Tagalog Diacritic Restoration using LSTM-based Autoencoders

De Leon, Ivan Yuri
Ateneo de Manila University
ivan.deleon@student.ateneo.edu

Abstract

This study investigates neural network modifications for diacritic restoration in Tagalog text using sequence-to-sequence autoencoders. Two architectural changes were introduced: a Penalty layer that constrains decoding by suppressing invalid character transformations, and a residual input connection that injects mean-pooled source embeddings into the decoder's initial hidden state. Theoretically, the Penalty layer enforces linguistic validity and reduces error propagation, while the residual pathway preserves contextual information at the cost of added computational overhead. Experimental results show that both modified models vastly outperform the Vanilla baseline, achieving nearly 8.5× higher character-level accuracy and correctly restoring over 60% of vowels, which translates to more than 30% of homographs being diacritized correctly. Latent space analysis further confirms that the modified architectures yield more separable and stable representations, achieving silhouette scores above 0.9 across more cluster sizes than the Vanilla baseline.

1 Introduction

Homographs are words that share the same spelling but have different pronunciations. In Tagalog, such words often introduce ambiguity, making accurate disambiguation a persistent challenge. For example, the word *puno* may mean *punô* (full) or *punò* (tree). Fortunately, diacritics resolve this ambiguity, but they are often omitted in modern texts, increasing confusion for both readers and NLP systems. This motivates the need for diacritic restoration to clarify homographs and improve text understanding.

To date, there are only two known attempts at Tagalog diacritic restoration. One study built a small homograph corpus and trained basic classification models (Flores, 2024), while another built a relatively larger corpus and explored transformer-based methods (Africa et al., 2025). However, both approaches rely on sentence-level classification and restricted homograph lists, which are too rigid for Tagalog. Thus, a more flexible approach is needed to handle the variety of word forms and diacritic patterns.

To mitigate the ambiguity caused by missing diacritics, this study introduces a character-level sequence-to-sequence model for diacritic restoration. The models are evaluated not only with standard accuracy metrics but also by analyzing the structure of their latent representations, providing insight into how well they capture underlying linguistic patterns. This approach provides a generalizable solution for Tagalog, with potential applications in machine translation and speech processing, where effective homograph disambiguation is essential.

2 Methodology

2.1 Dataset

The dataset consists of 162 articles from the *Sari-Samot* blog, published on the *Tagalog Ngayon* Facebook page. All texts were authored by National Artist Virgilio Almario, who frequently uses diacritics to disambiguate homographs. The articles cover a variety of topics, including literature, culture, social commentary, and the arts.

For data cleaning, only lines and sentences containing diacritized characters were retained. Each line or sentence was treated as a single entry, resulting in 4,312 entries with an average sequence length of 189 characters. The dataset was split using a 70–15–15 ratio, yielding 3,018 training samples, 647 validation samples, and 647 test samples. During model training, sentences were stripped of all diacritics, with their original diacritized forms retained as the targets for reconstruction.

To illustrate, Table 1 shows a sample input-target

pair.

Input	Target
Napanaginipan ko ang Santo Niño minsan isang gabi.	Napanaginipan ko ang Santo Niño minsan isang gabí.

Table 1: Example input and target pair for diacritic restoration. This is illustrative only.

2.2 Model Architecture

Three character-level sequence-to-sequence models were implemented to evaluate diacritic restoration performance. Each model is based on an encoder–decoder framework using LSTM,but differ in how they represent and constrain information during decoding.

1. **Vanilla LSTM-based Autoencoder (Vanilla)** – A baseline encoder–decoder model without modifications. The encoder, consisting of an LSTM followed by a linear layer, compresses the input sequence into a latent vector. The decoder then reconstructs the target sequence from this latent representation.

2. **LSTM + Penalty Layer (Penalty)** – Builds on the baseline by adding a penalty mechanism that constrains decoding. The penalty layer reduces the likelihood of generating invalid character transformations, thereby discouraging incorrect outputs during reconstruction.

3. **LSTM + Residual Input + Penalty (Residual-Penalty)** – Enhances the decoder further by incorporating a residual input connection. A mean-pooled representation of the source embeddings is added to the decoder's initial hidden state, preserving contextual information. This model also incorporates the penalty layer to ensure that only valid character transformations are produced.

All models were implemented in PyTorch using custom `nn.Module` definitions. Training was conducted with identical hyperparameters to ensure comparability across architectures.

