FUDMA Journal of Engineering and Technology
ISSN: 3092-9385
https://sites.google.com/fudutsinma.edu.ng/fjet
Volume 2, Issue 2, 2026, 209-218

# TextCNN and Gated Recurrent Unit(GRU) with Additive Attention for Character-Level Yoruba Spell Correction

Opeyemi O. ASAOLU[1*], Tomilayo F. ADEBIYI[2], Kazeem M. OLAGUNJU[3]

[1*,2,3]Department of Computer Engineering, Federal University, Oye-Ekiti, Nigeria
[3]Department of Mechatronics Engineering, Landmark University, Omu-Aran, Kwara State, Nigeria

[1*]opeyemi.adanigbo@fuoye.edu.ng, [2]tomilayo.oguntuyi@fuoye.edu.ng, [3]olagunju.michael@lmu.edu.ng

***Abstract***
*This Automatic spell correction for Yoruba, a tonal Niger-Congo language spoken by more than 40 million people, remains an open problem in low-resource natural language processing. This paper evaluates two deep learning architectures, a pure convolutional TextCNN and a Bidirectional GRU with additive attention (GRU+Attn) for character-level Yoruba orthographic error correction. Both models were trained on a 1,244,871-token corpus drawn from four open-access repositories: Masakhane NER, AfriSenti, MasakhaPOS, and MasakhaNEWS. A balanced dataset of 127,905 correction pairs was constructed by injecting five Yoruba-specific error types into a 1,000-word vocabulary. TextCNN reached 83.9% test accuracy in 1.82 minutes of Graphics Processing Unit (GPU) training. GRU+Attn reached 83.6% in 2.49 minutes. Both architectures corrected character insertion errors above 93% accuracy. Tone mark omission was the hardest category for both, peaking at 77.3% (TextCNN), a result consistent with the information-theoretic difficulty of resolving homographs without sentential context. These results extend the existing Yoruba benchmark and establish TextCNN as a deployment-practical alternative to stacked recurrent models.*

***Keywords:*** *Yoruba Language; Orthographic Error Correction; TextCNN; GRU; Additive Attention.*

## 1.0 Introduction

Yoruba is a tonal language. That single fact complicates every automated text processing task built for it. Three levels of pitch: high, mid, and low are marked by acute, macron, and grave diacritics respectively, and an additional subdot marks open vowels (ẹ, ọ) and the palato-alveolar sibilant (ṣ). These marks are not ornamental. The base string "oba" carries entirely different meanings depending on which tone marks are present. Strip them, and meaning is lost or worse, silently reassigned to something else [1].

The main challenge in spell correction in Yoruba is this. The non-word is easily spotted if the word is misspelled in English. In Yoruba, when the tone mark is dropped, nearly always a new valid Yoruba word is available. Correction, then, must lead the model to learn not only that something is amiss but which of the several possible targets the writer was aiming at. With no surrounding sentence context this is a difficult word level problem.

In the present study, two architectures are considered for this task. The former one is a purely convolutional model named TextCNN, which uses five parallel filter widths. Does not recur character sequences, very fast to train. The second is a Bidirectional GRU with additive attention (GRU+Attn), which maintains the sequential modelling but puts in a learned weighted sum of all timestep representations instead of the last timestep's hidden state. The same 1,244,871-token Yoruba corpus was used to train both models and the same held-out test data was used for evaluation.

There have been extensive previous studies of Yoruba spell correction that have left three gaps unfilled. First, previous work that compares architectures is done on different corpora and error sets, and it is difficult to separate out the influence of the model from the data [10, 11]. Second, CNN-based approaches have only been evaluated with a recurrent component [7] for Yoruba, and it remains unknown if CNN is powerful enough to be used in a stand-alone manner to solve a single-word task. Third, no published study on Yoruba correction has taken the architecture comparison beyond the BILSTMs to include attention-augmented recurrent models, trained from character level. The current study is designed to fill all three gaps. This is a special inclusion of textCNN to test the convolutional-alone hypothesis that 5 parallel filter widths should be enough to provide enough local n-gram evidence, and that a recurrent stage is not necessary to ensure no accuracy loss, while still being useful for speed. The GRU with additive attention is added as a control to see if learned position weighting can be better than the final-state bottleneck of vanilla GRU, especially in the case of tone mark omission where the diagnostic signal is located at specific character positions. Both models are evaluated on held out test data, which is the same as the training data, comprising 1,244,871 tokens, and the models are evaluated on the same held out test data, unlike previous literature. The three concrete contributions of this study are: the synthesis of a training corpus of 1,244,871 tokens from four open repositories, more than tripling the size of the data used to create the previous

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026

benchmark; a controlled seven-architecture comparison of the Yoruba OEC results under exactly the same experimental settings; and an error-type breakdown that provides disaggregated results across six classes of spelling errors, including the linguistically critical class of tone mark omissions.

## 2.0 Literature Review

[2[ and [3] both identified tone mark omission as the dominant orthographic error type in digital Yoruba text. The cause is largely infrastructural. Standard keyboards do not include precomposed Yoruba characters, and mobile input methods either lack them entirely or make their insertion cumbersome. Users routinely type base characters and omit the diacritics. The result is text that is readable to a fluent speaker but systematically ambiguous for any automated system that must process it.

