Comparative Analysis of LSTM and Bi-LSTM for Classifying Indonesian New Translation Bible Texts Using Word2Vec Embedding

Marthen Sattu Sambo[1], Suryasatriya Trihandaru[2], Didit Budi Nugroho[3], and Hanna Arini Parhusip[4]*
[1−4]Data Science Department, Faculty of Science & Mathematics, Satya Wacana Christian University
Salatiga, Indonesia 50711
Email: [1]632022004@student.uksw.edu, [2]suryasatriya@uksw.edu, [3]didit.budinugroho@uksw.edu, [4]hanna.parhusip@uksw.edu

***Abstract*—The research aims to compare the classification accuracy of Long Short-Term Memory (LSTM) and Bidirectional LSTM (Bi-LSTM) architectures in classifying Indonesian New Translation Bible texts with Word2Vec embedding. The main objective was to examine how these deep learning models addressed complex and context-rich Indonesian biblical text classification, particularly between Old Testament and New Testament. The dataset used during the analysis contained 31,102 labeled verses with Word2Vec embedding calculated through both Continuous Bag of Words (CBOW) and Skip-Gram methods. Furthermore, the models were evaluated based on accuracy, precision, recall, and F1-score metrics. The results show Bi-LSTM performing better than LSTM in all cases, with the best performance being 92.31% when using Skip-Gram embedding at a vector size of 300 versus 91.94% by LSTM. The model demonstrates its ability to resolve semantic ambiguities in context-rich texts, such as the Bible. The research contributes to the text classification discipline through the provision of empirical evidence of the advantages of Bi-LSTM in managing biblical text processing as well as finding the optimal model architecture mix with word embedding methods. Additionally, the analysis also shows that future studies have to explore other embeddings, including GloVe (Global Vector) and FastText, or use transformer-based models such as Bidirectional Encoder Representations from Transformers (BERT) to improve the classification accuracy.**

***Index Terms*—Text Classification, Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (Bi-LSTM), Indonesian New Translation Bible, Word2Vec Embedding**

Received: Aug. 10, 2024; received in revised form: March 03, 2025; accepted: March 03, 2025; available online: Sep. 15, 2025.
*Corresponding Author

I. Introduction

TEXT classification, an essential task in Natural Language Processing (NLP), is widely used in applications such as sentiment analysis and document organization. This task has been significantly transformed by Deep Learning (DL) models, particularly those based on Neural Networks (NN), which are capable of automatically extracting complex patterns and semantic relationships from text. In line with the discussion, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Bidirectional LSTM (Bi-LSTM) have excellent performance in handling sequential data, including text, due to their ability to capture long-term dependencies as well as contextual relationships.

Text classification in low-resource languages, such as Indonesian, remains a challenging task despite the advancements. Indonesian language, with its rich morphology and contextual nuances, poses unique difficulties for NLP tasks. The lack of annotated datasets in Indonesian further complicates the development of effective text classification models. The Indonesian New Translation Bible, a holy book for Christians and Catholics with a structured format and rich contextual dependencies, serves as an ideal dataset for exploring the challenges described in the research [1]. The Bible consists of two main sections: the Old Testament and the New Testament. Classifying these texts requires DL models capable of understanding both the linguistic complexity of Indonesia as well as the complicated relationships between words and phrases [2]. Moreover, the number of layers in a DL architecture can vary from shallow tissue with only a few layers to deep tissue, ranging from tens to even hundreds of neuron layers that are interconnected and process input data, propagating it through weighted connections [3]. NN architectures, such as RNN, LSTM [4], and Transformer, allow DL models to capture patterns and meanings in text in a more effective way compared to traditional methods.

Studies on Indonesian religious texts (such as the Indonesian New Translation) remain limited despite advancements in DL and word embedding for text classification. A comparison of LSTM and Bi-LSTM performance in low-resource language settings is lacking, as is the consistent use of the Confusion Matrix (CM) for evaluating model performance. As a result, the research addresses the gaps by comparatively analyzing LSTM and Bi-LSTM models for classifying Indonesian New Translation Bible texts using Word2Vec embedding, as well as providing evaluation using CM. By comparing the two models under identical experimental conditions, the researchers aim to objectively confirm the widely held belief that Bi-LSTM outperforms LSTM because the model is bidirectional.

The research aims to address the challenges by comparing the performance of LSTM and Bi-LSTM models for classifying Indonesian New Translation Bible texts using Word2Vec embedding. It is a widely used word embedding method to capture semantic relationships between words by mapping dense vector representations. Two variants of Word2Vec, namely Continuous Bag of Words (CBOW) and Skip-Gram, are used to generate word embedding, enabling the models to better understand the contextual and morphological nuances of the Indonesian language. By leveraging these embeddings, the researchers use LSTM and Bi-LSTM models to classify Bible text into the Old and New Testaments, respectively. The questions addressed are as follows: How do LSTM and Bi-LSTM models perform in classifying Indonesian New Translation Bible texts using Word2Vec embedding? Which model is more effective in handling the linguistic complexity of the Indonesian language?

There are several research contributions. First, the research provides a comparative analysis of LSTM and Bi-LSTM models for classifying Indonesian New Translation Bible texts, a task that has not been extensively studied. Second, it shows the effectiveness of Word2Vec embedding, particularly the Skip-Gram method, in capturing semantic relationships in the Indonesian language. Third, the analysis offers perceptions into the performance of Bi-LSTM over LSTM in handling long-term dependencies and complex linguistic structures, contributing to the development of more effective text classification models for low-resource languages.

A. *Indonesian New Translation Bible*

The Bible, originally written in Hebrew and Aramaic, has been translated into many languages over the centuries. The number of books included in the Bible is established through the process of canonization. The Old Testament was canonized around the 2nd century BC, while the New Testament was formalized in the late 4$^{\text{th}}$ century AD [5].

Indonesian New Translation Bible is a linguistically rich text that poses unique challenges for NLP tasks. The Indonesian New Translation Bible is widely used by Indonesian-speaking Christians and Catholics, translated into modern Indonesian by the Indonesian Bible Institute (Lembaga Alkitab Indonesia (LAI) [6]. Its complex sentence structures, rich contextual dependencies, and morphological variations in the Indonesian language make the book an ideal dataset for exploring advanced text classification methods [7]. In previous research on Natural Language Generation (NLG), the interpreted Bible text from English has been used as a dataset, and there has been no further examination of the translation of Indonesian Bible texts [8].

Indonesian New Translation Bible is divided into two main sections, namely Old and New Testaments. This structure provides a valuable foundation for text classification tasks. Despite the potential of the book as a dataset for NLP studies, the Indonesian New Translation Bible has not been extensively studied in this context compared to the English Bible [1, 9]. As a result, the researchers use the Indonesian New Translation Bible with 66 books, consisting of 37 books for Old Testament and 29 books for New Testament [10]. The linguistic complexity of the Indonesian language, with the rich contextual dependencies in the Indonesian New Translation Bible, presents unique challenges for text classification. These challenges include handling morphological variations, understanding long-term dependencies, and capturing semantic relationships in a low-resource language context. By delivering the Indonesian New Translation Bible as a dataset, the researchers aim to address the lack of analyses on Indonesian religious text classification and contribute to the development of more effective NLP models for low-resource languages.

B. *Word2Vec*

Word embedding is a fundamental component of modern NLP, enabling machines to represent words as dense vectors in a continuous vector space. Among the various methods for generating word embedding, Word2Vec is popular due to its efficiency and effectiveness in capturing semantic relationships between words [11–13]. Word2Vec uses two main architectures, namely CBOW and Skip-Gram. Both leverage a shallow NN to generate word embedding, where words with similar meanings occupy proximate regions in the vector space. Therefore, the text representation method based on the vector space model is simple and more effective in the text preprocessing stage [14, 15].

In the context of the research, Word2Vec is particularly valuable for processing Indonesian New Translation Bible, a text characterized by rich contextual dependencies and morphological complexity. Then, the Skip-Gram architecture excels at capturing rare or nuanced semantic relationships, making it well-suited for analyzing the diverse vocabulary and archaic terminology of the Bible. Consequently, the efficiency of CBOW with a larger dataset follows the structured nature of the textual divisions in the Bible.

Word morphology information is important, as demonstrated in previous studies, particularly in morphologically rich languages such as Indonesian [16]. By generating word embedding using Skip-Gram and CBOW architectures, the model can capture the semantic and syntactic relationships between words. This process enables more accurate classification of Indonesian New Translation Bible texts into respective testaments (Old and New Testaments). Although Word2Vec is extensively applied to high-resource languages such as English, its use for Indonesian text classification, particularly for religious text, remains underexplored. The research addresses the gap by evaluating the effectiveness of CBOW and Skip-Gram embedding in a low-resources language context, contributing to the development of robust NLP tools for Indonesian.

C. Long Short-Term Memory (LSTM)

LSTM is a specialized RNN architecture designed to address the vanishing gradients problem inherent in traditional RNN [4, 17]. The architecture incorporates memory cells and a gating mechanism (input, forget gates, and output) that regulates the flow of information, enabling the model to retain long-term dependencies in sequential data. This capacity makes LSTM particularly effective for tasks requiring contextual understanding of text, such as machine translation, speech recognition, and text classification [18].

In text classification, LSTM excels at capturing patterns and semantic relationships between words, outperforming traditional methods such as bag-of-words or Term Frequency-Inverse Document Frequency (TF-IDF). For instance, previous research has shown that LSTM achieves state-of-the-art results in sentiment analysis by modeling contextual dependencies in sentence structures [19]. However, the model processed sequences unidirectionally (from left to right), limiting its ability to incorporate future contextual information. It is a critical factor in tasks requiring a holistic understanding of text. The research explores the effectiveness of LSTM in classifying Indonesian Bible texts by leveraging the ability to model long-term dependencies, addressing both linguistic and domain-specific challenges.

D. Bidirectional Long Short-Term Memory (Bi-LSTM)

An improved version of the standard LSTM, known as Bi-LSTM, processes text in both directions to understand context from both preceding and following words, overcoming the limitations of one-way LSTM. It has two layers, one forward and one backward, forming a more contextually aware model useful for tasks requiring comprehensive text understanding [20]. Following the discussion, Bi-LSTM outperforms unidirectional LSTM in text classification because it considers context from both directions. Previous studies have shown its effectiveness in sentiment analysis as well as document classification [21, 22]. This bidirectional method is particularly helpful for tasks such as resolving ambiguity and understanding references, which are common in complex texts, including the Bible.

The application of the model to low-resource languages, such as Indonesian, has not been extensively studied, despite Bi-LSTM being proven effective for high-resource languages [8]. This is specifically true for religious texts, where the complex theological vocabulary and the need for understanding long-range contextual relationships show the importance of strong bidirectional modeling. In line with the discussion, classifying passages as Old or New Testaments necessitates an awareness of subtle thematic change, such as the difference between covenant language in Old Testament and grace-focused narratives in New Testament. Bi-LSTM is more capable of capturing these nuances compared to unidirectional models [23]. The research addresses the lack of analysis on bidirectional architectures for low-resource languages by evaluating the performance of Bi-LSTIM on Indonesian Bible text classification, while contributing to the development of NLP tools modified to religious and culturally significant texts.