2.2.1 LSTM-based Autoencoder

The core of the three models is an LSTM-based autoencoder. LSTMs are recurrent neural networks designed to process sequences and capture long-range dependencies through hidden states and memory cells (Hochreiter and Schmidhuber, 1997).

Unlike standard LSTM autoencoders, where the encoder and decoder are directly connected through the latent space, this architecture introduces linear projections between them. The input sequence is first processed by an LSTM encoder to produce a sequence representation. This representation is then passed through a linear layer to obtain a latent vector, which is subsequently projected via another linear layer to initialize the LSTM decoder.

The decoder reconstructs the output sequence from this projected latent representation, enabling effective sequence reconstruction while maintaining flexibility in the latent space. This is exactly the architecture of the first model (Vanilla), without any additional modifications, as illustrated in Figure 1.

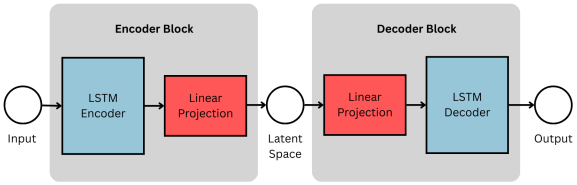

Figure 1: Illustration of the Vanilla LSTM-based autoencoder framework.

2.2.2 Penalty Layer

During prediction, the decoder produces an output vector of logits, one for each character in the vocabulary. Each logit represents a score indicating the likelihood of selecting the corresponding character as the next token in the sequence. Consequently, every character is initially considered a potential prediction. However, many character transformations are invalid. For instance, given the input *p*, only *p* is a valid output, whereas given *a*, valid outputs include *a*, *à*, *á*, or *â*.

To enforce valid predictions, prior work (Dang and Nguyen, 2020) introduced an additive penalty matrix $P' \in \mathbb{R}^{|V|\times|V|}$, where $|V|$ denotes the vocabulary size:

$$P'_{i,j} = \begin{cases} 0, & \text{if character } i \text{ can transform to } j \\ -\infty, & \text{otherwise} \end{cases}$$

This penalty layer is added to the output of the decoder block, effectively preserving only valid transformations. While it successfully eliminates invalid predictions, the hard constraint is static and

requires the loss to be computed before applying the penalty, as using infinity in computation is problematic. As a result, its influence on the learning dynamics is limited.

In this study, we propose a modified, dynamic penalty matrix $P \in \mathbb{R}^{|V|\times|V|}$. Let r denote the range of logits at each prediction step, defined as the difference between the maximum and minimum logits. Then:

$$P_{i,j} = \begin{cases} 0, & \text{if character } i \text{ can transform to } j \\ -1.2\,r, & \text{otherwise} \end{cases}$$

By subtracting $1.2\,r$ from the logits corresponding to invalid transformations, the penalty ensures that valid characters remain most probable. Unlike the static $-\infty$ approach, this dynamic penalty adapts to the distribution of logits in each batch, allowing the loss function to be computed after applying the penalty without adversely affecting gradient flow.

The second model (Penalty) extends the LSTM-based autoencoder by incorporating this penalty layer at the decoder output, as illustrated in Figure 2.

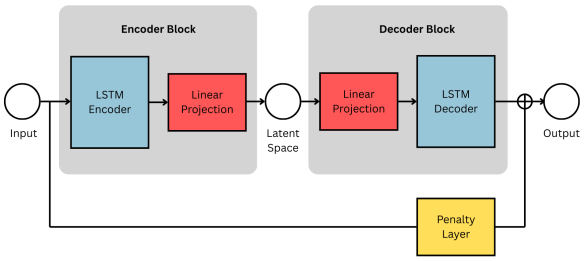

Figure 2: Illustration of LSTM-based autoencoder with penalty layer.

2.2.3 Residual Connection

Residual connections help neural networks learn by allowing later layers to access earlier representations, which stabilizes training and improves performance.

In this model, the mean-pooled embeddings of the input sequence are linearly projected to match the decoder's hidden state size and added to initialize the LSTM decoder. This gives the decoder extra context from the input, improving reconstruction and handling ambiguous character transformations.

This residual input is combined with the penalty layer in the final model variant, benefiting from both constrained predictions and richer input context, as shown in Figure 3.

