A Hybrid Regex-Based Preprocessing and LSTM Neural Network Approach for Sentiment Analysis of Tokopedia Reviews

Razi Rachman Widyadhana - 13523004
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: rr.widyadhana@gmail.com , 13523004@std.stei.itb.ac.id

***Abstract*— The rapid growth of Indonesia's e-commerce sector, particularly marketplaces like Tokopedia, has heightened the need for effective sentiment analysis to understand consumer preferences and perceptions. This study proposes a hybrid approach integrating regex-based text preprocessing with Long Short-Term Memory (LSTM) neural networks to analyze sentiments in Tokopedia reviews. The regex-based preprocessing pipeline cleans and standardizes raw text data by removing noise such as special characters, stopwords, and irrelevant patterns, while tokenization and padding ensure compatibility with machine learning models. The LSTM model captures contextual patterns in the processed sequences to classify sentiments as positive, negative, or neutral. This hybrid method provides a robust framework for extracting meaningful insights from unstructured review data, offering practical and theoretical contributions to enhance service quality and inform stakeholders in Indonesia's marketplace ecosystem.**

Keywords—Sentiment Analysis; Regular Expression; LSTM; Tokopedia; E-commerce;

I. Introduction

In the era of globalization and significant growth in information technology, Indonesia's e-commerce sector, particularly marketplaces like Tokopedia, has seen massive development [1]. This phenomenon is driven by changing consumer behavior, as people shift from traditional shopping to online platforms because of convenience and accessibility. Marketplaces enable businesses to reach broader audiences while providing consumers easy ways to satisfy their shopping needs. As the marketplace ecosystem grows, understanding public sentiment about its services and products is essential for businesses and market observers to meet consumer expectations.

Fig. 1. Indonesia's e-commerce reports in 2022. Adapted from [2].

To gain insights from public sentiment effectively, reliable technological methods are needed to help the extraction process from online data. This study proposes a hybrid approach using Regex-based preprocessing combined with LSTM (Long Short-Term Memory) neural networks for sentiment analysis of Tokopedia reviews. Regex-based preprocessing cleans and organizes raw text data to ensure only relevant information is used, whereas LSTM (Long Short-Term Memory) neural network captures patterns and context in reviews for accurate sentiment classification.

This hybrid approach aims to identify distinct patterns in consumer preferences and perceptions within Tokopedia's ecosystem. Combining targeted regex patterns for cleaning and feature extraction with the sequence-modeling capabilities of LSTM provides a solid pipeline for classifying sentiments as positive, negative, or neutral.

Therefore, this paper provides practical and theoretical contributions to support improvements in sentiment analysis, highlighting the significant role of a hybrid method for handling unstructured data to provide valuable insights for Tokopedia stakeholders and policymakers for adequate service quality and strengthen Indonesia's marketplace ecosystem.

II. Theoritical Foundation

A. String

A string is defined as a sequence or ordered collection of characters that represent textual data. These characters can comprise a mix of letters, digits, symbols, and various special characters. For example, `"Hello world!"`, `"1234"`, `"#@$!"`, and `"email@example.com"` all qualify as valid strings. In many programming languages, strings are treated as a fundamental data type, allowing developers to declare and manipulate them easily. In both Python and Java, strings can be defined using either double quotes (e.g., `"hello"`) or single quotes (e.g., `'world'`). These languages provide an extensive array of functions and methods for processing strings, which includes operations such as concatenation, slicing, searching, and replacing.

B. String Matching

String matching algorithms actively search for a specific sub-pattern, pattern, or word within a larger text or string. These algorithms aim to locate the presence of a given sub-pattern or pattern within a broader text. The general definition of string matching includes the following components:

1. T: The text, a string comprising n characters.
2. P: The pattern, a string of m characters (where m < n) that the algorithm searches for within the text.

For example, consider the following instance of string matching:

1. T: `spain stays mainly on the plain`
2. P: `main`

These string matching algorithms can be applied to various fields, for example, search features in text editors or PDF readers, keyword search engines like Google, image analysis such as fingerprint recognition, and bioinformatics such as DNA sequence matching.