II. Research Method

The dataset used is Indonesian New Translation Bible, obtained from alkitab.mobi/tb on July 14th, 2024. The data are extracted by copying the text of each verse from the website and pasting it into a Microsoft Excel 2021 spreadsheet (.xlsx). Furthermore, the initial raw data consist of 31,102 rows with five

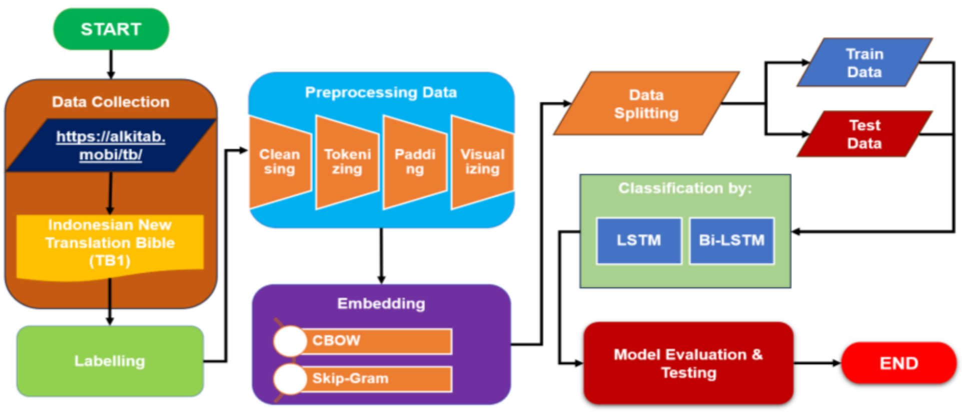


Fig. 1. Study method flowchart. Note: Long Short-Term Memory (LSTM), Bidirectional LSTM (Bi-LSTM), and Continuous Bag of Words (CBOW).

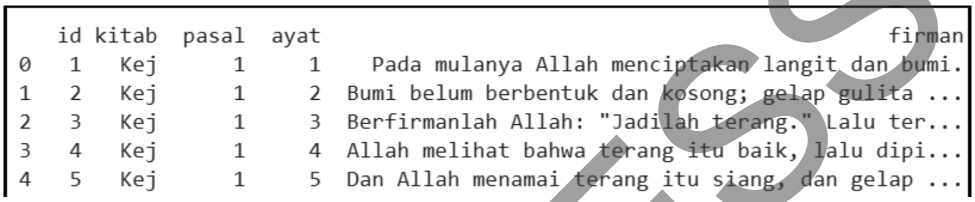

```
   id kitab  pasal  ayat                                            firman
0   1   Kej      1     1     Pada mulanya Allah menciptakan langit dan bumi.
1   2   Kej      1     2   Bumi belum berbentuk dan kosong; gelap gulita ...
2   3   Kej      1     3   Berfirmanlah Allah: "Jadilah terang." Lalu ter...
3   4   Kej      1     4   Allah melihat bahwa terang itu baik, lalu dipi...
4   5   Kej      1     5   Dan Allah menamai terang itu siang, dan gelap ...
```

Fig. 2. Early snippets dataset after being uploaded to Google Collaboratory. It has columns for id, book (*kitab*), chapter (*pasal*), verse (*ayat*), and word (*firman*), where each row represents one Bible verse. For instance, the exampe shows Genesis 1:1. The first row is In the beginning God created the heavens and the earth. The second row is Now the earth was formless and empty. The third row is And God said, "Let there be light" and. The fourth row is God saw that the light was good and. The fifth row is God called the light "day," and the darkness.

features, namely id (a unique identifier for each verse), book (*kitab*), chapter (*pasal*), verse (*ayat*), word (*firman*), and verse text. The research method is illustrated in Fig. 1.

A. Dataset Preprocessing

Figure 2 shows a snippet of the raw data after being uploaded to Google Collaboratory, signifying the initial structure of the dataset. The chapter (pasal) column contained values such as 'Genesis' and 'Matthew', while the word (*firman*) column contained the actual text of each verse. The raw textual data, particularly the entries in the "*firman*" column, are cleaned prior to its use in subsequent machine learning model development.

The Indonesian New Translation Bible comprises 66 books, 1,189 chapters, and 31,102 verses. It is divided into two primary categories: Old Testament with 39 books and New Testament with 27 books. The distribution of the number of chapters and verses in the Indonesian New Translation Bible is illustrated in Fig. 3. This figure presents two histograms that compare the frequency distributions of two variables. The histogram on the left shows the frequency distribution of the number of chapters. The x-axis represents the number of chapters, while the y-axis signifies the frequency or the number of occurrences of chapters with a specific count. According to this histogram, most chapters have a lower count, and the frequency decreases significantly as the chapter count increases. Meanwhile, the histogram on the right shows the frequency distribution of the number of verses. The x-axis represents the number of verses, and the y-axis signifies the frequency or the number of occurrences of verses with a specific count. Similar to the histogram of the chapters, this distribution shows that a lower verse count is more common, and the frequency also decreases as the verse count increases. The histogram shows that most books have a relatively small number of chapters and verses. It implies that the dataset is dominated by shorter books and chapters, influencing the performance of subsequent machine learning models.

In the data preprocessing process, the texts are labeled according to the canonical division: Old Testament (*Perjanjian Lama*, PL) consisting of 39 books (Genesis to Malachi), and New Testament (*Perjanjian Baru*, PB) having 27 books (Matthew to Revelation). Each verse (*ayat*) is extracted and assigned a testament mark (PL or PB), leading to a labeled corpus of 31,102

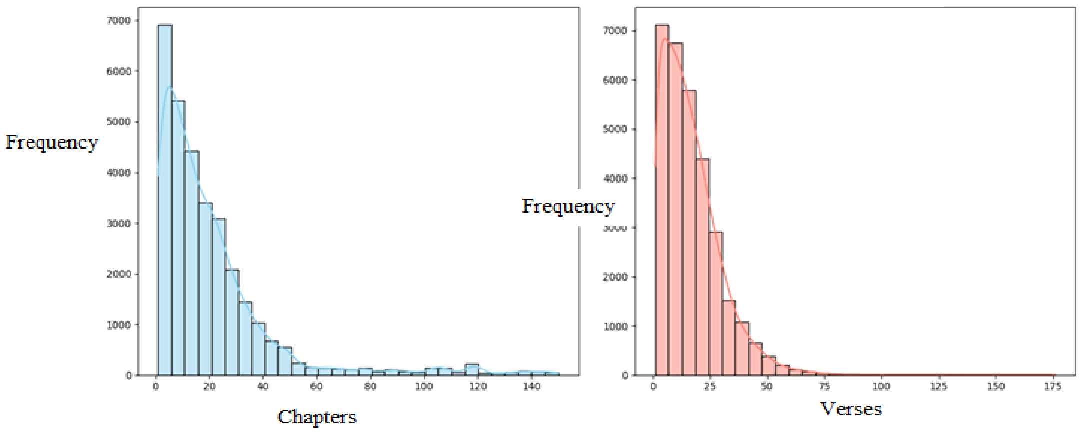


Fig. 3. The distribution of the number of chapters and verses in the Indonesian New Translation Bible.

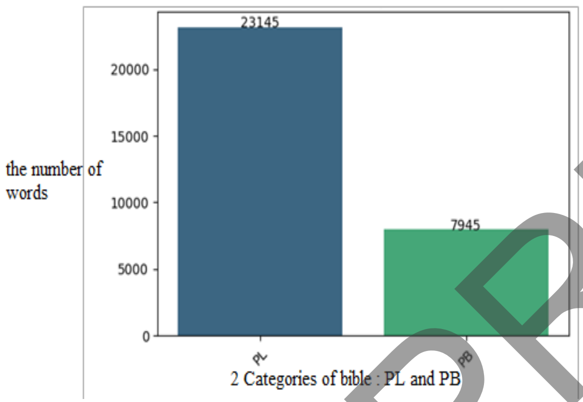


Fig. 4. Distribution of data per category in Indonesian New Translation Bible. It has Old Testament (*Perjanjian Lama*, PL) and New Testament (*Perjanjian Baru*, PB).

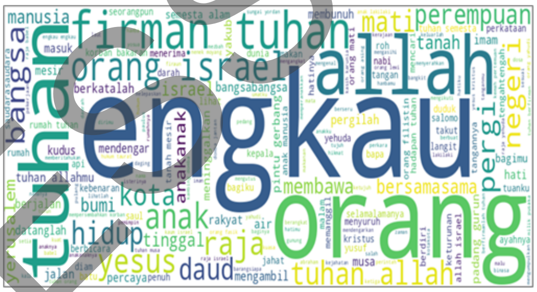


Fig. 5. Word Cloud of Indonesian New Translation Bible.

verses. In addition, the class distribution is shown in Fig. 4, with 74.4% of PL and 25.6% of PB.

The following steps are applied to the word (*firman*) column containing the verse text to prepare the text for model training. Initially, punctuation removal is performed, where special characters and diacritics are stripped to reduce noise. Following that process, lowercasing is applied, where all texts are converted to lowercase to ensure uniformity. Verses are then tokenized into individual words using the nltk.word_tokenize function from the Natural Language Toolkit (NLTK) library. Stopwords are removed using a custom Indonesian stopwords list following the previous step. Finally, text normalization is performed, where rare words (frequency < 2) are filtered to reduce dimensionality. These steps are applied to reduce text variation and ensure consistency for subsequent analysis. Stemming and lemmatization are not performed because the focus is on preserving the original word forms. The processed dataset is partitioned into training and testing sets, stratified by testament labels to preserve class balance.

Next, data visualization helps to ensure the cleaned text data is in accordance with expectations for the following process. A word cloud represents words in a text, where the size of each word reflects its frequency [24]. In the research, Word Cloud features the words that appear most frequently in the text of the Bible. This process effectively shows the major themes and topics in the text through a quick and understandable visual representation.

Figure 5 shows a Word Cloud, where the size of each word reflects its frequency or importance in the text. The different colors used for the words are primarily for visual appeal and do not signify any specific categorization. Moreover, prominent words such as “*engkau*” (you), “*orang*” (people), “*Allah*” (God), “*Tuhan*” (Lord), and “*tuhan*” (god) are larger, signifying the most commonly used words. Other significant words, including “Israel”, “*firman*” (word), “*raja*” (king), “*hidup*” (life), “*Yesus*” (Jesus), and “*perempuan*” (woman), are smaller, reflecting relatively lower

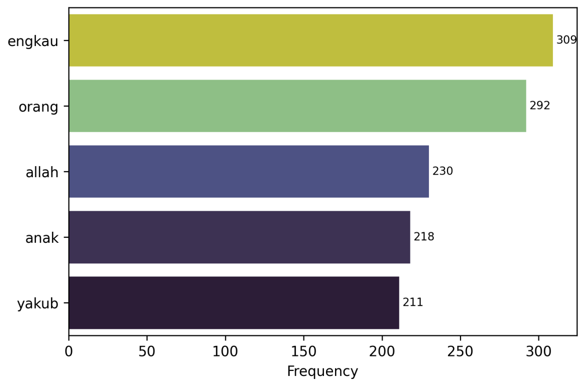


Fig. 6. The five most frequent words in the Genesis of the Indonesian New Translation Bible (in Indonesian). It has "*engkau*" (you), "*orang*" (people), "*Allah*" (God), "*anak*" (child), and Yakub.

frequency.