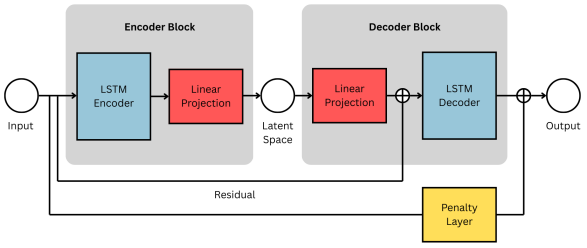

Figure 3: Illustration of LSTM-based autoencoder with penalty layer and residual connection.

2.3 Training Setup

All models were implemented in PyTorch using custom `nn.Module` classes. The vocabulary was limited to standard Tagalog characters and those appearing in the training set, totaling 141 characters. Each one was represented by a 32-dimensional embedding, similar in size to character embeddings commonly used in spaCy (Explosion AI, 2025), which is sufficient given the small vocabulary.

Models were trained using the Adam optimizer with cross-entropy loss. Early stopping with a patience of 10 epochs was applied to prevent overfitting. Experiments were conducted on a machine equipped with an NVIDIA GPU. To ensure reproducibility, random seeds were fixed for PyTorch, NumPy, and Python's `random` module, and deterministic operations were enabled where applicable.

The training hyperparameters for all models are summarized in Table 2.

Table 2: Training hyperparameters for all models.

Hyperparameter	Value
Batch size	32
Learning rate	0.001
Number of epochs	100
Latent dimension	64
LSTM hidden dimension	256
Embedding dimension	32
Early stopping patience	10
Vocabulary size	141

2.4 Evaluation Metrics

To evaluate model performance in Tagalog diacritic restoration, we use three accuracy-based metrics.

Character-Level Accuracy

Character-Level Accuracy (CLA) measures the overall proportion of characters in a sentence that

are predicted correctly:

$$\text{CLA} = \frac{\text{\# correctly predicted characters}}{\text{\# total characters}}.$$

This metric reflects raw reconstruction quality, independent of whether the characters are vowels or consonants. In this study, CLA is primarily used to demonstrate how the penalty mechanism reduces invalid character substitutions, which occasionally appear in unconstrained autoencoder models.

Vowel-Level Accuracy

Vowel-Level Accuracy (VLA) isolates performance on vowel characters only:

$$\text{VLA} = \frac{\text{\# vowels predicted correctly}}{\text{\# total vowels}}.$$

Since Tagalog diacritics (acute, grave, circumflex) appear exclusively on vowels, only vowel positions are relevant to diacritic restoration. For this reason, VLA is treated as the main evaluation metric, as it directly measures the diacritic-restoration component of the task by focusing on characters that can actually change due to diacritization.

Homograph-Level Accuracy

Homograph-Level Accuracy (HLA) measures how often the model correctly restores homographs to their intended diacritized forms:

$$\text{HLA} = \frac{\text{\# correctly restored homographs}}{\text{\# total homographs}}.$$

This metric reflects the overarching linguistic objective of the study: resolving ambiguous Tagalog homographs. While VLA evaluates diacritic accuracy at the character level, HLA measures whether the model succeeds at the higher-level task of restoring meaning-bearing distinctions.

3 Results

3.1 Accuracy & Convergence

Table 3 summarizes the validation performance and convergence of the three model variants. Both modified architectures (Penalty and Residual-Penalty) outperform the Vanilla model across all metrics.

The modified models show clear gains at the character level, achieving almost 8.5× higher CLA than Vanilla, indicating the Penalty mechanism effectively suppresses invalid predictions. For VLA, which reflects restoration of diacritics, both modified variants maintain consistently higher accuracy compared to Vanilla across all epochs (see Figure 4). For the best models, they correctly predict about 64% of vowel diacritics, compared to only 15% for the baseline.

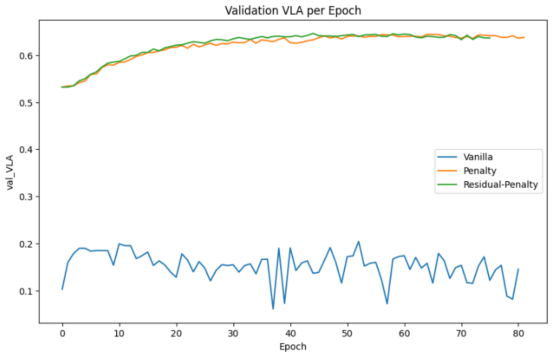

Figure 4: Validation VLA across epochs for all models.