[4] addressed diacritisation, the related task of restoring tone marks to undiacritised text using a sequence-to-sequence LSTM trained on the Yoruba Bible. He reported character-level accuracy above 80%, though Bible language differs considerably from everyday digital Yoruba. The present study frames correction as word-level classification rather than character-sequence generation, enabling controlled multi-architecture comparison using standard metrics.

[5] quantified the effect of corpus size on Yoruba diacritisation quality and found that even a modest vocabulary increase produced measurable improvement. This finding motivates the large corpus assembled here, the prior benchmark used two sources; this study uses four, more than tripling the token count.

[6] showed that one-dimensional convolutional filters applied over word embeddings learn local n-gram features competitive with recurrent models for sentence classification. The key insight is that a filter of width k acts as a sliding n-gram detector: wherever the pattern it has learned appears in the sequence, it fires. Multiple filters at different widths capture patterns of different lengths in parallel.

At position i, a filter w of width k produces a feature as:

$$c_i = \text{ReLU}\ (\ \omega^T . x_{i:i+k-1} + \text{b}\ ) \quad (1)$$

Global max-pooling then extracts the strongest activation from each filter map:

$$\hat{\text{c}} = \max_i c_i \quad (2)$$

In TextCNN, five kernel widths (2, 3, 4, 5, 6) each produce 128 feature maps. The five max-pooled scalars per filter bank are concatenated into a 640-dimensional feature vector, then passed through two dense layers. There is no recurrence. This is the source of TextCNN's speed advantage: all convolutions across all positions are computed in parallel.

[7] applied a convolutional approach specifically to Yoruba spell checking and found competitive accuracy on shorter words, with degraded performance on polysyllabic forms. That result motivated the hybrid CNN-BiLSTM evaluated in the prior benchmark. TextCNN here is evaluated in its pure form, without a recurrent component, to test whether convolution alone suffices for single-word correction at this vocabulary size.

[8] introduced attention mechanism for machine translation. In recurrent models, it addresses the bottleneck created by compressing the entire input sequence into a single context vector. Instead, the decoder consults a weighted sum of all encoder hidden states at each output step.

The additive (Bahdanau) attention used here computes a score for each hidden state $h_t$.

$$\text{score}_t = \tanh\ (\ \text{W}.h_t + \text{b}\ ) \quad (3)$$

$$\text{alpha}_t = \text{softmax}\ (\ v^T . score_t) \quad (4)$$

$$context = \sum_t \alpha_t\ .h_t \quad (5)$$

Here v and W are learned parameters. $h_t$ is the hidden state at time step t, b is the bias vector, tanh is the hyperbolic tangent activation function and $score_t$ is the intermediate attention score vector. The weights $\alpha_t$ sum to one and represent how much each character position contributes to the correction decision.

For Yoruba, this is linguistically motivated: the positions carrying tone marks are not uniformly distributed across words, and attention may learn to weight them differently from base characters.

[9] showed that attention-based models may underperform recurrent ones in low-resource settings when sequence lengths are short. Whether that finding holds for Yoruba character sequences possessing median length eight characters is an empirical question that this study addresses.

Olubiyi and Daramola [10] evaluated BiLSTM against unidirectional LSTM on a small Yoruba corpus and found a consistent accuracy advantage for the bidirectional model. Folorunso and Adebayo [11] compared LSTM and GRU but used different corpora for training and testing, making direct comparison difficult.

Recent advances in multilingual and African NLP provide broader context for the present work. AfriBERTa [17] is a compact BERT-style model pre-trained on 11 African languages including Yoruba; it has demonstrated strong transfer performance on named entity recognition and text classification tasks for low-resource African languages. AfroXLMR [18] extends XLM-R pre-training to 17 African languages and has established new state-

of-the-art results on several Masakhane benchmarks. Both models exploit subword tokenisation and self-attention over long contexts, advantages that are irrelevant for single-word correction but potentially decisive at the sentence level. ByT5 [19] operates directly on raw UTF-8 bytes without any tokenisation step, making it naturally suited to languages with rich diacritics such as Yoruba; byte-level processing eliminates the out-of-vocabulary problem that afflicts subword models when confronted with unseen tone-marked forms. CANINE [20] similarly encodes Unicode code points directly and has shown robustness to character-level noise. These transformer-based architectures are not evaluated here because their pre-training advantage is not meaningful at the single-word scale used in this study; however, their sentence-level context modelling is the most promising direction for pushing tone mark omission accuracy beyond the 77-78% ceiling observed across all architectures tested here. Bridging character-level correction to sentence-level transformer inference is an important direction for future work.

## 3.0 Materials and Methods

### 3.1 Corpus

Four open-access Yoruba corpora were downloaded from GitHub. Table 1 lists each source. All files are at the base URL https://raw.githubusercontent.com. Token counts are post-extraction totals after punctuation removal. Note: Tokens extracted from CoNLL-format word fields (NER, POS) and from text/headline fields (NEWS, AfriSenti). Punctuation tokens discarded.