C. *Regex (Regular Expression)*

Regular expression (regex or regexp for short) is a special text string that describes search pattern in any given text (string), like a supercharged set of wildcards [3].

```
"^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$"

/^\d{2}\/\d{2}\/\d{4}$/
```

Ever seen these unusual symbols before? Most developers use regular expressions to validate passwords, extract dates from text, and tackle many other tasks. Regular expressions are a powerful concept in computer science. It help us to search and extract data from text. This text can include system logs, documents, PDF, source code, and many else

Regular expressions are utilized in numerous modern programming languages and frameworks, such as Python, JavaScript, Java, TypeScript, Rust, and Go. Practical implementations of regular expressions often diverge from their theoretical foundations. Regex was developed under the influence of theoretical regular expressions [4]. In theoretical regular expressions, the basic form of regex is shown below.

Options | **Quick Reference**

`.`	Any character except newline.
`\.`	A period (and so on for `\*`, `\(`, `\\`, etc.)
`^`	The start of the string.
`$`	The end of the string.
`\d,\w,\s`	A digit, word character `[A-Za-z0-9_]`, or whitespace.
`\D,\W,\S`	Anything except a digit, word character, or whitespace.
`[abc]`	Character a, b, or c.
`[a-z]`	a through z.
`[^abc]`	Any character except a, b, or c.
`aa|bb`	Either aa or bb.
`?`	Zero or one of the preceding element.
`*`	Zero or more of the preceding element.
`+`	One or more of the preceding element.
`{n}`	Exactly *n* of the preceding element.
`{n,}`	*n* or more of the preceding element.
`{m,n}`	Between *m* and *n* of the preceding element.
`??,*?,+?, {n}?`, etc.	Same as above, but as few as possible.
`(expr)`	Capture *expr* for use with `\1`, etc.
`(?:expr)`	Non-capturing group.
`(?=expr)`	Followed by *expr*.
`(?!expr)`	Not followed by *expr*.

Near-complete reference

Fig. 2. Common Regex notation and its explanation. Adapted from [4].

Regular expressions offer remarkable flexibility, allowing developers to combine various operations seamlessly to create complex text search patterns as shown below.

operation	regular expression	matches ("in specified set")	does not match ("not in specified set")
concatenation	`aabaab`	aabaab	*every other string*
wildcard	`.u.u.`	cumulus jugulum	succubus tumultuous
union	`aa | baab`	aa baab	*every other string*
closure	`ab*a`	aa abbba	ab ababa
parentheses	`a(a|b)aab`	aaaab abaab	*every other string*
	`(ab)*a`	a abababa	aa abbba

regular expression	matches	does not match
`.*spb.*` *contains the trigraph* **spb**	raspberry crispbread	subspace subspecies
`a* | (a*ba*ba*ba*)*` *multiple of three* **b**'*s*	bbb aaa bbbaababbaa	b bb baabbbaa
`.*0....` *fifth to last digit is* **0**	1000234 98701234	111111111 403982772
`gcg(cgg|agg)*ctg` *fragile X syndrome indicator*	gcgctg gcgcggctg gcgcggaggctg	gcgcgg cggcggcggctg gcgcaggctg

Fig. 3.Examples of Regular Expression Usage. Adapted from [5].

D. *LSTM (Long Short-Term Memory) Neural Network*

LSTM stands for Long Short-Term Memory, and it is a type of recurrent neural network (RNN) architecture that is commonly used in natural language processing, speech recognition, and other sequence modeling tasks. Introduced by Hochreiter & Schmidhuber (1997), LSTMs are explicitly designed to avoid the long-term dependency problem.

Traditional RNNs are designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. However, they struggle with long-term dependencies due to:

- *Vanishing Gradients*: During backpropagation, gradients can shrink exponentially, making it difficult for the network to learn long-range dependencies.
- *Exploding Gradients*: Conversely, gradients can also grow exponentially, causing numerical instability.

Unlike a traditional RNN, which has a simple structure of input, hidden state, and output, an LSTM has a more complex structure with additional memory cells and gates that allow it to selectively remember or forget information from previous time steps. These gates enable LSTMs to retain important information for long durations, making them highly effective for tasks like time-series prediction, natural language processing, and speech recognition.

In standard RNNs, repeating module will have a very simple structure, such as a single tanh layer. Repeating module in an LSTM contains four interacting layers [6].

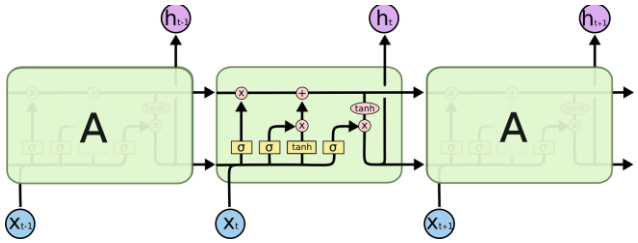


Fig. 4. The repeating module in an LSTM contains four interacting layers. Adapted from [7].