Figure 6 shows the most frequently appearing words in the book of Genesis. The colors of the bars vary possibly for visual differentiation and are connected to the frequency, with each color representing a specific frequency range as shown in the legend. During the analysis, the word "*engkau*" (you) is the most common, signifying its significant usage in the text of Genesis. The bar chart efficiently shows the most common words in this biblical book, with clear numerical labels ending each bar. It is useful for understanding the vocabulary emphasis in Genesis, particularly in Indonesian translation. Following the discussion, the analysis also observes other books which are not presented here.

The dataset under examination comprises text tokens extracted from the Indonesian New Translation Bible. A heatmap visualization is used to identify and analyze the most frequently occurring unique words in the text. This method effectively signifies the frequency distribution of the words, as shown in Fig. 7. The heatmap shows the frequency of specific words across various books of the Indonesian New Translation Bible, such as such as "*allah*" (god), "*anak*" (child), "*engkau*" (you), "*firman*" (word), Israel, "*orang*" (people), "*perempuan*" (women), "*raja*" (king), "*tuhan*" (lord), and "*yesus*" (jesus), tracked along the vertical axis. In contrast, the books of the Bible are listed along the horizontal axis. The color intensity represents the frequency of each word in a specific book, with lighter colors signifying fewer occurrences and darker shades, particularly blue, representing higher frequencies. For instance, "*allah*" (god) appears frequently in books such as Psalms and Isaiah. Then, "*orang*" (people) is prevalent in Exodus and Jeremiah, and "*tuhan*" (god) is frequent in Jeremiah and Isaiah. This visualization enables a quick and comprehensive comparison of word usage across different parts of the Bible, highlighting patterns and thematic stress in the text.

After the sequences have been padded to a uniform length, the dataset is partitioned into two distinct subsets: the training set and the testing set. This division follows an 80:20 split, allocating 80% of the data for training the model and the remaining 20% for evaluating the performance, as the rationale behind the ratio is to strike a balance. During the analysis, the larger training set (80%) provides the model with a substantial amount of data to learn from, enabling it to discern fundamental patterns and relationships in the data. Simultaneously, the testing (20%) serves as an independent benchmark to assess the ability of the model to generalize unseen data, ensuring that the evaluation was conducted on a representative sample, copying the characteristics of the total dataset [25].

Next, the process is creating, training, and evaluating the models. Two models of DL algorithms, LSTM and Bi-LSTM, are built for the binary text classification experiment during the analysis using word embedding. Figure 8 shows the architecture of vanilla LSTM and Bi-LSTM models used.

The function of lstm_model forms and compiles an LSTM-based NN using Keras. The model is initialized as a sequential model named "marss_model", allowing layers to be stacked linearly. The process starts with a masking layer to handle sequences of varying lengths, followed by LSTM layer with 128 units and L2 regularization to prevent overfitting. Batch normalization is then applied to normalize the output, and dropout is used to randomly set 20% of input units to zero during training to prevent overfitting further. The model continues with a dense layer containing 16 units with Rectified Linear Unit (ReLU) activation and additional L2 regularization, followed by another round of batch normalization and dropout. Finally, a dense layer with 2 units and a softmax activation function is added for multi-class classification. The model is compiled with categorical cross-entropy as the loss function, Adam optimizer with a learning rate of 0.0001, and metrics including accuracy, precision, and recall. During the process, the model is set to run eagerly, which aids in debugging, and the function returns this fully compiled model, ready for training as well as evaluation.

During the process, the function calls bilstm_model, which forms and compiles a Bi-LSTM model using Keras. The model is initialized as a sequential model named "marss_model", allowing layers to be added linearly. The process starts with a masking layer to handle sequences of varying lengths by skipping certain time steps, after a Bi-LSTM layer with 128 units processes

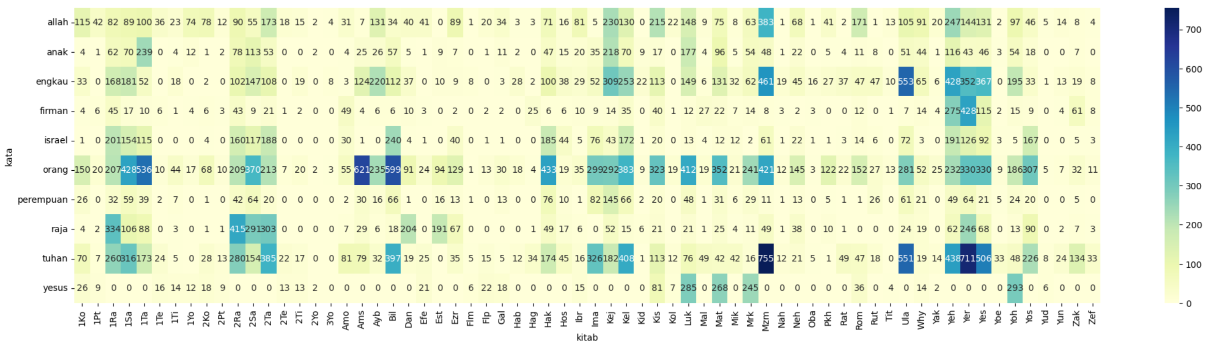
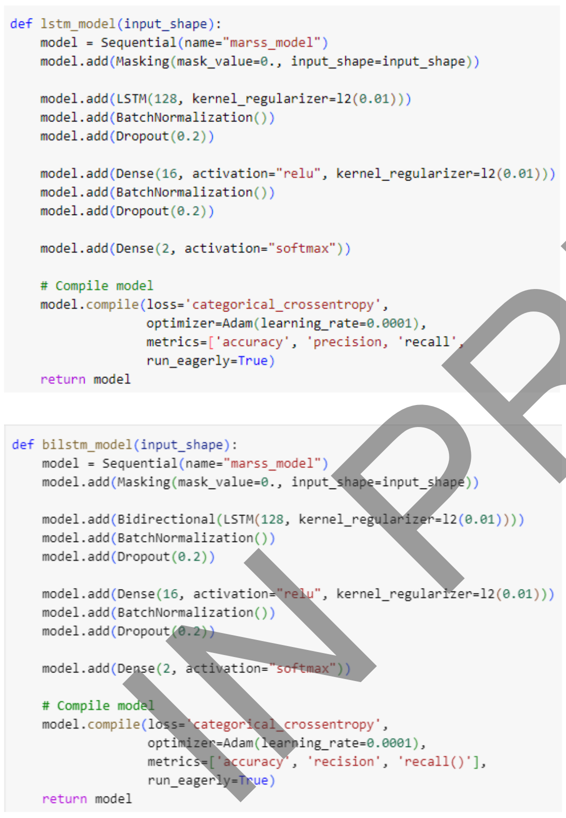


Fig. 7. The ten most frequent words and their frequency in every book in the Bible. It has *allah* (god), *anak* (child), *engkau* (you), *firman* (word), Israel, *orang* (people), *perempuan* (woman), *raja* (king), *tuhan* (god), and *Yesus* (Jesus).



```
def lstm_model(input_shape):
    model = Sequential(name="marss_model")
    model.add(Masking(mask_value=0., input_shape=input_shape))

    model.add(LSTM(128, kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(16, activation="relu", kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(2, activation="softmax"))

    # Compile model
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.0001),
                  metrics=['accuracy', 'precision, 'recall',
                  run_eagerly=True)
    return model
```

```
def bilstm_model(input_shape):
    model = Sequential(name="marss_model")
    model.add(Masking(mask_value=0., input_shape=input_shape))

    model.add(Bidirectional(LSTM(128, kernel_regularizer=l2(0.01))))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(16, activation="relu", kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(2, activation="softmax"))

    # Compile model
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.0001),
                  metrics=['accuracy', 'recision', 'recall()'],
                  run_eagerly=True)
    return model
```

Fig. 8. Long Short-Term Memory (LSTM) and Bidirectional LSTM (Bi-LSTM) models for the classification of two classes.

the input data in both forward and backward directions to capture context from both past and future states. Relating to the process, L2 regularization is applied to the kernel of the LSTM to prevent overfitting. The model then includes batch normalization to stabilize and speed up training, followed by a dropout layer that randomly sets 20% of the input units to zero to prevent overfitting further. A dense layer with 16 units and ReLU activation is added to help the model to learn more complex representations, with additional batch normalization and dropout layers to improve regularization. Finally, a dense output layer with 2 units and a softmax activation function is added for multi-class classification. The model is compiled using categorical cross-entropy as the loss function, the Adam optimizer with a learning rate of 0.0001, and metrics including accuracy, precision, and recall. Moreover, eager execution is enabled for easier debugging, and the function returns a fully compiled Bi-LSTM model, ready for both training and evaluation.

The architecture of the built LSTM model during the analysis consists of a single layer with 128 units, followed by a 16-unit dense layer before moving into the output layer. Similarly, the architecture of the Bi-LSTM model is kept the same for comparison of performance on the same dataset input. The Bi-LSTM architecture is built by replacing an LSTM layer with a Bi-LSTM layer having the same number of units. In these two models, dropout with a rate of 0.2 and L2 regularization with a coefficient of 0.01, as well as batch normalization, are employed to prevent overfitting. The model is trained using the Adam optimizer with a learning rate of 0.0001 and a loss function of categorical cross-entropy. Additionally, the performance metrics used are accuracy, precision, recall, and F1-Score.

The research reviews the performance analysis of the Indonesian New Translation Bible text classification model by comparing evaluation metrics, such as accuracy and F1-Score, between LSTM and Bi-LSTM models using CBOW as well as Skip-Gram embedding on a variety of vector sizes. The objective is to identify the optimal configuration model and discuss the effectiveness of the results on the selection of methods and the parameters. The research provides insight into the efficiency and effectiveness of each

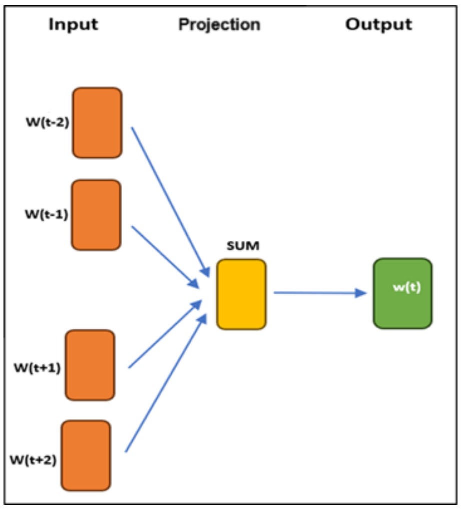


Fig. 9. Architecture of Continuous Bag of Words (CBOW).

model in the context of classifying biblical texts.

B. Word2Vec Embedding Generation

During the analysis, a Word2Vec embedding is generated using the Gensim library, with two architectures evaluated: CBOW and Skip-Gram. CBOW is an algorithm used in NLP to learn distributive representations of words [26]. The main objective of CBOW is to predict target words based on the context of the surrounding words, with a focus on efficiency and good performance in a large corpus. Relating to the process, the model uses a contextual window that includes words before and after the target word in a sentence. In the model, vector representations of words are combined linearly to predict the target word. This algorithm works by optimizing the function of the objective through a machine learning process to obtain a quality representation of the word vector. The method functions well in capturing the general meaning and semantic relationships between words that often appear together. The advantage of CBOW is the ability to overcome infrequent word frequency errors, making it more stable in applications with large volumes of data. Figure 9 shows the main architectural form of CBOW, where the $w(t)$ is the weight for the target word, the input represents the input layer, the projection signifies the hidden layer, and the output is the output layer.

