng, 2012) to report **Precision**, **Recall**, and $\mathbf{F_{0.5}}$.
- **GLEU:** The Generalized Language Evaluation Understanding metric (Napoles et al., 2016), utilized to measure fluency and overlap with reference sentences.

4.2 Ablation 1: Context Window Size (k)

The kernel size k dictates the granularity of the error context. While a larger k theoretically captures longer-range dependencies, it increases data sparsity, as finding larger exact n-gram matches in a clean corpus becomes exponentially harder.

We experimented with $k = 3$ (± 1 neighbor), $k = 5$ (± 2), and $k = 7$ (± 3). For this experiment, we held the **Sampling Strategy (Natural)** and **Error Density (Single Edit)** constant.

Kernel	Pairs	P	R	$F_{0.5}$	GLEU
$k = 3$ (± 1)	1,270,500	**36.74**	**15.52**	**28.85**	**68.6**
$k = 5$ (± 2)	156,085	28.26	12.04	22.26	67.05
$k = 7$ (± 3)	18,689	27.84	08.44	19.07	67.3

Table 2: Impact of Kernel Size on GEC performance. Dataset size represents the number of sentences where a matching context was found in the clean corpus.

As shown in Table 2, increasing the kernel size led to a linear degradation in performance. This is directly correlated with the massive reduction in yieldable data; $k = 7$ produced only ∼1.5% of the data volume compared to $k = 3$. Consequently, we selected $k = 3$ as the optimal trade-off between local context awareness and data availability.

4.3 Ablation 2: Error Sampling Strategy

Errors in the Wiki-Edits dataset follow a "Natural" long-tail distribution, where simple errors (e.g., adposition drops) are vastly more frequent than complex morphological shifts. We compared **Natural Sampling** against **Temperature Sampling** ($\tau = 0.5$), which flattens the distribution to give rare error patterns higher representation.

For this experiment, we held **Kernel ($k = 3$)** and **Error Density (Single Edit)** constant. Crucially, to ensure a fair comparison, the dataset size for Temperature Sampling was deliberately constrained to match the Natural Sampling size (≈ 1.27M sentences) using similar implementation to logic detailed in Appendix G.

Strategy	P	R	$F_{0.5}$	GLEU
Natural Sampling	36.74	**15.52**	28.85	68.6
Temperature Sampling	**40.50**	15.00	**30.22**	**69.08**

Table 3: Impact of Sampling Strategy. Both datasets contained ≈ 1.27M sentences.

Temperature sampling yielded a notable improvement in Precision and $F_{0.5}$ (Table 3). By upsampling rarer morphological error patterns, the model learned to generalize better to diverse error types, rather than overfitting to high-frequency surface edits.

4.4 Ablation 3: Error Density (Single vs. Multi-Edit)

Real-world texts often contain multiple errors per sentence. To support **Multi-Edit** generation, we engineered our infliction pipeline to apply errors iteratively from **right-to-left** to preserve index integrity. We compared a dataset with exactly one error per sentence against one where the number of errors follows a normal distribution using mean and standard deviation derived from Gold statistics (inspired by (Grundkiewicz et al., 2019)).

For this experiment, we held **Kernel ($k = 3$)** and **Sampling Strategy (Temperature)** constant.

Configuration	P	R	$F_{0.5}$	GLEU
Single Edit	**40.50**	**15.00**	**30.22**	**69.08**
Multi-Edit (Normal Dist.)	30.61	09.76	21.45	67.6

Table 4: Impact of Error Density on GEC performance.

Even though our Gold Test set contains a significant proportion of multi-edit sentences (827 multi-edit vs. 805 single-edit pairs), the Multi-Edit model performed worse (Table 5). We hypothesize that this is due to **infliction intensity**: applying a normal distribution of errors to the relatively short sentences typical of our corpus often obscures the semantic context required for reconstruction. Facing such high noise levels during training, the model struggles to learn precise correction mappings, leading to a notable drop in Precision (from 40.50 to 30.61). Consequently, we utilized the **Single Edit** configuration for the final corpus.

4.5 Resulting synthetic GEC Corpora

Based on the optimization study, we generated our final synthetic training corpus using **Kernel $k = 3$**, **Temperature Sampling**, and **Single Edit** injection of length 1.27 million sentence pairs.
To foster reproducibility and further research in Urdu NLP, we intend to release the **Synthetic Training Corpus** (1.27M pairs).