At the heart of an LSTM network is the memory cell, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt, which retains information over time. It runs straight down the entire chain, with only some minor linear interactions.

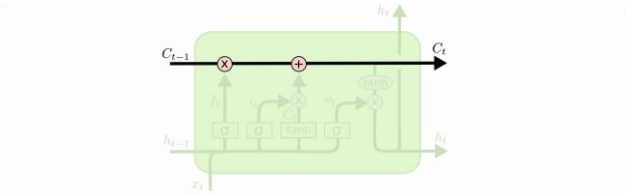
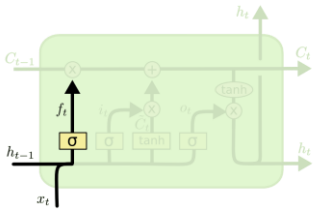


Fig. 5. Cell State in LSTM. Adapted from [7].

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. The cell state is manipulated by three types of gates:

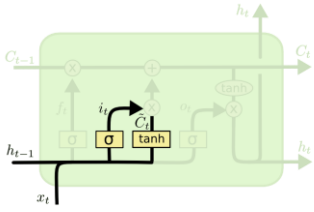
1) Forget Gate: Sigmoid layer that decide what information to keep and what to forget. It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 0 means to 'remove completely' while 1 means to 'keep information' as it is.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fig. 6. Forget Gate mechanism. Adapted from [7].

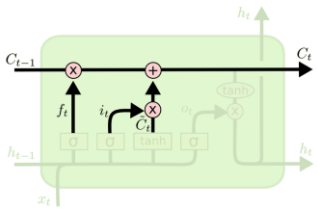
2) Input Gate & Tanh Layer: Determines which new information to add to the cell state. Input gate layer decides which values we'll update. Tanh layer creates a vector of new candidate values, $\tilde{C}_t$ that could be added to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 7. Input Gate with Tanh Layer mechanism. Adapted from [7].

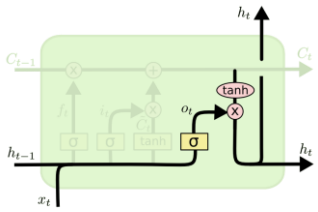
Multiply the old state by f_t to forget information from context vector and $i_t * C_t$ is new candidate values scaled by how much they needs to be updated.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 8. Updating the Cell State Values. Adapted from [7].

3) Output Gate: The output is a modified version of the cell state. For this sigmoid decides what part needs to be modified, which is multiplied with output after Tanh (used for scaling), resulting in hidden state output.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 9. Output Gate mechanism. Adapted from [7].

These gates are learned during training and are crucial for regulating the information flow.

E. *Attention Mechanism*

RNNs and LSTMs have been used widely for processing and making sense of sequential data. The usual approach is the encoder-decoder architecture for seq2seq tasks.

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus.

Attention mechanism helps to look at all hidden states from encoder sequence for making predictions unlike vanilla Encoder-Decoder approach.

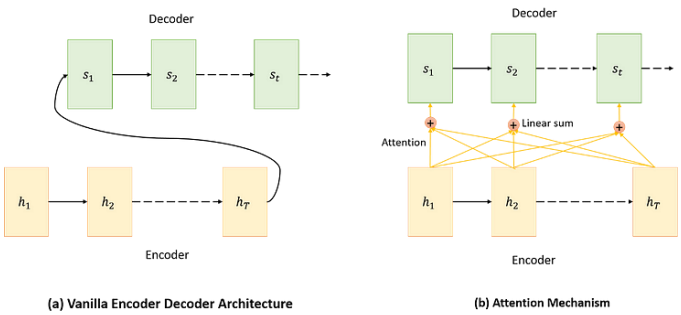


Fig. 10. Simple vs Attention based encoder-decoder architectures. Adapted from [8].

In a simple Encoder-Decoder architecture the decoder is supposed to start making predictions by looking only at the final output of the encoder step which has condensed information. On the other hand, attention based architecture attends every hidden state from each encoder node at every time step and then makes predictions after deciding which one is more informative.