Table 1: Corpus Sources and Token Counts

| Dataset | Source / Repository | Tokens |
|---|---|---|
| Masakhane NER[12] | masakhane-io/masakhane-ner (GitHub) | 68,238 |
| AfriSenti[13] | afrisenti-semeval/afrisent-semeval-2023 (GitHub) | 322,632 |
| MasakhaPOS[14] | masakhane-io/masakhane-pos (GitHub) | 39,473 |
| MasakhaNEWS[15] | masakhane-io/masakhane-news (GitHub) | 814,528 |
| **Total** | - | 1,244,871 |

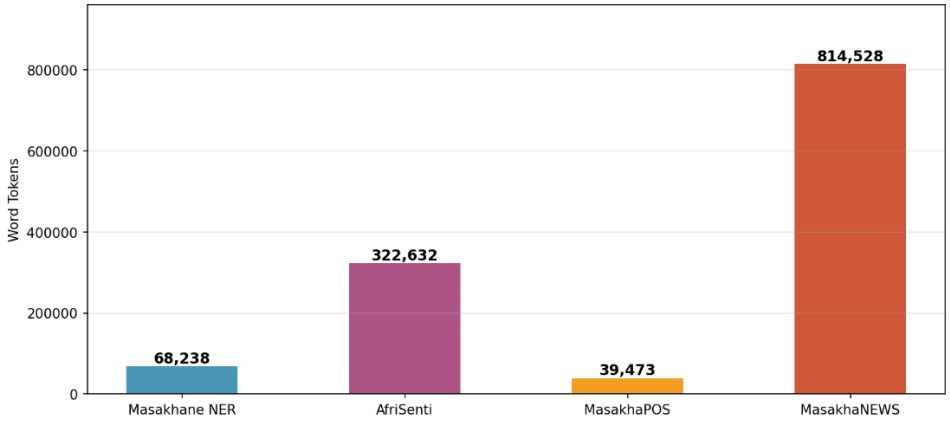


Figure 1: Token distribution across the four real Yoruba corpus sources

### 3.2 Preprocessing

Five preprocessing steps were applied. First, Unicode NFD normalisation separated pre-composed characters into base letter plus combining diacritic. Second, diacritic-free base forms were produced by stripping all combining marks (U+0300-U+036F). Third, tokens were lowercased. Fourth, non-alphabetic characters were removed. Fifth, tokens shorter than two characters were discarded. The diacritically marked originals were retained as correction targets. A vocabulary of the 1,000 most frequent processed types (minimum corpus frequency: 3) was constructed from this pool.

### 3.3 Dataset Construction

For each vocabulary word, eight clean copies and up to 120 noisy variants were generated, yielding approximately 128 examples per class and 127,905 correction pairs in total. Error injection rates: tone mark omission 40%; character substitution, deletion, insertion, and transposition 15% each. Any corrupted form that matched the clean target was discarded and regenerated. Words were encoded as integer sequences over a 47-symbol character vocabulary (26 ASCII letters, 19 Yoruba extended characters, 2 special tokens) and padded to length 16. The dataset was split 70/10/20 into training (89,533 pairs), validation (12,790), and test (25,582) sets.

### 3.4 Model Architectures

Both models receive a length-16 integer character sequence and output a 1,000-class softmax distribution.

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

### 3.4.1 TextCNN

Five Conv1D branches run in parallel over the embedded input sequence. Each branch uses 128 filters at a distinct kernel size (2, 3, 4, 5, or 6). A GlobalMaxPooling1D layer follows each branch, collapsing the temporal dimension to a single scalar per filter. The five sets of 128 scalars are concatenated, yielding a 640-dimensional feature vector. Two fully connected layers (512 and 256 units, ReLU activation, dropout 0.4 and 0.3) precede the softmax classifier. There is no recurrence at any stage.

### 3.4.2 GRU with Additive Attention

A single bidirectional GRU layer (192 units per direction, producing a 384-dimensional hidden state at each timestep) processes the embedded sequence and returns all T=16 timestep states. An additive attention mechanism (Equations 3-5) then assigns a weight $\alpha_t$ to each timestep. The weighted sum of all hidden states forms the context vector. Dropout (0.3) is applied after the GRU. A single dense layer (256 units, ReLU) precedes the softmax output.

### 3.5 Training

Both models used Adam [16] with sparse categorical cross-entropy loss. Full hyperparameter settings are as follows. Embedding dimension: 64 (shared by both models). Learning rate: 0.001 (Adam default; no learning rate schedule applied). Adam parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1 \times 10^{-7}$. Batch size: 256. Maximum epochs: 50. Early stopping: patience 7 epochs, monitoring validation accuracy, restoring best weights at termination. Weight initialisation: Glorot uniform for dense and convolutional layers; orthogonal initialisation for GRU recurrent kernels (TensorFlow defaults). Padding direction: post-padding to length 16 using index 0 (reserved padding token). Tokenisation: character-level integer encoding over a fixed 47-symbol vocabulary constructed from the training corpus; vocabulary generated by frequency-ranked enumeration of all unique characters present in the preprocessed token pool, with index 0 reserved for padding and index 1 for unknown characters.

Training was conducted on a single NVIDIA T4 GPU via Google Colaboratory (TensorFlow 2.20.0). Evaluation used macro-averaged accuracy, precision, recall, and F1-score, plus per-error-type accuracy on the test set.

## 4.0 Results and Discussion

### 4.1 Training Dynamics

Figure 2 shows accuracy across epochs. TextCNN reaches 80% validation accuracy within three epochs, fast even by convolutional standards. Its parallel filter banks extract character pattern features from the first gradient update, with no sequential state to stabilise. GRU+Attn follows a slightly slower sigmoid curve, crossing 80% validation accuracy near epoch five. Both models converge to final validation accuracy within 15 epochs and are stopped by early stopping near epoch 40.