5 Main Experiments

5.1 Baselines and Comparisons

To validate the efficacy of our proposed **Wiki-Edit Inflection** pipeline, we compare it against three distinct baselines, which are regarded as state-of-the-art in synthetic error generation:

1. **Zero-Shot Baseline:** The raw pre-trained model without any GEC-specific fine-tuning. This serves as a control to measure the model's inherent ability to rewrite text.

2. **Random Noise Injection:** We adapted the noise generation strategy proposed by Grundkiewicz et al. (2019). We generated a synthetic dataset of 1.27M sentence pairs by applying probabilistic edit operations (insertion, deletion, substitution, swap) to the clean Makhzan corpus. While maintaining the statistical error distribution

of the original study, we utilized Levenshtein distance for word substitutions to account for the lack of Urdu spell-checking resources (see implementation details in Appendix G). This tests whether our linguistically motivated "Kernels" provide value over stochastic noise.

3. **Neural AEG (Back-Translation):** Back translation is a widely used data augmentation technique in NLP where monolingual target-language text is translated back into the source language to create synthetic parallel data, thereby improving model robustness and performance in low-resource or error-correction settings. This approach has been identified as an effective artificial data generation method in recent surveys of grammatical error correction and translation research Bryant et al. (2023). Following Koyama et al. (2021), Rei et al. (2017), and Htut and Tetreault (2019), we trained a sequence-to-sequence model to generate errors. We fine-tuned an **mT0-large** model on the *Correct* $\rightarrow$ *Incorrect* direction using the raw Wiki-Edits data and used it to corrupt the clean Makhzan corpus. Then, following Xie et al. (2018), we generated five predictions for each sentence with varying noise to equalize the dataset size. This compares our rule-based linguistic inflection against a purely neural approach. Then we trained the correction model using the same hyperparameters (listed in Appendix F) as all our other models. We did not use a generative model as used by Luhtaru et al. (2024) for the purposes of a fair comparison. We hypothesize the low scores to be the result of non-grammatical edits unfiltered in raw Wiki-Edits data, whereas our approach intrinsically performs the required filtering.

4. **Raw Wiki-Edits:** We fine-tune directly on the **Wiki-Edits Dataset**, comprising approximately 240,000 parallel sentence pairs 3.1.1 to test if our synthetic amplification provides benefits over the natural source data.

5.2 Models and Implementation

We evaluated our finalized synthetic data on three competitive multilingual architectures to demonstrate model-agnostic gains:

Approach	P	R	$F_{0.5}$	GLEU
Zero-Shot	21.52	**25.08**	22.15	45.28
Random Noise	27.70	09.24	19.79	67.62
Neural AEG	10.51	03.04	07.05	62.58
Raw Wiki-Edits	20.74	05.40	13.22	66.02
Our Approach	**40.50**	15.00	**30.22**	**69.08**

Table 5: Our approach compared with baselines.

- **mT0-large** (1.2B parameters): An instruction-tuned variant of mT5.
- **NLLB-3.3B**: A massive multilingual translation model.

All models were fine-tuned using the **Adafactor** optimizer. Specific hyperparameters for all experiments are detailed in Appendix F. Results are reported using the metrics defined in Section 4.1.

6 Results and Analysis

This section presents the empirical results of our experiments.

6.1 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 6.

The results in Table 6 reveal a substantial and consistent improvement across all metrics for both models after being fine-tuned on our synthetic corpus. The low baseline scores, particularly for NLLB, confirm that even powerful multilingual models lack specialized GEC capacity for Urdu out-of-the-box. Fine-tuning provides a dramatic performance boost.

7 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

Table 6: Category-wise $F_{0.5}$ and GLEU scores on the gold test set.