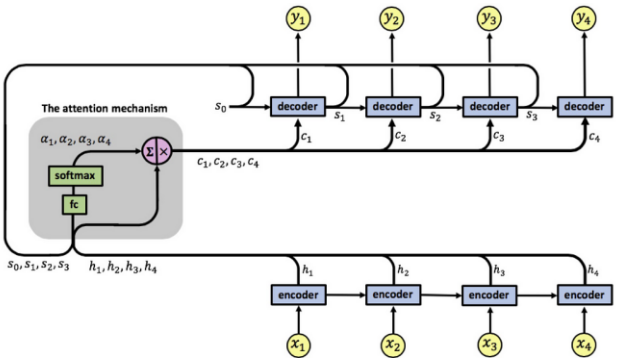


Fig. 11. Simplied version of complete Attention based architecture. Adapted from [8].

The attention mechanism involves four main steps. First, the encoder transforms the input into a sequence of hidden representations. Next, the model computes attention weights by comparing the decoder's current state with each encoder output. Then it creates a context vector as the weighted sum of those encoder representations. Finally, the decoder uses this context vector to generate the translated output.

F. Evaluation Metrics

In evaluating the performance of a classification model, selecting appropriate metrics is essential to ensure that the predicted results are fit for purpose analysis. Some metrics have different interpretations in the context of evaluating multi-class classification methods. For a multiclass classification problem with *C* classes (labeled 0, 1, 2, ..., C-1), the confusion matrix M is a C×C matrix where:

$$\mathrm{M} = \left(\begin{bmatrix} M_{11} & \cdots & M_{1c} \\ \vdots & \ddots & \vdots \\ M_{c1} & \cdots & M_{cc} \end{bmatrix} \right)$$

Where M_{ij} represents the number of samples from true class *i* that were predicted as class *j*.

1) Precision: The precision denotes the proportion of the retrieved samples which are relevant and is calculated as the ratio between correctly classified samples and all samples assigned to that class.

$$Prec_i = \frac{TP_i}{TP_i + FP_i} = M_{ii} \sum_j M_{ji} \tag{1}$$

2) Recall: The recall, also known as the sensitivity or True Positive Rate (TPR), denotes the rate of positive samples correctly classified, and is calculated as the ratio between correctly classified positive samples and all samples assigned to the positive class.

$$Rec_i = \frac{TP_i}{TP_i + FN_i} = M_{ii} \sum_j M_{ij} \tag{2}$$

3) F1 score: The F1 score is the harmonic mean of precision and recall, meaning that it penalizes extreme values of either. This metric is not symmetric between the classes, it depends on which class is defined as positive and negative.

$$F1_i = 2 \times \frac{Prec_i \times Rec_i}{Prec_i + Rec_i} = \frac{2 \times TP_i}{2 \times TP_i + FP_i + FN_i} \tag{3}$$

G. Sentiment Analysis

Sentiment analysis, also known as opinion mining, opinion analysis, or subjective analysis, is a computational method used to determine the sentiment expressed by individuals towards a product or service through automatic text processing using natural language processing techniques [9].

In sentiment analysis, data mining is used to analyze, process, and extract textual data from entities such as services, products, individuals, phenomena, or specific topics. The analysis can cover review texts, forums, tweets, or blog posts, and begins with preprocessing steps like tokenization, stop-word removal, stemming, sentiment detection, and sentiment classification. Sentiment analysis can be categorized into three levels:

1) Document level: aims to classify the sentiment expressed across the entire content of a document, such as news articles or research papers. It provided an overall sentiment assessment of the document.

2) Sentence-level: focuses on classifying the sentiment of each sentence within a document. This level is beneficial for shorter texts, such as comments, social media posts, or customer reviews, where each sentence can independently express a sentiment.

3) Aspect level, aims to identify and classify sentiments based on different aspects or topics within a text. It involves extracting specific aspects of interest and associating sentiments with those aspects. This approach enables a more granular understanding of sentiments expressed within the text, providing insights into various aspects and their corresponding sentiments.

Aspect-based sentiment analysis typically involves two main tasks: sentiment classification and aspect classification. Sentiment classification determines the sentiment polarity (e.g., positive, negative, or neutral) associated with each aspect. In contrast, aspect classification involves identifying and categorizing the aspects or topics discussed in the text.

III. Implementation

The instruments used in this paper consist of two main components, namely hardware and software. The hardware specifications used are: Intel i7-8550U Processor, Intel UHD Graphics 620 GPU, and 8 GB RAM. Then, the software specifications used are Python 3.12.4 programming language and Keras library for implementing the LSTM neural network, as well as Jupyter Notebook and Kaggle for the program kernel.

A. Dataset Exploration

In this phase, the dataset was obtained from [10], which contains detailed information on 14,175 Tokopedia customer reviews. The dataset was split into 12,564 (88.63%) of training set and 1,611 (11.37%) of testing sets without the review status label for evaluation.

