JURNAL SAINTEK DOI: 10.33830/saintek.v2i2.15475.2026
Volume 02 Nomor 02, Maret 2026
Halaman: 313 - 328

PERINGKASAN TEKS ABSTRAKTIF OTOMATIS MENGGUNAKAN MODEL SEQ2SEQ DENGAN LSTM *ENCODER-DECODER* DAN *ATTANTION* PADA DATA SET *AMAZON FINE FOOD REVIEWS*

Juliasmita Eka Rahayu, Dyah Paminta Rahayu, Sitta Alief Farihati*
Program Studi Matematika, Universitas Terbuka, Tangerang Selatan, Indonesia
**Penulis korespondensi: sitta@ecampus.ut.ac.id*

ABSTRAK

Penelitian ini bertujuan untuk membangun model peringkasan teks otomatis dengan strategi abstraktif. Model dibangun menggunakan pendekatan *sequence-to-sequence* (seq2seq) dengan arsitektur *encoder* dan *decoder* berbasis *Long Short-Term Memory* (LSTM) yang efektif untuk menangani data teks yang memerlukan proses mengingat informasi penting dari awal urutan dan menghubungkannya pada bagian akhir urutan. Kemudian ditambahkan mekanisme attention untuk membantu *decoder* fokus pada bagian-bagian tertentu yang memerlukan perhatian. Model dilatih pada pasangan data teks ulasan dan teks ringkasan pada data set *Amazon Fine Food Reviews* kemudian dilatih menggunakan fungsi kerugian (*loss*) *categorical crossentropy* dan algoritma optimasi RMSProp. Dari pelatihan diperoleh akurasi sebesar 67.2% dan kerugian sebesar 1.6314. Metode evaluasi dilakukan dengan menggunakan metrik BLEU (*Bilingual Evaluation Understudy*) untuk membandingkan hasil prediksi keluaran model dengan referensi manusia. Dari hasil evaluasi, model mampu menghasilkan ringkasan dengan skor komulatif BLEU-1 sebesar 0.428571, BLEU-2 sebesar 0.654654, BLEU-3 sebesar 0.756080, dan BLEU-4 sebesar 0.809107. Hasil metrik BLEU menunjukkan model dapat menghasilkan prediksi ringkasan yang cukup koheren dan relevan.

Kata kunci: *automatic* teks *summarization*, *bleu score*, *long short-term memory*, *natural language processing*, *sequence-to-sequence*

1 PENDAHULUAN

Berkembangnya era teknologi mendorong keberagaman aktivitas digital masyarakat. Mulai dari bertukar informasi, transaksi jual-beli, hingga aktivitas konsumsi layanan pada saat ini dapat dilakukan secara daring melalui sebuah *platform* digital. Untuk dapat bertahan dalam persaingan, *platform* digital seperti media sosial, *e-commerce*, hingga *platform* layanan memerlukan sebuah kolom ulasan sebagai bentuk penilaian terhadap suatu produk atau layanan yang disediakan. Menurut Febriana *et al.* (2023), ulasan dapat dijadikan sebagai alasan konsumen untuk lebih meyakini bahwa produk yang ditawarkan sesuai dengan yang diharapkan konsumen, baik ulasan positif atau ulasan negatif karena ulasan tersebut benar adanya diberikan oleh konsumen lain yang sudah berbelanja. Dengan demikian, ulasan dapat dijadikan sebagai sumber informasi dalam mengambil keputusan dan alat ukur untuk menilai apakah suatu produk layak atau memerlukan perbaikan.

Ulasan dengan teks yang panjang terkadang membuat produsen merasa bingung untuk membacanya. Oleh karenanya, dibuat sebuah alat peringkasan yang disebut dengan peringkasan teks otomatis atau *Automatic Text Summarzation* (ATS). *Automatic text summarization* merupakan salah satu teknik dari *Neural Language Processing* (NLP) yang merupakan proses otomatis untuk menghasilkan ringkasan dari teks panjang namun tidak menghilangkan makna utama teks tersebut. Tujuan dari *automatic text summarization* adalah

untuk membuat ringkasan yang singkat, mudah dipahami kalimatnya dan memiliki informasi penting yang ingin disampaikan dari teks original tersebut (Setyawan *et al.*, 2021).