Skip-Gram is an algorithm in NLP to study distributive representations of words to predict context words based on a given target word [27]. Concerning Skip-Gram, the context around the target word is given in the form of a window of a certain size. The target word representation vector is then generated by maximizing the probability of predicting the words of that context.

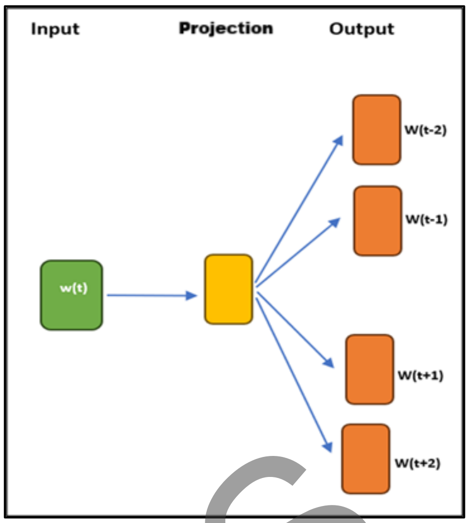


Fig. 10. Architecture of Skip-Gram.

This algorithm aims to generate a vector of word representations that describe the semantic and syntactic relationships between words in a corpus. Skip-Gram works by predicting contextual words based on the target word. It is more effective at capturing semantic relationships that rarely appear and are useful in a small corpus. Moreover, the model requires more computing resources than CBOW [28], providing better results in applications where the semantic relationship between important words is fine. Figure 10 shows the target word $w(t)$ is used as input, which is projected into the vector space. Then, the result of this projection is used to predict the surrounding context words.

In the research, CBOW and Skip-Gram embeddings are constructed using a corpus of 31,102 sentences and 303,139 words in the dataset. The main parameters include vector sizes (vector_size) of 50, 100, and 300. It is a 5-word window, and words with a minimum frequency of occurrence (min_count) of 2. During the analysis, both embedding models from Word2Vec are trained using the Indonesian New Translation Bible dataset to generate meaningful word vectors (wv). The vector dimension is set to three types or variations. The objective is to get the optimal representation of each word in the dataset. The model is trained with 100 iterations or epochs for both CBOW (sg = 0) and Skip-Gram (sg = 1) methods, utilizing four processor cores (workers = 4) to enhance training efficiency.

The resulting word embedding model allows a representation of words in the form of fixed-size vectors (tb_cbow.model or tb_skgram.model), which are used as inputs to LSTM and Bi-LSTM models in the task of classifying biblical texts. Vector representations of

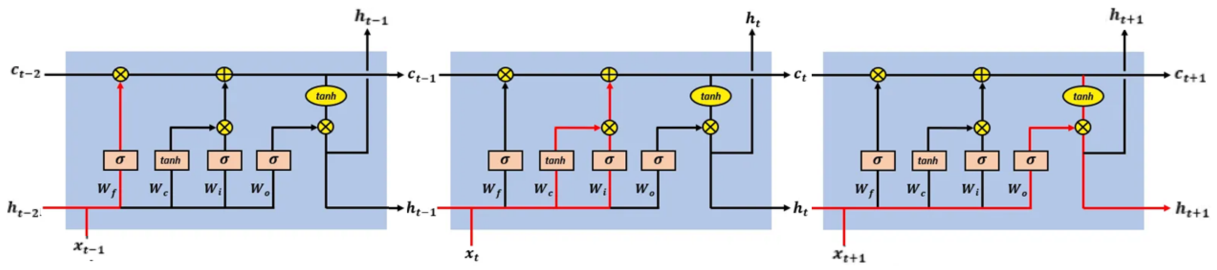


Fig. 11. Network Long Short-Term Memory (LSTM) cells at time steps $t-1$, t, and $t+1$.

words also allow the model to better understand and capture the semantic meaning of words in context. It also improve the performance and accuracy of the model. Then, the vector embedding process is performed for each sentence in the dataset, followed by padding to standardize the sentence length. Additionally, padding is conducted by adding at the end of a sentence (post), ensuring that all vector sequences have the same length (max_len = 37). Verification of the padding result is performed by printing the shape of the first ten sequences, ensuring uniform length.

C. Model Architecture (Long Short-Term Memory (LSTM) Versus Bidirectional LSTM (Bi-LSTM))

Two sequential models are implemented using TensorFlow.Keras. It is selected as the framework because of its flexible structure, user-friendly high-level Application Programming Interface (API), and optimized performance for DL. They support the efficient design and training of sequential NN models.

LSTM is a specialized RNN architecture designed to address the vanishing gradients problem inherent in traditional RNN [17]. The LSTM architecture model has the advantage of doing a combination of iterations in data processing to improve the accuracy of the model. Cell state (C_t) is the main component of LSTM that transmits information through a time sequence. It is often referred to as the model's long-term memory. Following the discussion, the LSTM network has three gates that update and control the cell state, namely forget, input, and output, using the hyperbolic tangent activation function. Moreover, the gate that fails to control information should be forgotten within the cell, allowing new data to enter the network [4].

Figure 11 shows the unrolled architecture of LSTM network. It illustrates the flow of information across time steps. Each cell contains three gates: forget, input, and output that regulate the cell state (C_t) and hidden state (h_t), enabling the model to capture long-term dependencies in sequential data.

Figure 12 shows internal structure of an LSTM cell. The architecture consists of a forget gate (W_f), input gate (W_i), candidate cell state (W_c), and output gate (W_o). They jointly regulate the update of the cell state (C_t) and hidden state (h_t). The gates use sigmoid and tanh activations to control information flow, enabling the model to retain or discard information across time steps.

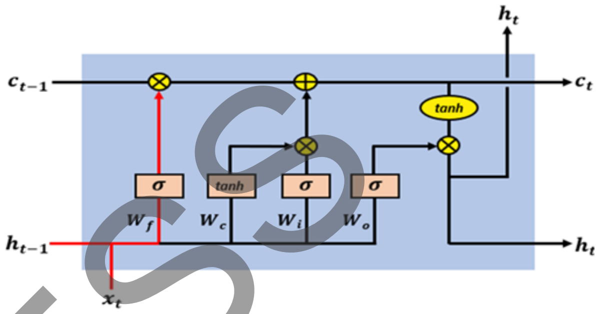


Fig. 12. Forget gate on Long Short-Term Memory (LSTM) architecture.

At the forget gate, irrelevant information is forgotten or not forwarded to the next gate. The area decides which information needs to be deleted from memory based on the current context. Then, on the mentioned gate, the two equalities occur during the analysis [29]. Equation (1) is the first activation function that determines how much information from the current input, and the previous state is retained or forgotten. Meanwhile, Eq. (2) represents the multiplication of the previous cell stated by the output of the forget gate to update the current cell state by retaining the relevant information [4, 19]. The equations show f_t as forget gate vector (controls retained/discarded info), σ as sigmoid activation function [0, 1], W_{fx} as forget gate weight applied to input, x_t as input vector at time step t, U_{fh} as weight matrix for the previous hidden state, h_{t-1} as previous hidden state, b_f as forget gate bias, and C_{t-1} as previous cell state.

$$f_t = \sigma(W_{fx}x_t + U_{fh}h_{t-1} + b_f), \tag{1}$$

$$C_{t-1/2} = C_{(t-1)} * f_t. \tag{2}$$

The function of the input gate in Fig. 13 is to

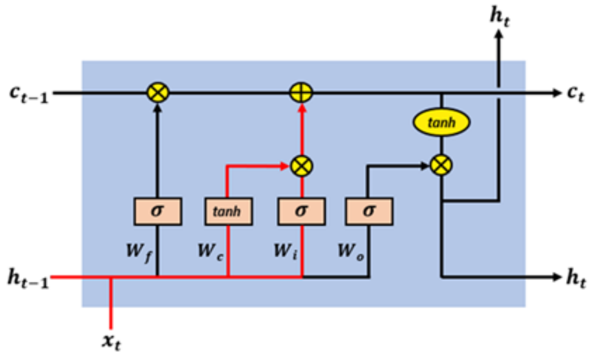


Fig. 13. Input gate on Long Short-Term Memory (LSTM) architecture.

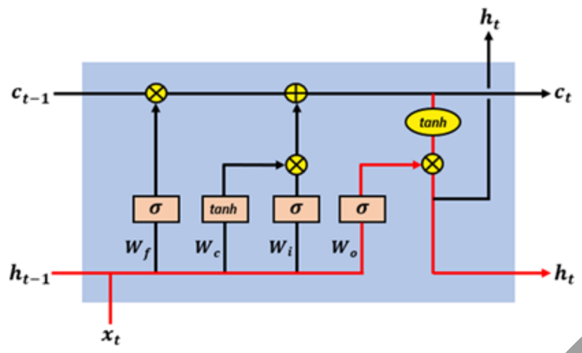


Fig. 14. Output gate on Long Short-Term Memory (LSTM) architecture.

determine what new information should be stored in memory, ensuring that only important or relevant data are added. Equation (3) calculates the value of the new cell state using the tanh activation function, while Eq. (4) specifies how much new information will be added to the cell state through the input gate. To update the current cell state, Eq. (5) combines the information retained from the previous cell state with the new relevant information [4]. They show C_t as updated cell state (long-term memory), i_t as input gate activation, $tanh$ as hyperbolic tangent activation function, W_{cx} and W_{ix} as weight matrices for the input vector, U_{ch} and U_{ih} as weight matrix for the previous hidden state, and b_c as input gate bias.

$$g_t = tanh(W_{cx}x_t + U_{ch}h_{t-1} + b_c), \tag{3}$$
$$i_t = \sigma(W_{ix}x_t + U_{ih}h_{t-1} + b_i), \tag{4}$$
$$C_t = C_{t-1} * f_t + i_t * g_t. \tag{5}$$

The output gate, shown in Fig. 14, takes the current input x_t and the previous hidden state h_t. It processes them through a sigmoid activation to generate a gating vector. Then, it multiplies this with the $tanh$ of the updated cell state C_t.

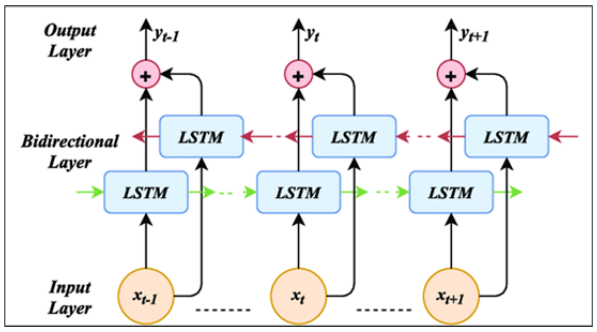


Fig. 15. Architecture of Bidirectional Long Short-Term Memory (Bi-LSTM) to show direction forward (green arrow) and backward (red arrow) [30].