This improvement translates to homograph-level restoration, with over 30% correctly restored by the modified models, while Vanilla fails completely. Across most metrics, the Penalty and Residual-Penalty models perform similarly. However, for HLA, the Penalty model achieves a noticeably higher score, although the difference does not affect overall performance trends.

All models converge steadily, but the modified architectures reach much lower validation loss (Figure 5). Vanilla trains slightly faster per epoch, so it actually reached its stopping point sooner, even though Residual-Penalty converges in fewer epochs overall.

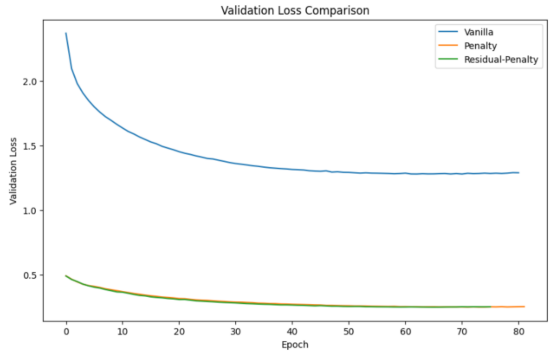

Figure 5: Validation loss curves across epochs for all models.

Finally, on the test set, the trends observed in the validation set are maintained, as shown in Table 4. Both modified models substantially outperform the Vanilla model across all metrics, and their performance is very similar, with the Residual-Penalty model only marginally outperforming Penalty. This consistency demonstrates that the improvements

Table 3: Validation results and convergence statistics.

Model	Val Loss	CLA	VLA	HLA	Convergence	Time/Epoch (s)
Vanilla	1.2794	0.1042	0.1534	0.0000	81 epochs	16.82
Penalty	0.2508	0.8784	0.6396	0.3925	82 epochs	19.39
Residual-Penalty	0.2498	0.8783	0.6394	0.3152	76 epochs	19.33

generalize well to unseen data.

Table 4: Test set performance.

Model	CLA	VLA	HLA
Vanilla	0.1042	0.1518	0.0000
Penalty	0.8770	0.6359	0.3328
Residual-Penalty	0.8791	0.6422	0.3140

Overall, the Penalty and Residual-Penalty models achieve similar performance, both outperforming Vanilla at the cost of a slightly longer per-epoch training time, representing a small trade-off between accuracy and efficiency.

3.2 Latent Space Evaluation

The goal of this analysis is to assess how well the models organize information in their latent spaces. In theory, a well-structured latent space should separate different types of inputs clearly, facilitating downstream tasks such as clustering or classification. To quantify this, we extract the latent representations from the test set for each model and analyze their clustering behavior using the elbow method and silhouette scores.

We use the elbow method on the Vanilla latent space to find the best number of clusters, k. Figure 6 shows the elbow curve, plotting the within-cluster sum of squares (WCSS) and relative improvement $\Delta(K)$ for different K values.

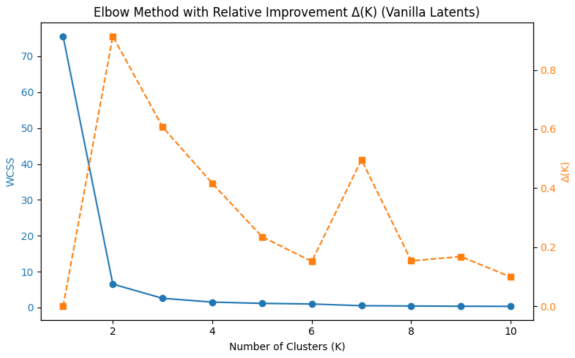

Figure 6: Elbow method with relative improvement $\Delta(K)$ for Vanilla latent space.

The sharp drop at $K = 2$, together with the maximum $\Delta(K)$, indicates that two clusters best represent the latent space. Using k-means with $K = 2$, all three models achieved high silhouette scores, confirming clear cluster separation. Table 5 summarizes the results.

Table 5: Silhouette scores at $K = 2$.

Model	Silhouette Score
Vanilla	0.953
Penalty	0.977
Residual-Penalty	0.999

However, while all models exhibit strong cluster separation, the Penalty model outperforms the Vanilla model, and the Residual-Penalty model achieves the highest score, nearly perfect at 0.999. This suggests that incorporating penalty mechanisms progressively improves the clarity of the latent space clusters.

Now, to further evaluate stability, silhouette scores were computed for $K = 2$ to $K = 10$. Figure 7 compares the three models across this range.