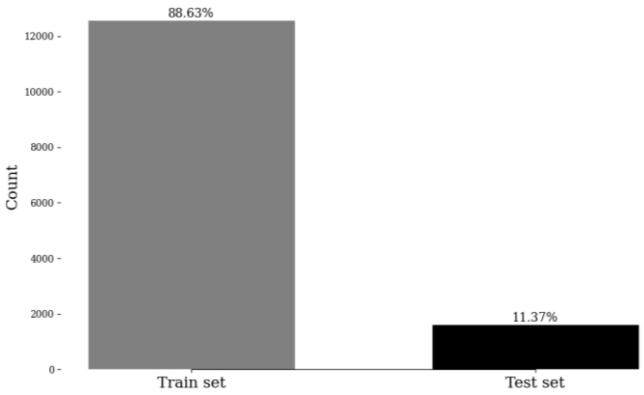


Fig. 12. Data distribution between Training and Testing set.

In the training set, the distribution between Good and Bad labels are balanced, with Neutral having far little amount from other label.

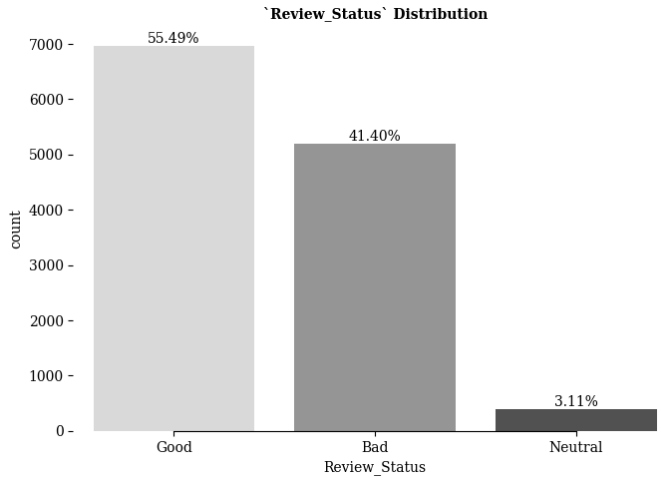


Fig. 13. Review Status distribution in Training Set

Here are some examples of the reviews in the dataset for more information.

	id	Review	Stars	Review_Status
0	6674226eaa0c235d0d705893	packing aman dan safety, sesuai dgn pesanan 👍👍👍	5	Good
1	665eb05425b38bc6834da057	Mantap, biar bintang yang berbicara	5	Good
2	665e0be722d56c91a7e336b7	jasa pengirimanya lama banget sampe 11 hari	1	Bad
3	665e0707aba34dc4a6b3f2c3	sampai rumah dicolok bisa abis itu mati, gabis...	1	Bad
4	665e011c7906f45eeb0738a2	Ini ga bisa dipake untuk first media dekorder ...	1	Bad

Fig. 14. Samples of Reviews in the Training Set.

B. *Data Preprocessing*

Text preprocessing is a fundamental stage in the sentiment analysis pipeline, aimed at cleaning, standardizing, and optimizing text representation for machine learning algorithms. In the context of sentiment analysis, preprocessing not only involves removing noise from the data but also preserving relevant semantic information for determining sentiment polarity.

The text preprocessing pipeline is designed to transform raw text data into a clean, structured format suitable for natural language processing tasks, specifically for training a machine learning model. It involves a series of steps: the raw text undergoes comprehensive cleaning to remove unwanted characters, URLs, emojis, and formatting issues while standardizing the format using Regular Expressions. Indonesian stopwords are also filtered out to focus on meaningful content words.

```python
def clean(text):
    text = str(text)

    # 1. Literal \t, \n, \u, and backslash
    text = re.sub(r'(\\t|\\n|\\u|\\\\)', ' ', text)

    # 2. Mention/hashtag and URL (e.g. "http://…")
    text = re.sub(r'([@#][A-Za-z0-9_]+)|(\w+://\S+)', ' ', text)

    # 3. Emoji to shortcode (e.g. 😉 → :smile:)
    text = emoji.demojize(text)

    # 4. non-ASCII characters
    text = re.sub(r'[^\x00-\x7F]+', '', text)

    # 5. Punctuation and Numbers
    text = re.sub(r'[^A-Za-z\s]', '', text)

    # 6. Single-char words
    text = re.sub(r'\b[A-Za-z]\b', ' ', text)

    # 7. Extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    # 8. Lowercase
    text = text.lower()

    return text
```

Fig. 15. Text Cleaning Process in Python

The cleaned text is tokenized using Keras' Tokenizer to convert words into numerical sequences with a vocabulary limit of 20,000 most frequent words. These sequences are padded to ensure uniform length for batch processing in neural networks. The data is split into training and validation sets with proper label encoding for model training and evaluation.