The output gate functions to control which part of the information is stored in memory and used to produce the output at each time step. During the analysis, Eq. (6) determines how much information from the cell state will be passed to the output through the activation function. To generate the current hidden state, Eq. (7) combines the sigmoid cell state that has been activated through $tanh$ with the output gate. Equation (8) is used to calculate the final output of the LSTM network based on the updated hidden state. The combination of these three gates allows LSTM to effectively capture and retain critical information in sequential data for both the long and short term. They show o_t as output gate activation (controls what information to output), h_t as hidden state at current time step, W_{ox} and W_{oy} as weight matrices for output vector, U_{oh} as weight matrix for the previous hidden state, y_t as final output, and b_o, b_{oy} as output gate bias.

$$o_t = \sigma(W_{ox}x_t + U_{oh}h_{t-1} + b_o), \tag{6}$$
$$h_t = tanh(c_t) * o_t, \tag{7}$$
$$y_t = W_{oy}h_t + b_{oy}. \tag{8}$$

Bi-LSTM is a variant of the LSTM architecture. The objective is to improve the ability of the model to process and understand data sequences. Following the process, the Bi-LSTM architecture is shown in Fig. 15.

In the context of Bi-LSTM, the model consists of two separate independent LSTMs that run simultaneously and in parallel. LSTM processes the sequence of data from start to finish (forward), while another part handles it in the same way (backward). In this case, two independent LSTM factors are connected to the same input layer but produce two different sets of outputs. Each model functions as a standard LSTM with the same cell state and gate. The outputs of these two LSTMs are combined, often through a concatenation or sum operation, to produce a final representation that considers information from both directions.

In forward LSTM, for each step, the hidden state forward is calculated by processing the input along with $\overrightarrow{h_t}$ in the previous cell state. Equation (9) shows the forward computation of the LSTM, where the current hidden state $\overrightarrow{h_t}$ is derived from the input at time step x_t and the previous hidden state $\overrightarrow{h_{t-1}}$. This process enables the model to capture sequential dependencies in the forward direction of the input sequence. It has $\overrightarrow{h_t}$ as hidden state from the forward LSTM at time step t, $LSTM_{forward}$ as an LSTM transformation function that processes data sequentially from $t = 1$ to $t = T$ (from the beginning to the end of the sequence), x_t as input vector at time step t, and $\overrightarrow{h_{t-1}}$ as the hidden state of the forward layer from the previous time step $t - 1$.

$$\overrightarrow{h_t} = LSTM_{forward}(x_t, \overrightarrow{h_{t-1}}). \quad (9)$$

In backward LSTM, the model processes data from the start to the end of the sequence, capturing future context. At each step of time t, LSTM backward handles the input along with the next x_t hidden state of h_{t+1} and the memory cell C_{t+1}. Equation (10) represents the backward pass of the LSTM, in which the hidden state at time step t, $\overleftarrow{h_t}$, is computed using the current input x_t and the hidden state from the following step $\overleftarrow{h_{t-1}}$. This mechanism enables the model to learn contextual information from the sequence in a reverse order. It has $\overleftarrow{h_t}$ as hidden state from the backward LSTM at time step t, $LSTM_{backward}$ as the function which processes the input sequence in reverse chronological order from $t = T$ to $t = 1$, x_t as input vector at time step t, and $\overleftarrow{h_{t-1}}$ as the hidden state of the backward layer from the next step $t + 1$.

$$\overleftarrow{h_t} = LSTM_{backward}(x_t, \overleftarrow{h_{t-1}}). \quad (10)$$

The final output of the two LSTMs is combined into one output during the analysis. For example, when the output of the forward LSTM at time t is $\overrightarrow{h_t}$, and the output of the backward LSTM is $\overleftarrow{h_t}$. Then, the final output at time y_t at time t is in Eq. (11). Equation (11) shows how the output y_t is generated by combining the forward hidden state $\overrightarrow{h_t}$ and the backward hidden state $\overleftarrow{h_t}$. These two vectors are concatenated, then linearly transformed using the weight matrix W_y and bias term b_y. This process allows the model to integrate contextual information from both past and future time steps. Equation (11) has y_t as output vector at time step t, b_y as bias term, and W_y as weight matrix for output vector.

$$y_t = W_y[\overrightarrow{h_t}; \overleftarrow{h_t}] + b_y. \quad (11)$$

In contrast to traditional unidirectional LSTMs, the Bi-LSTM framework processes the data in both forward and backward directions. Through processing the sequence of data in two directions, Bi-LSTM understood the context of both directions [31]. This dual-stream processing enables the model to capture complex long-range dependencies that may otherwise be overlooked. Synthesizing information from previous and upcoming tokens results in richer and more nuanced feature representations for each element in the sequence. It is particularly useful for case examples requiring a detailed understanding of sequences, such as language translation, sentiment analysis, and sequence tagging.

TABLE I
EXPERIMENT SETTINGS.

Parameters	Value
input_length	37
embedding_dim	50, 100, 300
lstm_units & bi-lstm_units	128 (vanilla)
dropout_rate	0.2
batch_size	128
epochs	20
activation	ReLU
optimizer	Adam
learning_rate	0.0001
kernel_regularizer	L2=0.01
number_classes	2
classifier	Softmax

D. Training and Evaluation Metrics

The training parameters applied during the analysis are shown in Table I. This process is conducted to ensure robust model convergence while avoiding overfitting. The experimental settings are defined with an input length of 37 with embedding dimensions of 50, 100, and 300. Both LSTM and Bi-LSTM models employ 128 hidden units in a vanilla architecture, while a dropout rate of 0.2 is applied to mitigate overfitting. Training is conducted with a batch size of 128 over 20 epochs. The ReLU activation function is used to introduce non-linearity, and the Adam optimizer with a learning rate of 0.0001 is adopted to ensure efficient parameter updates. An L2 regularization factor of 0.01 is applied to further control overfitting. Finally, the classification task involves two output classes, implemented with a Softmax layer.

CM has been widely used in the evaluation of scientific models and engineering applications in different areas, including NLP [32]. CM is constructed for both LSTM and Bi-LSTM classifiers to provide a granular evaluation of model performance. During the process, CM shows the count of correct and incorrect predictions made by the model for each target class. It offers valuable perceptions into the performance of text

TABLE II
CONFUSION MATRIX IN THE RESEARCH [20].

	Predicted PL	Predicted PB
Actual PL	True Positive	False Negative
Actual PB	False Positive	True Negative

Note: PL: Old Testament and PB: New Testament

classification models in Indonesian New Translation Bible context. Often used in binary classification, CM is distinguished between positive and negative classes. It also adapts multi-class classification scenarios [20, 33]. This matrix (see Table II) quantifies four major outcomes during the analysis as follows:

1) True Positive (TP): Old Testament verses are correctly classified as Old Testament,
2) True Negative (TN): New Testament verses are correctly classified as New Testament,
3) False Positive (FP): New Testament verses are misclassified as Old Testament,
4) False Negative (FN): Old Testament verses are misclassified as New Testament.

By analyzing these values, the researchers evaluate the performance of the model using metrics such as accuracy, precision, recall, and F1-Score. Accuracy measures the total correctness of the predictions of the model, while precision and recall provide perceptions into how well the model identifies and classifies Bible passages. F1-Score combines precision and recall into a single metric, offering a balanced view of the model's ability to handle the nuanced and complex text of the Indonesian New Translation Bible. CM enables a detailed analysis of class-specific issues, such as how the model shows bias toward misclassifying verses as Old Testament (high FP) or vice versa (high FN). For example, in the Bi-LSTM model, a lower FP rate compared to LSTM signifies improved ability to distinguish nuanced theological terminology in New Testament texts. This analysis is critical for understanding how bidirectional context modeling improved classification robustness.

Binary cross-entropy is applied to measure the loss of the model in the analysis. This process is appropriate for the two-class classification task. Model performance is rigorously assessed using four standard metrics as follows:

1) Accuracy measures the total proportion of correctly classified verses.

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN). \quad (12)$$

2) Precision quantifies the ability of the model to avoid false positives (e.g., misclassifying New Testament verses as Old Testament).

$$\text{Precision} = TP/(TP + FP). \quad (13)$$

3) Recall (sensitivity) evaluates the ability of the model to identify all relevant instances (e.g., minimizing missed Old Testament verses).

$$\text{Recall} = TP/(TP + FN). \quad (14)$$

4) The F1-Score provides a balanced measure of precision and recall, which is critical for imbalanced datasets.

$$\text{F1-Score} = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall}). \quad (15)$$

Accuracy (Eq. (12)) measures the overall correctness of predictions by comparing the proportion of correctly classified instances ($TP + TN$) to the total number of cases. Precision (Eq. (13)) quantifies the proportion of correctly predicted positive cases (TP) among all instances predicted as positive ($TP+FP$), while recall (or sensitivity) (Eq. (14)) indicates the ability of the model to identify actual positive cases by calculating the ratio of TP to all actual cases ($TP + FN$). Finally, the F1-Score (Eq. (15)) provides a balanced measure between precision and recall by computing their harmonic mean, offering a more reliable metric in scenarios with imbalanced class distributions.

III. RESULTS AND DISCUSSION

A. Performance Comparison (Long Short-Term Memory (LSTM) Versus Bidirectional LSTM (Bi-LSTM))

The experimental results show that the Bi-LSTM model outperforms the standard LSTM in classifying Indonesian New Translation Bible texts. As shown in Table III, Bi-LSTM achieves an accuracy of 92.31%, compared to 91.94% in LSTM. Similarly, Bi-LSTM shows superior performance in precision (92.45% vs. 91.72%) and recall (92.28% vs. 91.65%). These results signify its improved ability to capture bidirectional contextual relationships in the text. The marginal yet consistent improvement of Bi-LSTM is associated with its theoretical advantage of leveraging both past and future contextual information. It is particularly critical for classifying Bible texts. The thematic shift between Old and New Testaments often depends on nuanced linguistic signs, spanning entire sentences or paragraphs.

B. Confusion Matrix Analysis

The CM in Table IV provides a granular view of model performance. For Bi-LSTM, the majority of misclassifications occur in Old Testament texts (FN

TABLE III
PERFORMANCE COMPARISON OF LONG SHORT-TERM MEMORY (LSTM) AND BIDIRECTIONAL LSTM (BI-LSTM) MODELS.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
LSTM	91.94	91.72	91.65	91.68
Bi-LSTM	92.31	92.45	92.28	92.36

TABLE IV
RESULTS OF CONFUSION MATRIX FOR LONG SHORT-TERM MEMORY (LSTM) VERSUS BIDIRECTIONAL LSTM (BI-LSTM) WITH OLD TESTAMENT (PL) AND NEW TESTAMENT (PB).

	Predicted PL	Predicted PB
	Bi-LSTM	
Actual PL	2,150 (TP)	45 (FN)
Actual PB	38 (FP)	1,867 (TN)
	LSTM	
Actual PL	2,130 (TP)	65 (FN)
Actual PB	52 (FP)	1,853 (TN)

Note: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

TABLE V
COMPARISON OF WORD2VEC ARCHITECTURES.

Architecture	Vector Size	Accuracy (%)
Continuous Bag of Words (CBOW)	300	91.12
Skip-Gram	300	92.31

= 45). Meanwhile, LSTM shows a higher rate of false positives (FP = 52). The results suggest that bidirectional context modeling with Bi-LSTM reduces the misclassification of New Testament texts as Old Testament. It is possibly due to the ability to resolve confusing terms (e.g., “covenant” in Old Testament vs. “grace” in New Testament) through holistic sentence analysis. Moreover, the reduced FN and FP rates in Bi-LSTM show its suitability for tasks requiring fine-grained contextual understanding. These tasks include classifying religious texts with overlapping themes during the analysis.