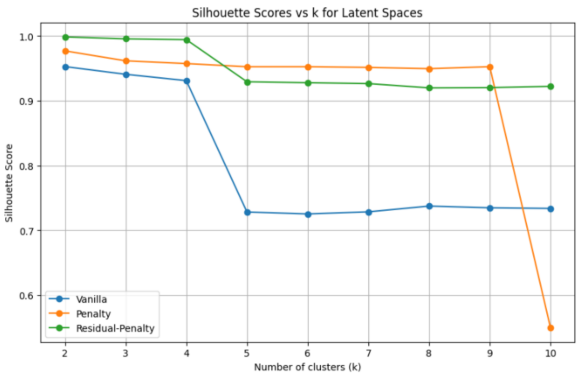

Figure 7: Silhouette scores vs. number of clusters (K) for Vanilla, Penalty, and Residual-Penalty latent spaces.

Analyzing clustering performance across different values of K, the three models exhibit distinct behaviors:

- **Vanilla**: Achieves high silhouette scores for $K = 2$ to $K = 4$, but performance drops sharply by $K = 5$ and remains consistently

lower afterward. This indicates that its latent space becomes less separable as cluster count increases.

- **Penalty**: Delivers consistently strong scores from $K = 2$ to $K = 9$, maintaining tight cluster boundaries across a wide range. Its silhouette score collapses only at $K = 10$, suggesting that the penalty term enforces robust structure but may over-constrain the representation at higher cluster counts.
- **Residual-Penalty**: Achieves the best overall scores, nearly perfect from $K = 2$ to $K = 4$, and remains stable and high up to $K = 10$, with only a modest decline starting at $K = 5$. This indicates that the residual connection helps preserve information even as the number of clusters increases.

Overall, the Residual-Penalty model provides the most flexible and stable latent organization across all cluster sizes, especially for medium to large K. The Penalty model is a strong choice for small to medium K (2–9), while Vanilla performs well only for very coarse clustering ($K = 2$–4) and degrades quickly afterward.

These latent space patterns align with the diacritic restoration results. Specifically, the Residual-Penalty model produces the most stable and well-separated clusters across all K and achieves slightly higher CLA and VLA accuracy on the test set, while the Penalty model performs similarly overall. In contrast, the Vanilla model exhibits poorly separated clusters at higher K, reflecting its rapid decline in reconstruction accuracy and homograph recovery. These findings *suggest* that better-organized latent representations contribute to more reliable and accurate model outputs.

4 Conclusion

This study introduced two architectural modifications—a Penalty layer and a residual input connection—to an LSTM-based autoencoder designed for Tagalog diacritic restoration. The Penalty layer proved to be the main driver of performance gains, substantially improving accuracy by eliminating invalid outputs and enforcing valid character transformations. In contrast, the residual connection contributed only modest improvements in supervised reconstruction accuracy but played a key role in stabilizing and structuring the latent space. Combined, these modifications yielded models that generalize more effectively, produce more coherent latent representations, and consistently outperform the Vanilla LSTM-based autoencoder across both reconstruction and representation-quality evaluations.

These results emphasize the importance of constrained output spaces and stable latent representations in tasks that require precise character-level restoration. Future work may investigate more adaptive penalty mechanisms, explore full sequence-to-sequence architectures, or develop hybrid supervised–unsupervised strategies that leverage the improved latent clustering observed in this study. Extending these methods to full-sentence diacritic restoration or broader homograph disambiguation in Tagalog presents a promising direction. Overall, this work provides a solid foundation for building more robust and linguistically informed models for Tagalog diacritic restoration.

References

Zachary Aldridge L. Africa, Ivan Yuri P. De Leon, and Lanz Railey A. Fermin. 2025. Advancing tagalog diacritic restoration: Corpus expansion, linguistic analysis, and transformer-based methods. Capstone, Ateneo de Manila University.

Trung Duc Anh Dang and Thi Thu Trang Nguyen. 2020. Tdp–a hybrid diacritic restoration with transformer decoder. In *Proceedings of the 34th Pacific Asia Conference on Language, Information and Computation*, pages 76–83.

Explosion AI. 2025. Model architectures: Predefined model architectures included with the core library. https://spacy.io/api/architectures. Accessed: 2025-12-02.

Richell Isaiah Flores. 2024. Diacritics restoration for homograph disambiguation in tagalog: Corpus generation and experiments. Capstone, Ateneo de Manila University.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.