	Review_Initial	Review_Processed
0	packing aman dan safety, sesuai dgn pesanan 👍👍👍	[packing, aman, safety, sesuai, dgn, pesanan, ...
1	Mantap, biar bintang yang berbicara	[mantap, biar, bintang, berbicara]
2	jasa pengirimanya lama banget sampe 11 hari	[jasa, pengirimanya, banget, sampe]
3	sampai rumah dicolok bisa abis itu mati, gabis...	[rumah, dicolok, abis, mati, gabisa, retur]
4	Ini ga bisa dipake untuk first media dekorder ...	[ga, dipake, first, media, dekorder, klo, deko...

Fig. 16. Samples of Review after Preprocessing

Each step ensures the text is simplified, relevant, and compatible with machine learning algorithms, improving model performance and interpretability.

C. *Sentiment Analysis*

This modeling approach compares two neural network architectures for text classification: a baseline LSTM model and an enhanced LSTM with attention mechanisms. Both models use word embeddings to convert text into dense vector representations, followed by LSTM layers to capture sequential dependencies in the text. The key difference lies in the attention mechanism, which allows the second model to focus on the most relevant parts of the input sequence when making predictions. Both models are trained for multi-class classification with 3 output classes using categorical cross entropy loss and `Adam` optimizer.

```python
model_base = Sequential()
model_base.add(Embedding(vocab_size, embedding_dim))
model_base.add(LSTM(32, return_sequences=True))
model_base.add(LSTM(64))
model_base.add(Dense(64, activation='relu'))
model_base.add(Dense(num_classes, activation='softmax'))
model_base.compile(loss='categorical_crossentropy',
                   optimizer=Adam(),
                   metrics=['accuracy'])
model_base.fit(X_train, y_train, epochs=5,
               validation_data=(X_val, y_val))
model_base.summary()
```

Fig. 17. LSTM without Attention definition in Python

This baseline model follows a straightforward sequential architecture designed for text classification. It begins with an embedding layer that converts word indices into 128-dimensional dense vectors, creating meaningful representations of the vocabulary.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 243, 128)	1,859,712
lstm (LSTM)	(None, 243, 32)	20,608
lstm_1 (LSTM)	(None, 64)	24,832
dense (Dense)	(None, 64)	4,160
dense_1 (Dense)	(None, 3)	195

Fig. 18. LSTM without Attention Architecture

The model then employs two LSTM layers in succession: the first LSTM (32 units) with `return_sequences=True` processes the entire sequence and passes all hidden states to the next layer, while the second LSTM (64 units) processes these sequences and outputs only the final hidden state. This final state captures the overall context of the input sequence, which is then passed through a dense layer with `ReLU` activation for feature transformation before the final `softmax` layer produces probability distributions over the 3 classes.

```python
inputs = Input(shape=(None,))
x = Embedding(vocab_size, embedding_dim)(inputs)
x = LSTM(16, return_sequences=True)(x)
x = MultiHeadAttention(num_heads=2, key_dim=32)(x, x)
x = LSTM(16, return_sequences=True)(x)
x = MultiHeadAttention(num_heads=3, key_dim=32)(x, x)
x = LSTM(64)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
outputs = Dense(num_classes, activation='softmax')(x)

model_att = Model(inputs, outputs)
model_att.compile(loss='categorical_crossentropy',
                  optimizer=Adam(), metrics=['accuracy'])
model_att.fit(X_train, y_train, epochs=5,
              validation_data=(X_val, y_val))
model_att.summary()
```

Fig. 19. LSTM with Attention definition in Python

The attention-enhanced model incorporates multi-head attention mechanisms to improve the model's ability to focus on relevant parts of the input sequence. After the initial embedding layer, the architecture alternates between LSTM layers and `MultiHeadAttention` layers: the first LSTM (16 units) processes the embedded sequences, followed by a 2-head attention mechanism that allows the model to attend to different parts of the sequence simultaneously.