Model	Variant	Error Categories											
		Verb & Aux		Noun & Pron		Adpos		Mod & Misc		Ortho		Overall	
		$F_{0.5}$	GLEU	$F_{0.5}$	GLEU	$F_{0.5}$	GLEU	$F_{0.5}$	GLEU	$F_{0.5}$	GLEU	$F_{0.5}$	GLEU
MT0	large (baseline)	24.27	45.35	16.35	43.71	27.10	51.21	22.20	47.74	13.44	36.20	22.15	45.28
	large (finetuned)	32.74	67.62	18.56	66.35	49.86	78.03	8.36	73.89	23.20	70.63	30.22	69.08
NLLB	3.3B (baseline)	7.12	23.29	3.88	23.07	11.05	23.33	11.38	22.72	2.07	30.19	7.00	23.51
	3.3B (finetuned)	29.86	66.04	16.84	62.83	37.68	74.48	14.29	71.73	13.01	66.43	26.12	66.66

models like mT0 and NLLB 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. Furthermore, given the substantial morphosyntactic similarities Urdu shares with other Indo-Aryan languages such as Hindi, Punjabi, and Sindhi (Butt and Ahmed, 2011; Mangrio et al., 2021), we hypothesize that our data generation pipeline is transferable to this broader linguistic family. Provided that prerequisite resources such as reliable POS taggers and dependency treebanks are available, this methodology can be effectively adapted to develop robust GEC systems for other low-resource Indic languages.

7.1 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.

8 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/.

Urduhack: A library for urdu natural language processing. version 1.0.3. Accessed: 2026-01-05.

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.

Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. 2023. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 49(3):643–701.

Miriam Butt and Tafseer Ahmed. 2011. The redevelopment of indo-aryan case systems from a lexical semantic perspective. *Morphology*, 21(3):545–572.

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.

Mariano Felice and Zheng Yuan. 2014. Generating artificial errors for grammatical error correction. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 116–126, Gothenburg, Sweden. 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.

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

Phu Mon Htut and Joel Tetreault. 2019. The unbearable weight of generating artificial errors for grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 478–483, Florence, Italy. Association for Computational Linguistics.

Sudhanshu Kasewa, Pontus Stenetorp, and Sebastian Riedel. 2018. Wronging a right: Generating better errors to improve grammatical error detection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4977–4983, Brussels, Belgium. Association for Computational Linguistics.

Aomi Koyama, Kengo Hotate, Masahiro Kaneko, and Mamoru Komachi. 2021. Comparison of grammatical error correction using back-translation models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 126–135, Online. 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.

Agnes Luhtaru, Taido Purason, Martin Vainikko, Maksym Del, and Mark Fishel. 2024. To err is human, but llamas can learn it too. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12466–12481, Miami, Florida, USA. Association for Computational Linguistics.

Shirong Ma, Yinghui Li, Rongyi Sun, Qingyu Zhou, Shulin Huang, Ding Zhang, Li Yangning, Ruiyang Liu, Zhongli Li, Yunbo Cao, Haitao Zheng, and Ying Shen. 2022. Linguistic rules-based corpus generation for native Chinese grammatical error correction. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 576–589, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Riaz Ahmed Mangrio. 2016. *The Morphology of Loanwords in Urdu: The Persian, Arabic and English Strands*. Cambridge Scholars Publishing, Newcastle upon Tyne. Accessed on January 5, 2026.

Riaz Ahmed Mangrio, Muhammad Javed Iqbal, Zafeer Hussain Kiani, and Rashida Imran. 2021. Morpho-phonological similarities in indo aryan languages: A descriptive account. *Kashmir Journal of Language Research*, 23(2). Published online March 21, 2021.

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*.

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*.

Marek Rei, Mariano Felice, Zheng Yuan, and Ted Briscoe. 2017. Artificial error generation with machine translation and syntactic patterns. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 287–292, Copenhagen, Denmark. Association for Computational Linguistics.

Grigori Sidorov, Anubhav Gupta, Martin Tozer, Dolors Catala, Angels Catena, and Sandrine Fuentes. 2013. Rule-based system for automatic grammar correction using syntactic n-grams for English language learning (L2). In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 96–101, Sofia, Bulgaria. Association for Computational Linguistics.

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.

Ziang Xie, Guillaume Genthial, Stanley Xie, Andrew Ng, and Dan Jurafsky. 2018. Noising and denoising natural language: Diverse backtranslation for grammar correction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 619–628, New Orleans, Louisiana. Association for Computational Linguistics.

Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386, San Diego, California. Association for Computational Linguistics.