C. Results of Model Performance

Skip-Gram architecture produces superior results compared to CBOW, with a vector size of 300 achieving the highest accuracy, as shown in Table V. The ability of Skip-Gram to capture rare and nuanced semantic relationships proves critical for processing diverse vocabulary in the Indonesian New Translation Bible, which includes archaic terms (e.g., “*kudus*” (holy), “*perjanjian*” (covenant)) as well as domain-specific theological language. This result is associated with prior work showing the effectiveness of Skip-Gram in low-resource language contexts [11]. By capturing fine-grained semantic relationships, Skip-Gram embedding enables the model to distinguish subtle thematic differences between Old and New Testaments texts, such as the shift from a legalistic language to a grace-centric narrative.

The LSTM model with CBOW, as shown in Fig. A1 in Appendix, exhibits an increase in performance alongside an improvement in vector size. The best accuracy (0.9172) is obtained during the analysis when using a vector size of 300. Meanwhile, the Bi-LSTM model with CBOW also exhibits an increasing trend in performance with larger vector sizes. The best accuracy, with a value of 0.9185, is obtained when using a vector size of 300.

Moreover, the LSTM model with Skip-Gram in Fig. A2 in Appendix shows more varied performance. Its best accuracy (0.9194) is obtained with a vector size of 300. It is higher compared to all CBOW vector sizes. Meanwhile, the Bi-LSTM model with Skip-Gram presents the best performance among all model combinations and embedding methods at a vector size of 300, with an accuracy of 0.9231.

LSTM model achieves its best F1-Score with a vector size of 300 (0.9145) using the CBOW method. Similarly, using Skip-Gram, the LSTM model achieves the best accuracy, which is 0.9187 at a vector size of 300. It is slightly higher than the best performance of LSTM with CBOW. Table VI presents a recapitulation of the experiment using an LSTM model with CBOW and Skip-Gram word embeddings for three variations in vector dimensional sizes.

The Skip-Gram method with a vector size of 300 achieves the highest F1-Score (0.9187). It shows the best balance between precision and recall. Although CBOW also produces good results on all vector sizes, Skip-Gram is superior at vector sizes of 300. In general, increasing the vector size from 50 to 300 would improve the performance of the LSTM model, especially for the Skip-Gram method.

The Bi-LSTM model with the embedding of the CBOW method achieves the best F1-Score, with a vector size of 300, at 0.9165. For the use of the Skip-Gram method, the Bi-LSTM model achieves the best total F1-Score with a vector size of 300 (0.9218), outperforming the performance of LSTM. Additionally, the results of the experiment using a Bi-LSTM model with CBOW and Skip-Gram word embedding for three variations in

TABLE VI
EVALUATION RESULTS OF LONG SHORT-TERM MEMORY (LSTM) MODEL.

Embedding	Vector Size	Accuracy	Precision	Recall	F1-Score
Continuous Bag of Words (CBOW)	50	0.9133	0.9118	0.9133	0.9115
	100	0.9151	0.9136	0.9151	0.9132
	300	0.9172	0.9166	0.9172	0.9145
Skip-Gram	50	0.9138	0.9138	0.9138	0.9111
	100	0.9098	0.9098	0.9098	0.9059
	300	0.9194	0.9184	0.9194	0.9187

TABLE VII
EVALUATION RESULTS OF BIDIRECTIONAL LONG SHORT-TERM MEMORY (BI-LSTM) MODEL.

Embedding	Vector Size	Accuracy	Precision	Recall	F1-Score
Continuous Bag of Words (CBOW)	50	0.9127	0.9119	0.9127	0.9097
	100	0.9164	0.9159	0.9164	0.9135
	300	0.9185	0.9173	0.9185	0.9165
Skip-Gram	50	0.9112	0.9103	0.9112	0.9082
	100	0.9133	0.9128	0.9133	0.9102
	300	0.9231	0.9219	0.9231	0.9218

vector size are shown in Table VII.

The Skip-Gram model with a vector size of 300 yields the highest F1-Score (0.9218). It shows the best balance between precision and recall for the Bi-LSTM model. Despite CBOW producing consistent results with an increase in vector size, it cannot perform with Skip-Gram at a vector size of 300. In LSTM models, increasing the vector size from 50 to 300 tends to improve the performance of Bi-LSTM models in general, especially in the Skip-Gram method. The best option based on the F1-Score value, for the classification of Indonesian New Translation Bible texts, is to use a Bi-LSTM model with Skip-Gram embedding and a vector size of 300.

In short, the experimental results in Tables VI and VII show that the Skip-Gram method with a vector size of 300 yields the highest accuracy, at 0.9194 and 0.9231, in both models. The performance evaluation of the model shows that Bi-LSTM provides higher accuracy compared to LSTM. These outcomes signify that the Bi-LSTM model can capture more complex and contextual relationships between words, providing better accuracy results.

Next, the F1-score value presents a similar trend to accuracy, as shown in Figs. 16 and 17. Figure 16 compares the F1-Score of two word embedding models, CBOW and Skip-Gram, with the LSTM model across three vector sizes (50, 100, and 300). The Y-axis represents the F1-Score, a metric that harmonizes precision and recall to evaluate the performance of a classification model, specifically in cases of imbalanced class distributions [14]. The X-axis shows the vector sizes used in the experiment, which are the dimensions of the numerical word representations

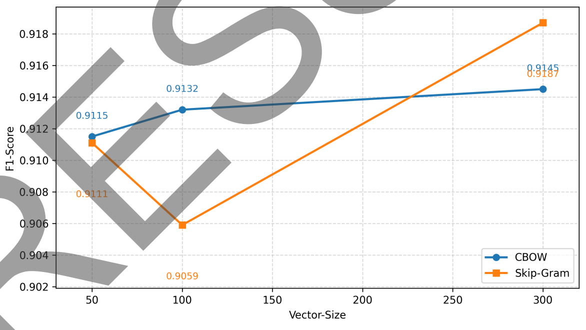


Fig. 16. F1-Score performance of Long Short-Term Memory (LSTM) models with different embedding types and vector sizes.

produced by CBOW and Skip-Gram models.

The blue line illustrates how the F1-Score changes for the LSTM model when using CBOW embedding with different vector sizes. It signifies a steady increase in F1-Score as the vector size increases, from 0.9115 with 50 vector size to 0.9132 of 100 and reaching 0.9145 of 300. Meanwhile, the orange line illustrates the variation in F1-score when using Skip-Gram embedding. Initially, there is a decrease in F1-Score from 0.9111 at 50 vector size to 0.9099 at 100. However, a significant improvement is visible at 300 vector size, where F1-Score peaks at 0.9187.

CBOW shows consistent improvement in F1-Score with increasing vector size, signifying better performance with larger vectors. Skip-Gram, while initially underperforming at a vector size of 100, achieves a significant boost at a vector size of 300, surpassing the performance of CBOW at this size. The outcome shows that for the LSTM model in the specific task, using Skip-Gram with a vector size of 300 produces the best

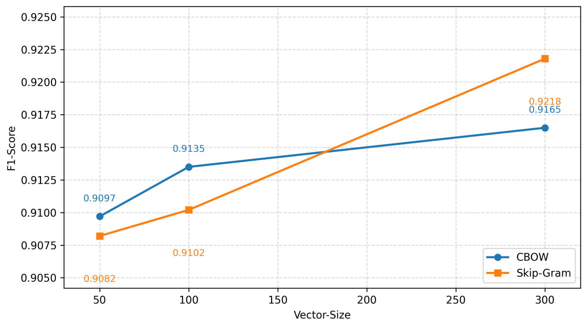


Fig. 17. F1-Score performance of Bidirectional Long Short-Term Memory (Bi-LSTM) models with different embedding types and vector sizes.

performance. Meanwhile, the Bi-LSTM model with the Skip-Gram method at a vector size of 300 produces the highest F1-Score value. The results reflect the excellent performance of the Bi-LSTM model in detecting Old and New Testaments classes with the best balance of all the experiments conducted in the research.

The new graphs in Fig. 17 compare F1-Scores between CBOW and Skip-Gram models using a Bi-LSTM model across three different vector sizes: 50, 100, and 300. Some significant differences are observed in the results compared to the previous graph. In Fig. 17, F1-Scores for both CBOW and Skip-Gram models are slightly higher across all vector sizes. Specifically, F1-Score of CBOW model rises from 0.9097 at vector size of 50 to 0.9165 at 300. Meanwhile, the Skip-Gram model starts at 0.9082 for a vector size of 50, slightly lower than its starting point in the previous graph. However, the model shows a more dramatic improvement, achieving an F1-Score of 0.9218 at a vector size of 300. It differs from the previous graph, where the Skip-Gram model experiences a dip in performance at a vector size of 100 before recovering. Figure 19 also shows a steady upward trend for both models without any dips. The result signifies a more stable and reliable improvement in performance, particularly for the Skip-Gram model, as the vector size increases.

Figure 18 shows CM for the LSTM model concerning the CBOW and Skip-Gram methods from Word2Vec. In these results, the performance of the two models in classifying Indonesian New Translation Bible texts varies depending on the embedding method used. In the LSTM model with CBOW embedding for a vector size of 300, the classification accuracy shows an uneven distribution of predictions compared to Skip-Gram, where minority classes tend to be less well classified. This outcome is observed from the higher number of TP, although there is a slight increase in FP.

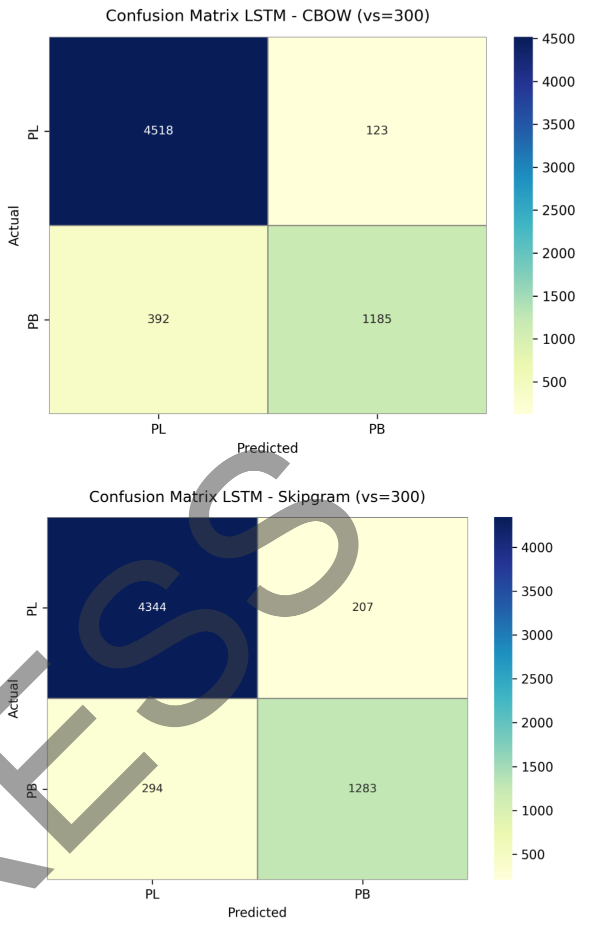


Fig. 18. Comparison of the Confusion Matrix (CM) for Long Short-Term Memory (LSTM) models across embedding schemes at vector size 300 with Old Testament (PL) and New Testament (PB).