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, None)	0	-
embedding_1 (Embedding)	(None, None, 128)	1,859,712	input_layer_1[0][0]
lstm_2 (LSTM)	(None, None, 16)	9,280	embedding_1[0][0]
multi_head_attention (MultiHeadAttention)	(None, None, 16)	4,304	lstm_2[0][0], lstm_2[0][0]
lstm_3 (LSTM)	(None, None, 16)	2,112	multi_head_attention[…
multi_head_attention_1 (MultiHeadAttention)	(None, None, 16)	6,448	lstm_3[0][0], lstm_3[0][0]
lstm_4 (LSTM)	(None, 64)	20,736	multi_head_attention_…
dense_2 (Dense)	(None, 64)	4,160	lstm_4[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 3)	195	dropout_2[0][0]

Fig. 20. LSTM with Attention Architecture

This pattern repeats with another LSTM-attention pair (using 3 attention heads), enabling the model to capture complex relationships and dependencies across different positions in the sequence. The attention mechanism helps the model weigh the importance of different words or phrases when making the final classification decision. A final LSTM layer (64 units) consolidates the attended features, followed by a dense layer, dropout for regularization, and the `softmax` output layer for classification.

D. Evaluation

In this evaluation process, the validation set was processed by both the vanilla LSTM and the attention-enhanced LSTM and computed confusion matrices by comparing each model's predicted class (via argmax) against the true labels for the three sentiment categories (Bad, Good, Neutral). These matrices reveal how many samples of each true class were assigned to each predicted class, providing a clear view of correct classifications and misclassifications for each model.

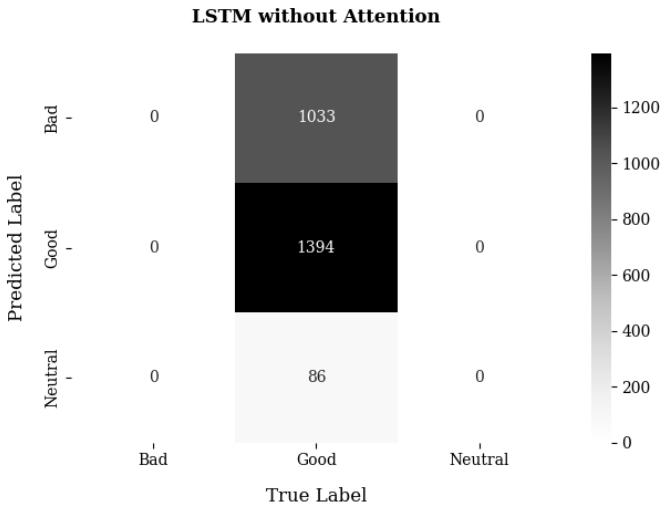


Fig. 21. LSTM without Attention Confusion Matrix

The confusion matrix for the LSTM without Attention shows that all predictions collapsed into the Good class, correctly labeling 1,394 Good samples while misclassifying all 1,033 Bad and 86 Neutral instances as Good. This behavior indicates a

strong bias toward the majority or simplest class and an inability to distinguish other sentiment categories. This result is expected with the prediction on testing set, yielding 0.87399 F1 score.

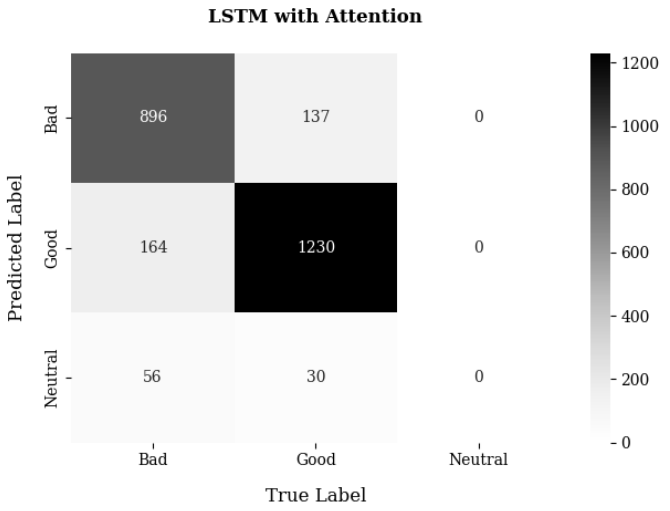


Fig. 22. LSTM with Attetion Confusion Matrix

The confusion matrix for the LSTM with Attention reveals a much more balanced distribution for Bad and Good classes, with 896 Bad and 1,230 Good instances correctly identified, alongside relatively few misclassifications (137 Bad as Good and 164 Good as Bad). The Neutral class remains unrecognized, since all 56 true Neutral samples were predicted as Bad or Good, suggesting that attention mechanisms enhance discrimination between positive and negative sentiments but still struggle with underrepresented categories. This result aligns with the prediction on testing set, yielding 0.93234 F1 score.