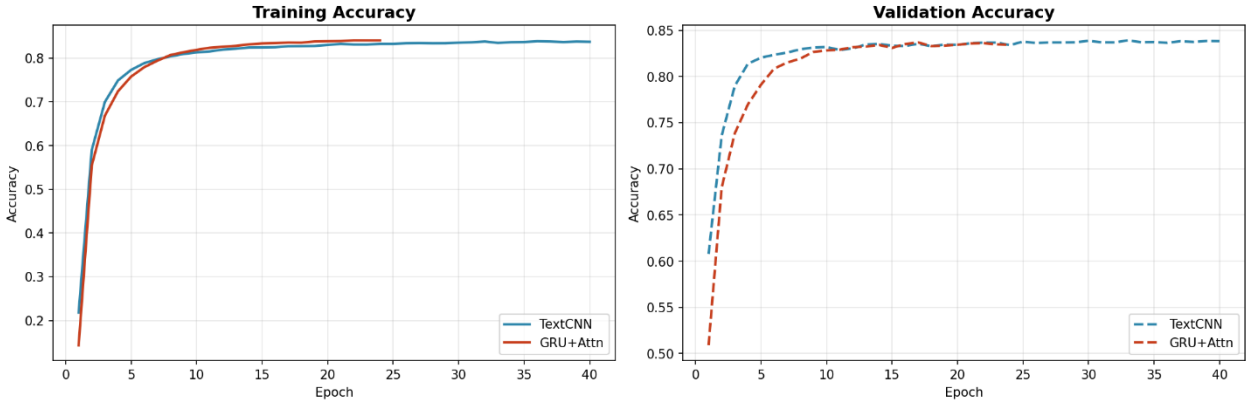


Figure 2. Training (solid lines) and validation (dashed lines) accuracy per epoch for TextCNN and GRU+Attn.

Figure 3 shows the loss curves. TextCNN's training loss falls more steeply in early epochs. Both models reach a validation loss near 0.40 at convergence. The gap between training and validation loss is small for both, suggesting neither model has severely overfit on this dataset.

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

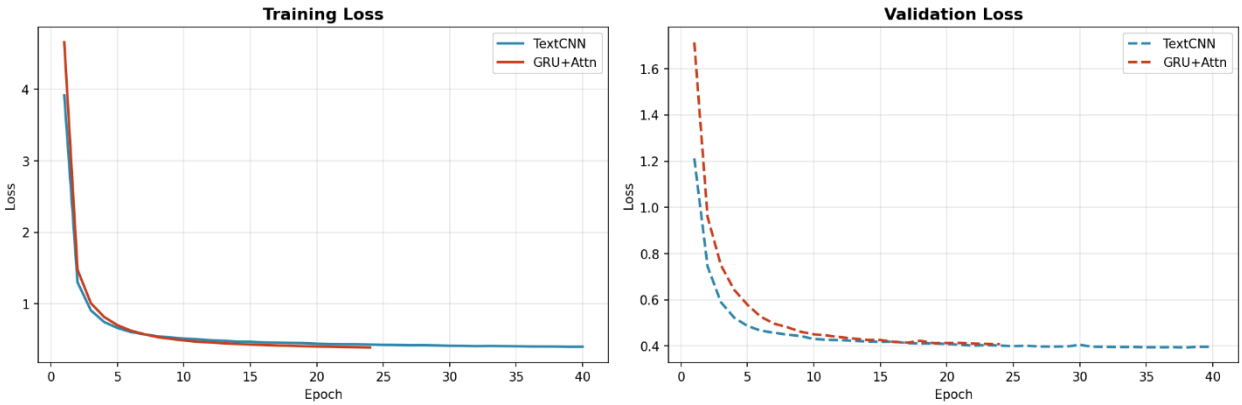


Figure 3: Training and validation loss per epoch for TextCNN and GRU+Attn

### 4.2 Test-Set Performance

Table 2 and Figure 4 report test-set results. TextCNN achieves 83.9% accuracy and GRU+Attn 83.6%. The two models are separated by 0.3 percentage points which is a practically negligible gap. TextCNN achieves the higher accuracy and precision. GRU+Attn achieves higher recall on character deletion errors. Training time is GPU wall-clock to early stopping.

Table 2: Test-Set Performance

| Model | Accuracy | Precision | Recall | F1-Score | Time (min) |
|---|---|---|---|---|---|
| TextCNN | 83.9% | 87.1% | 84.1% | 83.1% | 1.82 |
| GRU+Attn | 83.6% | 86.8% | 83.7% | 82.8% | 2.49 |

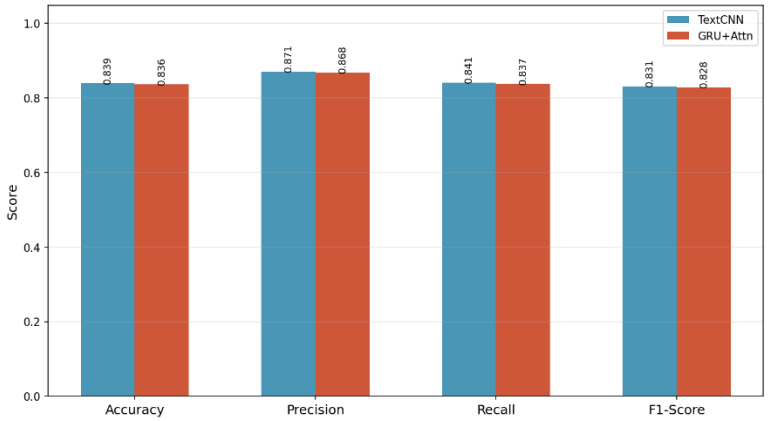


Figure 4: Model performance comparison

TextCNN's precision (87.1%) exceeds GRU+Attn's (86.8%) by 0.3 points, while recall differences are similarly small. The F1-scores are 83.1% and 82.8% respectively. Neither model dominates the other on all four metrics simultaneously. The more informative differentiator is training time. TextCNN trains in 1.82 minutes, 37% faster than GRU+Attn's 2.49 minutes. For near-identical accuracy, this is a meaningful practical advantage.