Next, the analysis observes the CM result in Bi-LSTM, as shown in Fig. 19. The Bi-LSTM model with Skip-Gram embedding presents more consistent results with fewer classification errors (FP and FN) compared to CBOW. In general, these CM results show that Bi-LSTM with Skip-Gram produces more stable performance compared to LSTM, although LSTM with Skip-Gram also offers power in handling certain class variations.

Then, a paired t-test is conducted on ten independent training runs during the process to confirm the statistical significance of the performance advantage of Bi-LSTM. The results show a significant difference in accuracy between Bi-LSTM and LSTM ($p < 0.01$), with a mean improvement of 0.37% (standard deviation = 0.12). This improvement is meaningful in the context of low-resource language processing, where even marginal additions often require substantial

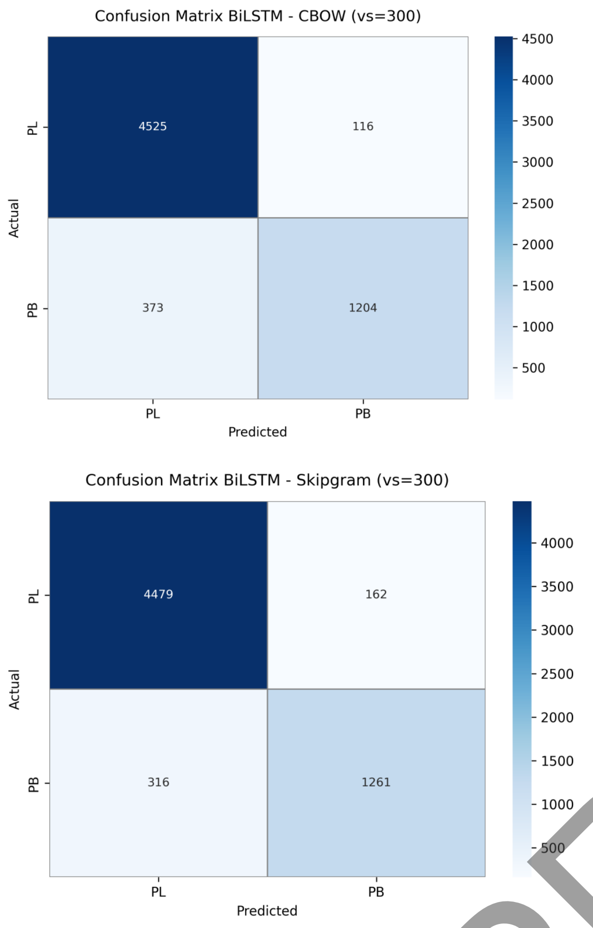


Fig. 19. Comparison of the Confusion Matrix (CM) for Bidirectional Long Short-Term Memory (Bi-LSTM) models across embedding schemes at vector size 300 with Old Testament (PL) and New Testament (PB).

architectural innovation.

These results challenge the assumption that high-resource language models can be directly applied to low-resource contexts without architectural adaptation. By showing the importance of bidirectional processing and context-aware embedding, the analysis advances the development of culturally and linguistically inclusive NLP systems. It means that it is important to customize the architecture of NLP models for languages with limited data. Bi-LSTM, with its advantage in capturing bidirectional context, points the way towards building NLP systems that are more equitable, inclusive, and suited to language-cultural diversity.

The research findings are consistent with previous research. For instance, previous research has reported that Bi-LSTM is more capable of capturing nuanced information compared to unidirectional models [23]. Meanwhile, previous studies have also emphasized that Bi-LSTM outperforms unidirectional LSTM in text classification tasks because it considers context from both directions [21, 22]. In line with these studies, the results of this research further reinforce the evidence that Bi-LSTM provides better performance in handling contextual information. However, the research novelty lies in demonstrating that the developed model is also effective in addressing the more complex case of text classification using the Bible written in a low-resource language, thereby filling a study gap and contributing to the broader understanding of Bi-LSTM applications in underrepresented linguistic contexts.

Last, the research contributes to the NLP literature by providing a novel comparative analysis of LSTM and Bi-LSTM on Indonesian Bible text, a low-resource and linguistically complex corpus. The findings empirically validate the advantage of bidirectional context modeling in Bi-LSTM and highlight the effectiveness of Skip-Gram embedding for religious text classification. Beyond technical outcomes, the research offers a reproducible pipeline that can inform the development of NLP tools for Indonesian and other low-resourced languages, with potential applicability in digital humanities.

IV. Conclusion

The classification of religious texts, such as the Indonesian New Translation Bible, has unique challenges in NLP, particularly in low-resource languages with rich morphological and contextual complexity. To address the challenges, the research conducts a comparative analysis of LSTM and Bi-LSTM, using Word2Vec embedding, for the classification of Indonesian New Translation Bible texts. The major results obtained during the analysis process are as follows. First, the Bi-LSTM model consistently outperforms LSTM, achieving 92.31% accuracy compared to 91.94% for LSTM with a 300-dimensional Skip-Gram embedding. This improvement originates from bidirectional context modelling of Bi-LSTM, which captures nuanced dependencies critical for distinguishing Old and New Testaments themes. Second, although the performance of CBOW is consistent across smaller vector sizes (50 and 100), Skip-Gram presents a clear advantage with a 300-dimensional vector space. The model demonstrates its ability to resolve semantic ambiguities in context-rich texts, such as the Bible. Third, CM shows that LSTM has difficulty in recalling New Testament texts (the minority class) when combined with CBOW embedding. Meanwhile, Bi-LSTM is more robust in avoiding misclassifications.

The research enhances culturally inclusive NLP tools by demonstrating the effectiveness of Bi-LSTM and Skip-Gram embedding for classifying the Indonesian Bible. It highlights the importance of innovative architecture and context-sensitive embedding for low-resource languages, setting the stage for a global analysis of religious and heritage texts. The research contributed significantly to the fields through a novel methodological framework for evaluating LSTM and Bi-LSTM performance in low-resource languages. The contribution is validated on the linguistic complexity of Indonesian Bible. Technically, it provides empirical validation of bidirectional architecture of Bi-LSTM and the effectiveness of Skip-Gram in the context of religious text classification. In a practical manner, the analysis delivers a reproducible pipeline for processing Indonesian Bible text. It also offers valuable applicability to digital humanities studies and the development of multilingual NLP tools.

The research limitations include the focus on a single religious text, the Indonesian New Translation Bible, and the hyperparameter optimization is constrained to Word2Vec and LSTM variants. Future researchers should extend the proposed framework to other low-resource languages in Indonesia, such as Sundanese and Javanese Bible translations. Furthermore, exploration of advanced contextual embedding, such as BERT-ID or IndoBERT, should be warranted to improve semantic representation. The integration of Bi-LSTM architecture with attention mechanisms presents a promising avenue for improved accuracy. Finally, methods such as Synthetic Minority Oversampling Technique (SMOTE) or Focal Loss should be implemented to mitigate class imbalance resulting from the uneven distribution of Old Testament (74.4%) and New Testament (25.6%) in the dataset used in the research.

Author Contribution

Conceived and designed the analysis, M. S. S., S. T., and D. B. N.; Collected the data, M. S. S.; Contributed data or analysis tools, M. S. S., S. T., and H. A. P.; Performed the analysis, M. S. S.; Wrote the paper, M. S. S.; Converted the article to be a LaTeX format, M. S. S.; Provided the feedback and improvements to Python code, S. T.; Provided input and improvements to the writing structure, D. B. N.; and Provided input and improvements to the writing structure and mathematical formulas; H. A. P.

Data Availability

The data that support the findings of the research are available in GiHub at https://github.com/MarthenSS/Bible-Dataset. These data are derived from the following resources available in the public domain at https://alkitab.mobi, manually extracted and structured into a CSV and XLSX files by the authors.

References

[1] E. Saragih, “Analysis of grammatical metaphors in doctrinal verses of Alkitab Terjemahan Baru 1974,” in *Conference of ELT, Linguistics, Literature and Translation ICELT5th Graduate Schoolof NHU*, North Sumatera, Indonesia, February 2018, pp. 84–96.

[2] M. Martinjak, D. Lauc, and I. Skelac, “Towards analysis of biblical entities and names using deep learning,” *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 5, pp. 491–497, 2023.

[3] L. Deng and Y. Liu, *Deep learning in natural language processing*. Springer, 2018.

[4] J. Kumar, R. Goomer, and A. K. Singh, “Long Short Term Memory Recurrent Neural Network (LSTM-RNN) based workload forecasting model for cloud datacenters,” *Procedia computer science*, vol. 125, pp. 676–682, 2018.

[5] E. M. Kelley, “The principles, process, and purpose of the canon of scripture,” *Diligence: Journal of the Liberty University Online Religion Capstone in Research and Scholarship*, vol. 5, no. 1, pp. 1–27, 2020.

[6] Lembaga Alkitab Indonesia, “Meluruskan fakta yang dipelintir tentang terjemahan LAI,” 2024. [Online]. Available: http://bit.ly/3VsXKDx

[7] A. Tarlam, “Hermeneutik dan kritik Bible,” *AL-KAINAH: Journal of Islamic Studies*, vol. 1, no. 2, pp. 103–118, 2022.

[8] S. Cahyawijaya, G. I. Winata, B. Wilie, K. Vincentio, X. Li, A. Kuncoro, S. Ruder, Z. Y. Lim, S. Bahar, M. L. Khodra, A. Purwarianti, and P. Fung, “IndoNLG: Benchmark and resources for evaluating Indonesian natural language generation,” 2021. [Online]. Available: http://arxiv.org/abs/2104.08200

[9] M. N. A. D., I. Godbole, P. M. Kapparad, and S. Bhattacharjee, “Comparative analysis of religious texts: NLP approaches to the Bible, Quran, and Bhagavad Gita,” in *Proceedings of the New Horizons in Computational Linguistics for Religious Texts*. Abu Dhabi, UAE: Association for Computational Linguistics, 2025, pp. 1–10.

[10] H. J. De Jonge, “Erasmus’s translation of the New Testament: Aim and method,” *The Bible Translator*, vol. 67, no. 1, pp. 29–41, 2016.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[12] S. Sivakumar, L. S. Videla, T. R. Kumar, J. Nagaraj, S. Itnal, and D. Haritha, “Review on

Word2vec word embedding neural net," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. Trichy, India: IEEE, Sep. 10–12, 2020, pp. 282–290.

[13] S. H. Lee, H. Lee, and J. H. Kim, "Enhancing the prediction of user satisfaction with Metaverse service through machine learning," *Computers, Materials & Continua*, vol. 72, no. 3, pp. 4983–4997, 2022.

[14] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, and L. He, "A survey on text classification: From traditional to deep learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 2, pp. 1–41, 2022.

[15] I. R. Hendrawan, E. Utami, and A. D. Hartanto, "Analisis perbandingan metode TF-IDF dan Word2vec pada klasifikasi teks sentimen masyarakat terhadap produk lokal di Indonesia," *Smart Comp: Jurnalnya Orang Pintar Komputer*, vol. 11, no. 3, pp. 497–503, 2022.

[16] A. Suryadibrata, J. C. Young *et al.*, "Embedding from Language Models (ELMos)-based Dependency Parser for Indonesian language," *International Journal of Advances in Soft Computing & Its Applications*, vol. 13, no. 3, pp. 2–11, 2021.