Appendices

A Raw Wiki-Edits Cleaning

To ensure the quality of the "Wiki-Edits Dataset," the raw data extracted from Wikipedia revision histories were subjected to the following cleaning pipeline:

1. **Character Filtering:** Removal of extra whitespaces and non-Urdu characters (e.g., emojis or unsupported symbols).
2. **Noise Filtering:** Removal of sequences containing a high percentage of numeric or non-alphabetical characters, as these usually represent metadata or formatting code rather than natural language.
3. **Deduplication:** Removal of extremely similar sentence pairs to prevent data leakage and redundancy.
4. **Unicode Normalization:** Translation of non-Urdu Unicode characters to the standard Urdu range (e.g., converting Arabic-specific Unicode points to their Urdu equivalents) and translation of English numerals to Urdu numerals using normalization provided by the UrduHack (urd) Toolkit.

B Linguistic Resources

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.

C Alignment Cost Function

Our alignment algorithm utilizes a weighted cost function to prioritize grammatical changes over semantic substitutions. The 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.[1]
- 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 for minorcounting minor for spelling variations.

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

[1] 0.499 is chosen to be just below the threshold of 0.5 to prioritize lemma matches over POS mismatches in edge cases

D Detailed Annotation Schema

This appendix details the data structures used to implement the kernel-based error representation described in Section 3.1.2. The schema is designed to be flexible for any kernel size k, though we illustrate using the default $k = 3$.

D.1 Kernel Definition and Placeholders

To handle boundary conditions, the "kernel" utilizes a specific placeholder tag, `'%'`, to handle edge cases:

- **Standard Context:** For Substitution and Insertion errors, the kernel is a list of k UPOS tags (e.g., [UPOS_left, UPOS_middle, UPOS_right]). If the edit occurs at the start or end of a sentence, the missing neighbor is replaced with `'%'`.
- **Deletion Context:** Since a Deletion error represents a "gap" between two words, the middle element is always the placeholder. The structure is strictly [UPOS_left, `'%'`, UPOS_right] (for $k = 3$).

D.2 JSON Annotation Objects

For every observed edit, a structured JSON object is created. The specific fields vary by error type to optimize for the distinct context requirements of each operation (e.g., storing neighbor features for Indels but not for Substitutions).

Substitution (S) An annotation is created only if both the incorrect and correct words pass the OOV check.

- type: 'S'
- kernel_upos: The list of three UPOS tags forming the syntactic frame.

- incorrect_feats: A UPOSFeats object containing the UPOS and morphological features (e.g., Gender=Masc|Number=Sing) of the **source** word.
- correct_feats: A UPOSFeats object containing the features of the **target** word.
- occurrence: A counter for this specific morphological transition under this kernel.

Deletion (D) An annotation is created if the deleted word and its neighbors pass the OOV checks. Note that, unlike Substitution, we store the morphological features of the *neighbors* to ensure strict context matching.

- type: 'D'
- kernel_upos: The three-element kernel with '%' in the middle.
- kernel_feats: A list of UPOSFeats objects for the left and right context neighbors.
- deleted_words: The list containing the exact surface word that was deleted.
- occurrence: Frequency counter.

Insertion (I) An annotation is created if the inserted word and the surrounding context in the correct sentence pass the OOV check.

- type: 'I'
- kernel_upos: The list of three UPOS tags.
- kernel_feats: A list of UPOSFeats objects for all three words (Left, Inserted, Right) to capture the full morphological agreement context.
- inserted_word: The exact surface word that was inserted.
- occurrence: Frequency counter.

D.3 Storage Structure

To allow for efficient querying during the infliction phase, these objects are stored in a dictionary keyed by the syntactic frame:

- **Key:** The string representation of the kernel_upos list (e.g., "['NOUN', '%', 'VERB']").
- **Value:** A list of all unique annotation objects (as defined above) that share that syntactic frame.

Figure 2 visualizes how these objects are serialized in the final JSON output.