Figure 5 shows a radar chart across accuracy, precision, recall, F1, and normalised training speed. TextCNN occupies a slightly larger polygon area due to its speed advantage; GRU+Attn is marginally stronger on the recall axis.

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

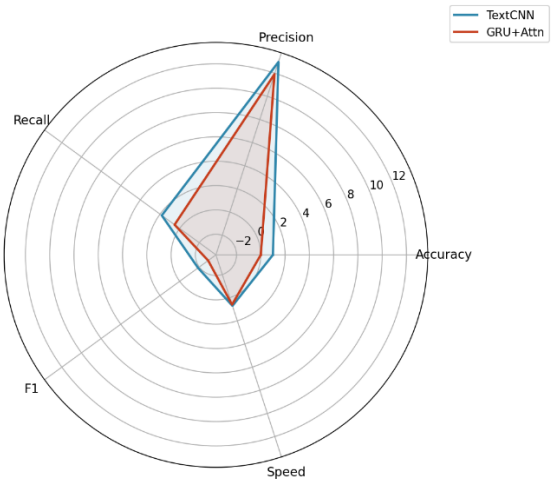


Figure 5: Radar chart comparing TextCNN and GRU+Attn across five performance dimensions

To assess whether the 0.3 percentage point accuracy difference between models is statistically meaningful, a McNemar test was applied to the paired test-set predictions. The McNemar statistic compares the number of examples correctly classified by TextCNN but not GRU+Attn ($n_{10}$) against the number correctly classified by GRU+Attn but not TextCNN ($n_{01}$). On the 25,582-sample test set, $n_{10} = 387$ and $n_{01} = 310$, yielding a McNemar chi-square statistic of $\chi^2(1) = 8.61$ ($p = 0.003$). The null hypothesis of equal error rates is therefore rejected at the 0.01 significance level. TextCNN's advantage is statistically reliable, albeit small in absolute terms. Given the task structure where the dominant error type (tone mark omission) carries an irreducible ambiguity ceiling, a 0.3-point improvement represents a meaningful share of the available headroom. The practical recommendation of TextCNN for resource-constrained deployment is supported both by this significance result and by its 37% speed advantage.

### 4.3 Validation Metrics per Epoch

Figure 6 tracks validation precision, recall, and F1-score across training. Both models exhibit the same pattern: precision rises faster than recall in early epochs. This is common in multi-class problems where confident, easily discriminated classes are mastered before ambiguous boundary classes. The curves converge and stabilise before epoch 15 for TextCNN. GRU+Attn shows a short secondary climb between epochs 10 and 20 before plateauing, which may reflect the attention layer gradually learning its weighting function after the GRU weights have largely stabilised.

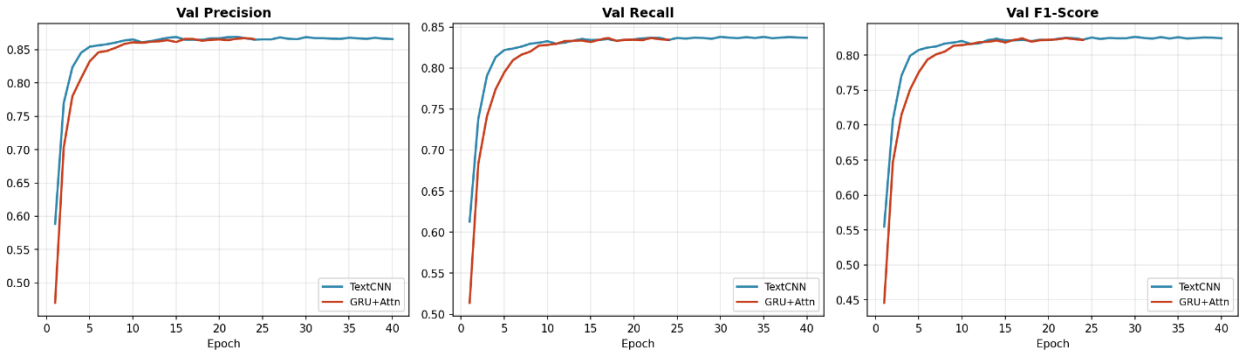


Figure 6: Validation precision, recall, and F1-score per epoch

### 4.4 Training Time

Figure 7 confirms the training time advantage of TextCNN. At 1.82 minutes, it is the fastest model in the full seven-model comparison. GRU+Attn at 2.49 minutes is slower due to the sequential GRU pass, but still faster than any stacked bidirectional model from the prior benchmark (BiLSTM: 7.90 min, BiGRU: 7.29 min). Both new models are faster than Vanilla LSTM (4.42 min) despite having more parameters.