[17] M. Rhanoui, S. Yousfi, M. Mikram, and H. Merizak, "Forecasting financial budget time series: ARIMA random walk vs LSTM neural network," *IAES International Journal of Artificial Intelligence*, vol. 8, no. 4, pp. 317–327, 2019.

[18] D. T. Hermanto, A. Setyanto, and E. T. Luthfi, "Algoritma LSTM-CNN untuk binary klasifikasi dengan Word2vec pada media online," *Creative Information Technology Journal*, vol. 8, no. 1, pp. 64–77, 2021.

[19] Y. Zhang, Q. Liu, and L. Song, "Sentence-state LSTM for text representation," 2018. [Online]. Available: http://arxiv.org/abs/1805.02474

[20] B. Jang, M. Kim, G. Harerimana, S. U. Kang, and J. W. Kim, "Bi-LSTM model to increase accuracy in text classification: Combining Word2vec CNN and attention mechanism," *Applied Sciences*, vol. 10, no. 17, pp. 1–14, 2020.

[21] J. L. C. Yew, "Comparative study on SVM and Bi-LSTM combined with Word2vec and TFIDF on the sentiment analysis of movie," Ph.D. dissertation, Tilburg University, thesis.

[22] A. M. Almars, "Attention-based Bi-LSTM model for Arabic depression classification," *Computers, Materials & Continua*, vol. 71, no. 2, pp. 3091–3106, 2022.

[23] S. F. Sabbeh and H. A. Fasihuddin, "A comparative analysis of word embedding and deep learning for Arabic sentiment classification," *Electronics*, vol. 12, no. 6, pp. 1–16, 2023.

[24] S. Trihandaru, H. A. Parhusip, B. Susanto, and C. F. R. Putri, "Word cloud of UKSW lecturer research competence based on Google Scholar data," *Khazanah Informatika: Jurnal Ilmu Komputer dan Informatika*, vol. 7, no. 2, pp. 52–59, 2021.

[25] A. Géron, *Hands on machine learning with Scikit-Learn, Keras & TensorFlow*. O'Reiley Media, 2019.

[26] D. Qiu, H. Jiang, and S. Chen, "Fuzzy information retrieval based on continuous bag-of-words model," *Symmetry*, vol. 12, no. 2, pp. 1–11, 2020.

[27] H. Park and J. Neville, "Generating post-hoc explanations for Skip-gram-based node embeddings by identifying important nodes with bridgeness," *Neural Networks*, vol. 164, pp. 546–561, 2023.

[28] J. K. Yi and Y. F. Yao, "Advancing quality assessment in vertical field: Scoring calculation for text inputs to large language models," *Applied Sciences*, vol. 14, no. 16, pp. 1–15, 2024.

[29] S. Skansi, *Introduction to deep learning: From logical calculus to artificial intelligence*. Springer International Publishing, 2018.

[30] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on Long Short-Term Memory networks for time series prediction," *Procedia Cirp*, vol. 99, pp. 650–655, 2021.

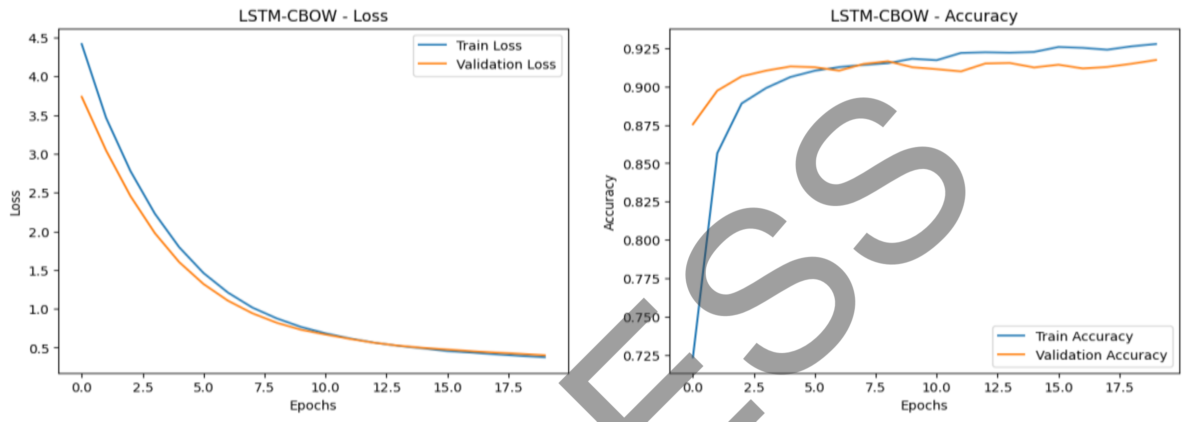
[31] M. Kowsher, A. Tahabilder, M. Z. I. Sanjid, N. J. Prottasha, M. S. Uddin, M. A. Hossain, and M. A. K. Jilani, "LSTM-ANN & BiLSTM-ANN: Hybrid deep learning models for enhanced classification accuracy," *Procedia Computer Science*, vol. 193, pp. 131–140, 2021.

[32] D. Krstinić, M. Braović, L. Šerić, and D. Božić-Štulić, "Multi-label classifier performance evaluation with confusion matrix," *Computer Science & Information Technology*, vol. 1, no. 2020, pp. 1–14, 2020.

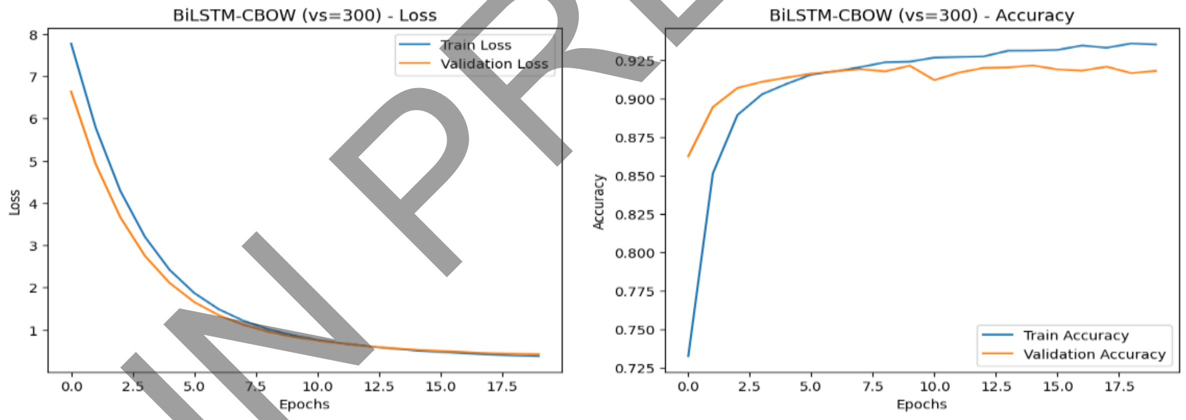
[33] A. W. Putri, "Implementasi Artificial Neural Network (ANN) backpropagation untuk klasifikasi jenis penyakit pada daun tanaman tomat," *MATHunesa: Jurnal Ilmiah Matematika*, vol. 9, no. 2, pp. 344–350, 2021.

Appendix

The Appendix can be seen in the next page.

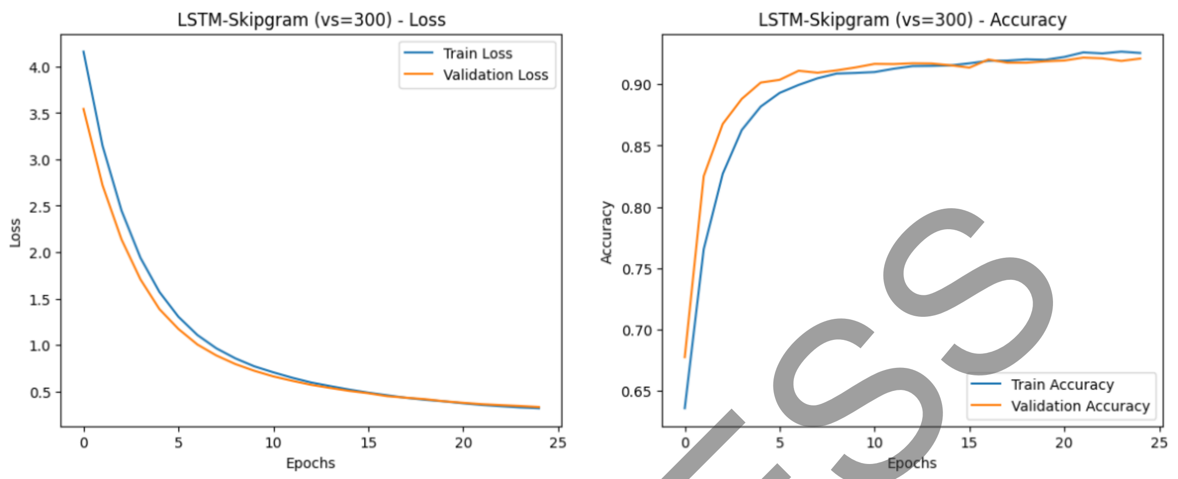


(a) Loss Graph

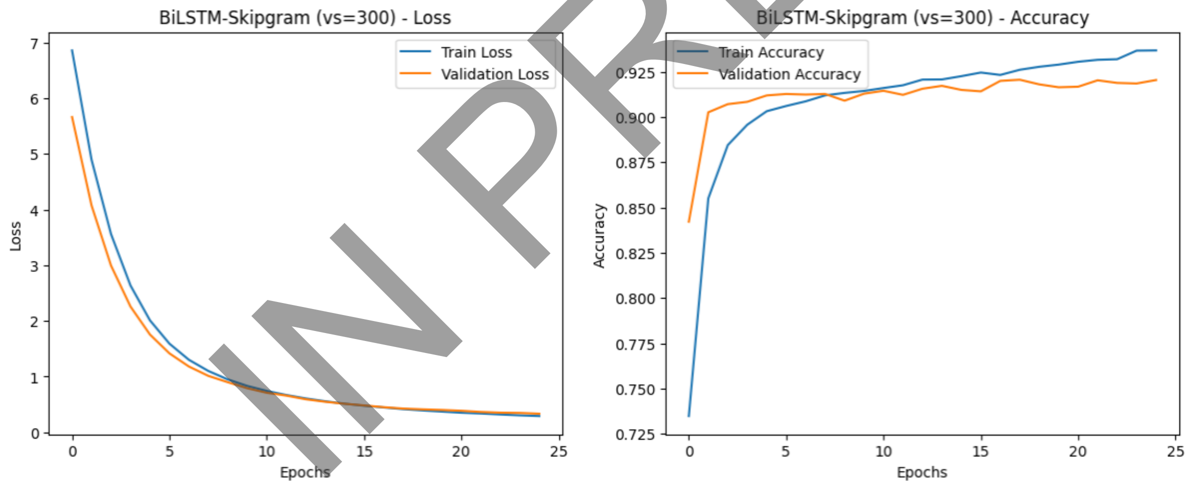


(b) Accuracy Graph

Fig. A1. Performance of Long Short-Term Memory (LSTM) and Bidirectional LSTM (Bi-LSTM) model with Continuous Bag of Words (CBOW) embedding (vector size 300).



(a) Loss Graph



(b) Accuracy Graph

Fig. A2. Performance of Long Short-Term Memory (LSTM) and Bidirectional LSTM (Bi-LSTM) model with Skip-Gram embedding (vector size 300).