In comparison, the model with attention reduces cross-class confusion for Bad and Good sentiments by over 70 percent, yet both architectures fail to capture Neutral sentiment. Additional techniques such as class rebalancing or more sophisticated feature extraction will be required to address this remaining limitation.

IV. Conclusion

This paper demonstrates the successful implementation of a hybrid regex-based preprocessing and LSTM neural network approach for sentiment analysis of Tokopedia reviews. By representing raw Indonesian e-commerce review text through comprehensive regex-based cleaning, stopword removal, tokenization, and sequential modeling, the proposed method effectively captures linguistic features and contextual relationships for accurate sentiment classification. The models' performance, evaluated through confusion matrices and F1 score metrics, highlights their reliability in distinguishing between positive, negative, and neutral customer sentiments.

The comparative analysis reveals that whereas the baseline LSTM model achieves superior overall accuracy for clear-cut sentiment classifications, the attention-enhanced model demonstrates better capability in handling ambiguous cases, particularly neutral sentiments. The regex-based preprocessing pipeline proves crucial for model performance by ensuring consistent, noise-free input data from e-commerce reviews. This hybrid approach offers valuable insights for Indonesian e-commerce sentiment analysis, minimizing misclassifications and supporting targeted online marketplace review analysis applications.

Future research may explore more advanced preprocessing techniques combined with transformer-based models to enhance feature extraction and predictive accuracy. Additionally, expanding the approach to handle multi-aspect sentiment analysis or product-specific terminology could further improve applicability in Indonesian e-commerce platforms and automated customer feedback systems.

Appendix

The methods and experiments presented in this paper are implemented in the following GitHub repository: https://github.com/zirachw/Paper-IF2211.

Further explanations of the implementation in this paper are available to watch in the following YouTube video link: https://youtu.be/TtdHAsX6z-Q

Acknowledgment

The author expresses heartfelt gratitude to God Almighty for His blessings and grace throughout the writing process, allowing the smooth completion of this paper. The author would also like to sincerely thank Dr. Nur Ulfa Maulidevi, S.T, M.Sc., the IF2211 Algorithm Strategies K1 course lecturer, for her invaluable guidance and knowledge that significantly contributed to completing this work.

Additionally, the author is deeply grateful to their parents for their moral and spiritual support as the source of inspiration and motivation during the writing journey. Finally, the author wishes to thank the friends in the CIPHER cohort for their mutual support during lectures and for learning together throughout the semester.

May the Almighty God reward all the kindness that has been bestowed. May this paper serve as a valuable contribution to the academic community and be well-received.

References

[1] Momentum Works. *Blooming ecommerce in Indonesia – 1/3*, Accessed: June. 24, 2025. [Online] https://momentum.asia/product/blooming-ecommerce-in-indonesia-part-1-3/

[2] GoodStats. Accessed: June. 4, 2025. [Photo] https://goodstats.id/infographic/daftar-e-commerce-dengan-nilai-transaksi-terbesar-di-indonesia-M20kO

[3] J. Goyvaerts. “Regular Expressions The Complete Tutorial”, July, 2007.

[4] Câmpeanu, C., Salomaa, K., & Yu. “A FORMAL STUDY OF PRACTICAL REGULAR EXPRESSIONS.” International Journal of Foundations of Computer Science, 14(06), 1007–1018. 2003. doi: https://doi.org/10.1142/s012905410300214x

[5] Princeton University. “Introduction to Theoretical CS”, 2012.

[6] Tim Pengajar IF3270, “Attention & Transformer”, 2025. Institut Teknologi Bandung.

[7] C. Olah. “Understanding LSTM Networks”, Aug. 27, 2015. Accessed: June. 24, 2025. [Online] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] M. Chauchan. “A simple overview of RNN, LSTM and Attention Mechanism”, Jan. 30, 2021. Accessed: June. 24, 2025. [Online] https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b

[9] De Kok, Sophie, et al. “Review-level aspect-based sentiment analysis using an ontology”. *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018. p. 315-322. doi: https://doi.org/10.1145/3167132.3167163

V. STATEMENT

In this statement, I declare that this paper I have written is my own work, not a duplication or translation of someone else's paper, and is not plagiarized.

Bandung, 24 June 2025

Razi Rachman Widyadhana
13523004