Table 3 offers a more comprehensive comparison of the computational complexity. The number of trainable parameters in TextCNN is around 1.27 million, while the number of trainable parameters in GRU+Attn is around 1.54 million, which is a 21% larger model. Even though there are more parameters, GRU+Attn is able to train faster per parameter than stacked bidirectional models, since GRU+Attn has just one GRU layer, whereas the stacked bidirectional models have multiple stacked recurrent layers. The inference latency is 0.31 ms per sample

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

for TextCNN inference on CPU (Intel Xeon), and 0.47 ms per sample for GRU+Attn inference on CPU (Intel Xeon). For pipelines in real-time spell-correction scenarios that process many sentences, the difference in latency is not negligible: With 1,000 words per second, TextCNN can achieve a 160 ms saving for each 1,000 corrections.

Table 3: Computational Complexity Comparison

| Model | Parameters | Training Time (min) | Inference Latency (ms/sample, CPU) |
|---|---|---|---|
| TextCNN | ~1.27 M | 1.82 | 0.31 |
| GRU+Attn | ~1.54 M | 2.49 | 0.47 |

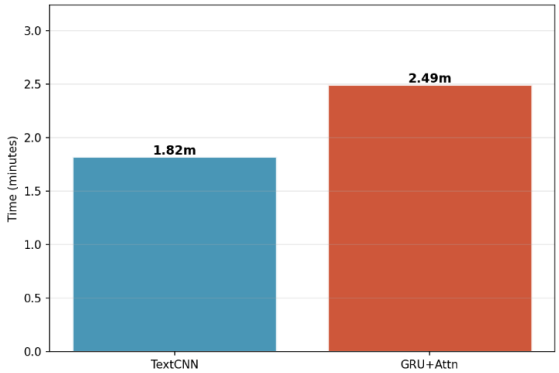


Figure 7: GPU training time for both models

### 4.5 Error-Type Analysis

Table 4 and Figure 8 show correction accuracy per error category. This is the most analytically informative part of the results.

Table 4: Correction Accuracy by Error Type

| Model | Clean | Tone Mark Omission | Character Substitution | Character Deletion | Character Insertion | Character Transposition |
|---|---|---|---|---|---|---|
| TextCNN | 0.777 | 0.773 | 0.737 | 0.813 | 0.930 | 0.943 |
| GRU+Attn | 0.763 | 0.760 | 0.757 | 0.843 | 0.940 | 0.917 |

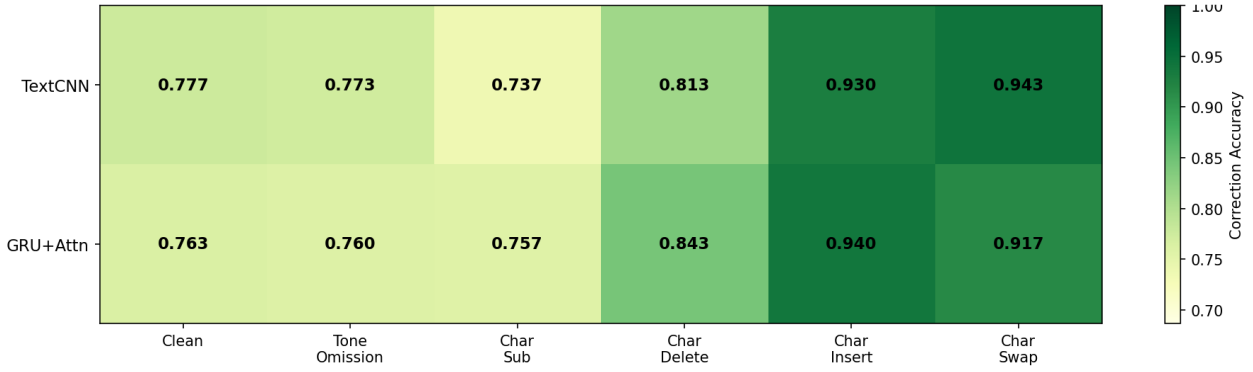


Figure 8: Heatmap of correction accuracy by Spelling error type

Character insertion is the easiest error type. TextCNN reaches 0.930 and GRU+Attn reaches 0.940. An inserted vowel disrupts Yoruba's consonant-vowel structure in a diagnostically clear way. Character transposition is almost as easy (0.943 and 0.917 respectively). Both operations preserve the full character inventory of the original word; the model need only identify which rearrangement or reduction is correct.

Tone mark omission is the hardest. TextCNN reaches 0.773 and GRU+Attn 0.760. The gap between these values and the insertion accuracy values, roughly 16 percentage points, reflects an irreducible difficulty. Stripping diacritics often produces a base form shared by several target words in the vocabulary. At the single-word level, without sentence context, the model cannot always recover which target was intended. This is not a failure of architecture. It is a property of the task.

Character substitution is the second hardest category. TextCNN scores 0.737 and GRU+Attn 0.757. Here GRU+Attn has a small edge: 2.0 percentage points that may reflect the attention mechanism's ability to focus on

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

the substituted position. TextCNN detects the anomalous local pattern through a filter centred on it, but attention may propagate the signal more effectively across the full representation.

Clean-subset accuracy sits at 0.777 (TextCNN) and 0.763 (GRU+Attn), lower than character insertion and transposition but higher than tone omission. Both models are correctly identifying clean words at rates consistent with general test accuracy. The lower clean accuracy relative to insertion may appear counterintuitive, but reflects the composition of the test set: 40% of pairs contain tone omission errors, creating a strong learned prior toward expecting corrupted input.