E Infliction Mechanism Details

E.1 Lemma Dictionary Construction

To enable the **Lemma Constraint** described in Section 3.1.3, we required a mapping that could answer the question: *"For a given root lemma, what are all the valid morphological surface forms that exist in standard Urdu?"*

We constructed this knowledge base by iterating through every sentence in the **UD 2.12 Urdu Treebank** (Bhat et al.; Palmer et al., 2009). Since this treebank is manually annotated by linguists, it serves as a ground-truth source for valid morphological variation. The construction process was as follows:

1. We iterated through all sentences in the treebank.
2. For every token, we extracted its **Lemma**, its **Surface Form**, and its **Morphological Features** (e.g., Case, Gender, Number).
3. We populated two dictionaries; One which stores all surface forms of a lemma, and two where the key is the *word* which can be a lemma, or a surface form, and the value is a list of observed *Feature Set*.

Figure 3 visualizes the first dictionary structure. Figure 4 shows the second diagram. Wenary structure; we use their combination in our infliction algorithm to dynamically generate the "incorrect" version of a word found in the clean corpus. For example, if the clean corpus contains the masculine verb form *karta* (lemma: *kar*), and the error pattern dictates a switch to feminine, we query the first dictionary to get the key *kar* and query the second dictionary to retrieve the valid feminine form *karti*, provided it was observed in the treebank with the required feature set.

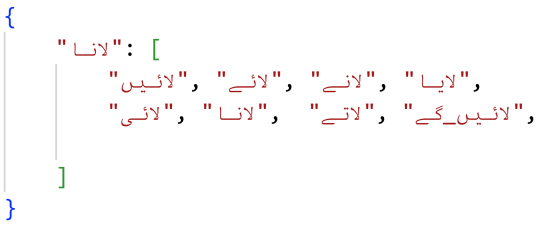

Figure 3: Structure of the pre-compiled word-to-lemma dictionary. The lemma serves as the key, pointing to a list of all its derived forms observed in the treebank.

```
{
    "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 the UD 2.12 treebank data represented as a dictionary. This is used during the Candidate Search phase to ensure that a target incorrect form corresponds to a valid grammatical usage.

E.2 Insertion and Deletion Context Validation

While Substitutions rely on lemma lookups, Insertion and Deletion infliction relies on strict context matching. To ensure high fidelity:

- **Pre-processing:** We apply character normalization to the clean text and run the Stanza pipeline to generate UPOSFeats objects.
- **Sliding Window:** We utilize a sliding window of size k (set to $k = 3$ for our primary experiments). To optimize performance, we compute a hash of the k-gram UPOS tag sequence to instantly retrieve relevant error patterns from the Annotations Database.
- **OOV Safety:** Before any deletion or insertion is finalized, we perform an Out-of-Vocabulary (OOV) check on the target word. This prevents the system from learning or propagating typos or rare spellings that may have existed in the noisy Wikipedia source.

F Training Hyperparameters

All neural models in this work were trained using a consistent set of hyperparameters to ensure fair comparison across experiments. These values were selected based on preliminary validation experiments and computational constraints.

Hyperparameter	Value
Maximum sequence length	128
Batch size	8
Gradient accumulation steps	4
Effective batch size	32
Optimizer	Adafactor
Learning rate	5×10^{-6}
Number of epochs	1
Mixed precision	bfloat16 (bf16)

Table 7: Training hyperparameters used for all model fine-tuning experiments.

G Random Noise Implementation

Our random noise baseline follows the statistical noise injection method described by Grundkiewicz et al. (2019). We applied word-level and character-level noise to clean sentences with the following parameters:

Error Rates: For each sentence, an error probability p was sampled from a normal distribution $\mathcal{N}(\mu, \sigma)$. We set $\mu = 0.2$ based on the empirical error rate observed in our Gold Test Set, and $\sigma = 0.2$ following the reference study.

Operations: Edits were applied based on the operation distribution: Substitution (0.7), Deletion (0.1), Insertion (0.1), and Swap (0.1). Furthermore, character-level noise (typos) was applied to 10% of tokens.

Confusion Sets: The reference study utilizes Aspell to generate confusion sets for word substitutions. Due to the lack of a robust phonetic spellchecker for Urdu, we generated confusion sets using Levenshtein Edit Distance.

For any given word, candidates were selected from the vocabulary if they were within an edit distance of 2. In cases where no Levenshtein neighbors were found, the system fell back to random word substitution.

```
{
  "['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 the Final Annotation Data Structure showing a Substitution object (top) and a Deletion object (bottom).