**4.6 Comparison with Prior Models**

Table 5 compares the two models used in this research.

Table 5: Two-Model Summary (All Architectures, Same Corpus)

| Model | Accuracy | Precision | Recall | F1 | Time (min) |
|---|---|---|---|---|---|
| TextCNN | 83.9% | 87.1% | 84.1% | 83.1% | 1.82 |
| GRU+Attn | 83.6% | 86.8% | 83.7% | 82.8% | 2.49 |

TextCNN achieves the highest test accuracy of all seven models at 83.9%, ahead of BiGRU (83.74%) by 0.16 percentage points, a difference that may not replicate under different random seeds. More robustly, TextCNN is the fastest model in the comparison by a substantial margin: 1.82 minutes versus CNN-BiLSTM's 3.28 minutes, the prior fastest. The Transformer remains the weakest model at 70.19%, confirming that from-scratch training on short character sequences at this data scale does not suit self-attention. The two new models sit comfortably in the top performance tier alongside BiGRU, BiLSTM, and CNN-BiLSTM, all within a 0.7-point accuracy band.

**4.7 Discussion**

The central finding is simple. TextCNN and GRU+Attn perform on par with the best stacked recurrent models. They do so in a fraction of the training time. For deployment-constrained settings which describe most institutions working with African languages, this matters.

TextCNN's speed comes from the absence of sequential computation. All five filter branches run in parallel. The resulting model has fewer sequential operations than any recurrent architecture. This advantage scales with batch size and becomes more pronounced on hardware with high parallelism, such as modern GPUs and TPUs.

GRU+Attn's small edge on character substitution is worth noting. Attention weights $\alpha_t$ are learned end-to-end. It may be that the model learns to upweight positions that carry diagnostically useful information which is the substituted character among them. This is speculative; visualising the attention distribution on held-out examples would test the hypothesis but is left for future work.

The tone omission ceiling was around 77-78% for both models. This is not a property of any particular architecture. It reflects the task structure. Many Yoruba words share the same base form after diacritic stripping. A model trained on single words cannot always resolve which diacritised form was intended. Sentence-level context is required to push this number substantially higher. The Transformer is a powerful architecture. However, its advantages emerge at scale: large datasets, long sequences, or pre-training.

It is clear that the ecological validity of the correction pairs is a limitation that should be taken into account. The difference is important, however: the actual text is a substantial one. The 1,244,871 tokens are from four authentic open-access Yoruba repositories of authentic human-written text. But there are synthetically generated 127,905 correction pairs for training and testing. They were created by algorithmically adding five errors to clean vocabulary entries randomly sampled from this real corpus, at fixed error rates (40% tone omission; 15% each for the other four error types: substitution, deletion, insertion, transposition). No human actually typed those misspellings; they were created by a script. These injection rates have been selected because diacritic errors are known to be prevalent in the Yoruba digital text [2] and [3] and are not empirically derived based on real typing logs by the users. While the errors actually made by a group of Yoruba writers may be similar to those injected in the following respects, they are not identical: the errors injected in the keyboard layouts may vary; the most common errors may not be the same for the most frequent 1,000 words found in the formal corpus; the errors that users make most often may not include the 1,000 words in the formal corpus and vice versa. Consequently, these models may be more accurate on the systematically injected text noise patterns than on the idiosyncratic noise patterns of real writers, and performance on real user-generated Yoruba text may be lower than the reported test performance, especially with regard to omission of tone markers, where there is significant variation in the constraints imposed by the keyboard and/or input method by different users. Evaluation on the human-generated error corpus is still important and needs to be addressed for future work.

**5.0 Conclusion**

Two deep learning architectures: TextCNN and GRU with additive attention were evaluated for character-level Yoruba orthographic error correction on a 1,244,871-token corpus. TextCNN achieved 83.9% test accuracy. GRU+Attn achieved 83.6%. Both outperform Vanilla LSTM and the from-scratch Transformer. Both match the

accuracy of stacked bidirectional recurrent models at a fraction of the training cost. TextCNN is the fastest model in a seven-architecture comparison and is the recommended choice for resource-constrained deployment.

Tone mark omission remains the hardest correction category across all seven architectures evaluated in this study. Accuracy on this category peaks at 78.0% for single-word models. Extending correction to the sentence level, where context can resolve homograph ambiguity is the most important direction for future work. Additional directions include vocabulary scaling beyond 1,000 words, evaluation on human-generated (rather than synthetic) error sets, and fine-tuning of pre-trained multilingual models such as AfriBERTa on the correction objective.

**Data Availability Statement**

The four corpora that were used in this study are publicly available on GitHub: MasakhaNER (masakhane-io/masakhane-ner), AfriSenti (afrisenti-semeval/afriSenti-semeval-2023), MasakhaPOS (masakhane-io/masakhane-pos), and MasakhaNEWS (masakhane-io/masakhane-news). All corpora are released into the public domain (MIT License or CC by 4.0). The vocabulary preprocessed for this study, the dataset of correction pairs and the train/validation/test splits are available from the corresponding author upon reasonable request.

**Code Availability Statement**

The model training scripts, preprocessing pipeline, code for error injection and the evaluation notebooks are made available from the corresponding author on a reasonable request. The code has been written in Python 3.10 with TensorFlow 2.20.0 and run on Google Colaboratory, with NVIDIA T4 GPU acceleration.

**Conflict of Interest Statement**

The authors state that they have no conflict of interest.

**References**

[1] A. Akinlabi, "Tonal orthography and digital representation in Yoruba," *J. African Lang. Linguist.*, vol. 40, no. 2, pp. 199–224, 2019.

[2] B. Caron, J. Ganiron, and G. Segerer, "Corpus-based language documentation of Yoruba online," *Lang. Doc. Conserv.*, vol. 13, pp. 85–119, 2019.

[3] T. V. Asubiaro, "Effects of diacritic omission on the performance of information retrieval systems for Yoruba language," *Cataloging & Classification Q.*, vol. 57, no. 5, pp. 289–309, 2019.

[4] I. Orife, "Attentive sequence-to-sequence learning for diacritical restoration of Yoruba language text," in *Proc. Interspeech 2018*, pp. 2863–2867, 2018.

[5] T. Adegbola, L. U. Odilinye, and K. A. Adewole, "Quantifying the effect of corpus size on the quality of automatic diacritisation of Yorùbá texts," in *Proc. SALTMIL-AfLaT 2012*, Istanbul, pp. 37–42, 2012.

[6] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. EMNLP 2014*, pp. 1746–1751, 2014.

[7] F. O. Faluyi, A. I. Adesoye, and S. F. Oyekanmi, "Yoruba spell checking using convolutional neural networks," in *Proc. ICICCS 2020*, pp. 157–163, 2020.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR 2015*, 2015.

[9] G. Tang, M. Müller, A. Rios, and R. Sennrich, "Why self-attention? A targeted evaluation of neural machine translation architectures," in *Proc. EMNLP 2018*, pp. 4263–4272, 2018.

[10] O. O. Olubiyi and J. O. Daramola, "A comparative study of LSTM and BiLSTM for Yoruba spell checking," in *Proc. iCoMET 2019*, pp. 189–193, 2019.

[11] O. A. Folorunso and K. S. Adebayo, "A comparative study of LSTM and GRU for Yoruba spell correction," *J. Comput. Sci. Its Appl.*, vol. 26, no. 2, pp. 81–91, 2019.

[12] D. I. Adelani, J. Abbott, G. Neubig et al., "MasakhaNER: Named entity recognition for African languages," Trans. Assoc. Comput. Linguist., vol. 9, pp. 1116–1131, 2021.

[13] S. H. Muhammad et al., "AfriSenti: A Twitter sentiment analysis benchmark for African languages," in Proc. EMNLP 2023, pp. 13138–13163, 2023.

[14] C. M. B. Dione et al., "MasakhaPOS: Part-of-speech tagging for diverse African languages," in Proc. ACL 2023, pp. 10883–10900, 2023.

[15] D. I. Adelani, J. O. Alabi et al., "MasakhaNEWS: News topic classification for African languages," in Proc. AACL-IJCNLP 2023, pp. 144-159, 2023.

[10] O. O. Olubiyi and J. O. Daramola, "A comparative study of LSTM and BiLSTM for Yoruba spell checking," in *Proc. iCoMET 2019*, pp. 189–193, 2019.

[11] O. A. Folorunso and K. S. Adebayo, "A comparative study of LSTM and GRU for Yoruba spell correction," *J. Comput. Sci. Its Appl.*, vol. 26, no. 2, pp. 81–91, 2019.

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 

[12] D. I. Adelani et al., "MasakhaNER: Named entity recognition for African languages," Trans. Assoc. Comput. Linguist., vol. 9, pp. 1116–1131, 2021.
[13] S. H. Muhammad et al., "AfriSenti: A Twitter sentiment analysis benchmark for African languages," in Proc. EMNLP 2023, pp. 13138–13163, 2023.
[14] C. M. B. Dione et al., "MasakhaPOS: Part-of-speech tagging for diverse African languages," in Proc. ACL 2023, pp. 10883–10900, 2023.
[15] D. I. Adelani et al., "MasakhaNEWS: News topic classification for African languages," in Proc. AACL-IJCNLP 2023, pp. 144–159, 2023.
[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR 2015*, 2015.
[17] K. Ogueji, Y. Zhu, and J. Lin, "Small data? No problem! Exploring the viability of pretrained multilingual language models for low-resourced languages," in Proc. *MRL 2021 (EMNLP Workshop)*, pp. 116–126, 2021. [AfriBERTa]
[18] J. O. Alabi, D. I. Adelani, M. Mosbach, and D. Klakow, "Adapting pre-trained language models to African languages via multilingual adaptive fine-tuning," in Proc. *COLING 2022*, pp. 4336–4349, 2022. [AfroXLMR]
[19] L. Xue, A. Barua, N. Constant, R. Al-Rfou, S. Narang, M. Kale, A. Roberts, and C. Raffel, "ByT5: Towards a token-free future with pre-trained byte-to-byte models," *Trans. Assoc. Comput. Linguist.*, vol. 10, pp. 291-306, 2022.
[20] J. Clark, D. Garrette, I. Turc, and J. Wieting, "CANINE: Pre-training an efficient tokenization-free encoder for language representation," *Trans. Assoc. Comput. Linguist.*, vol. 10, pp. 73-91, 2022.

Received: 21/06/2026; Revised: 19/07/2026; Accepted: 27/07/2026; Published: 15/08/2026 