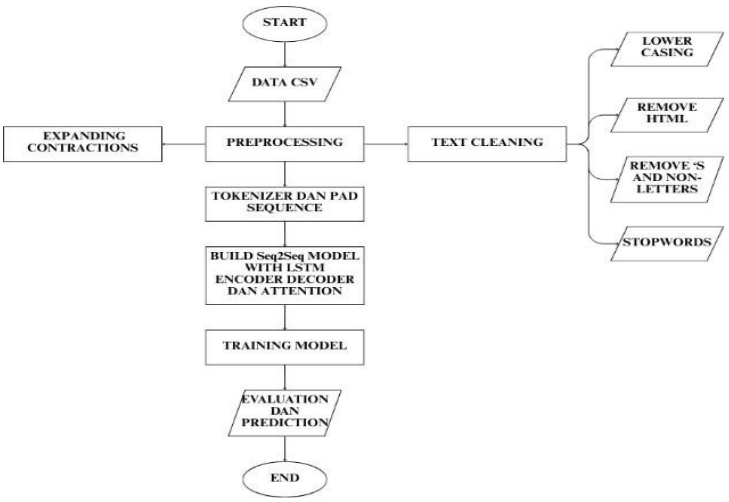
Automatic Text Summarzation memiliki dua strategi utama yaitu *extractive* yang menyusun ringkasan dengan mengambil kalimat penting dalam sumber dan *abstractive* yang menghasilkan ringkasan berupa teks baru berdasarkan pemahaman dalam sumber namun tetap pada intinya (Luo *et al.*, 2024). Dari keduanya, strategi pendekatan *abstractive* menjadi lebih populer dibandingkan dengan *extractive,* karena pendekatan *abstractive* dapat menghasilkan sebuah ringkasan yang lebih alami layaknya hasil ringkasan manusia (Shakil *et al.*, 2024), namun dalam pengimplementasiannya membutuhkan teknik dan arsitektur yang lebih kompleks.

Salah satu metode yang digunakan dalam peringkasan teks abstraktif otomatis adalah model *Sequence-to-Sequence* (Seq2Seq) dengan menggunakan model *Recurrent Neural Networks* (RNN), khususnya arsitektur *Long Short-Term Memory* (LSTM). LSTM sangat efektif untuk menangani data teks yang memerlukan proses mengingat informasi penting dari awal urutan dan menghubungkannya pada bagian akhir urutan. Arsitektur ini dipandang lebih akurat menangani urutan temporal dan dependensi jarak jauh daripada RNN konvensional (Paaß & Giesselbach, 2023). Namun, sering kali LSTM kesulitan untuk mempertahankan fokus bagian-bagian penting saat panjang teks meningkat. Oleh karena itu, *attantion* ditambahkan untuk membantu memilih informasi yang paling berguna selama proses *encoding* dan *decoding*.

Meskipun pendekatan dengan LSTM *encoder-decoder* dan *attention* cukup efektif untuk bidang peringkasan teks otomatis, namun tantangan tetap muncul ketika model digunakan pada data yang tidak berstruktur, memiliki panjang yang bervariasi, gaya bahasa yang informal, dan menggunakan kata-kata yang hiperbolis. Pada artikel ini, akan dibahas mengenai pemanfaatan arsitektur LSTM dengan *attention* dalam sistem *Automaic Teks Summarization* untuk merangkum sekumpulan ulasan produk makanan dalam bahasa Inggris.

2 METODE

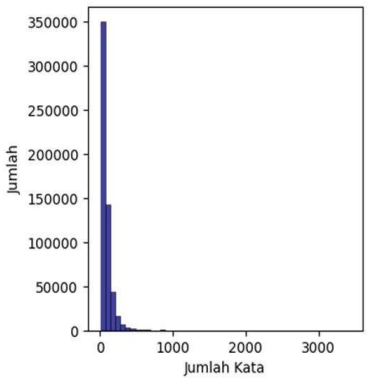
Penelitian ini menggunakan salah satu teknik dalam *neural language processing* (NLP) dengan pendekatan Seq2Seq. Model ini terdiri dari dua komponen yaitu *encoder* dan *decoder*. Dua komponen ini dibangun dengan jaringan *Long Short-Term Memory* (LSTM). Kemudian ditambahkan mekanisme *attention* untuk memfokuskan *decoder* pada bagian-bagian penting dalam masukan saat menghasilkan ringkasan. Untuk mencapai tujuan penelitian, diperlukan tahapan-tahapan seperti *data collecting*, *exploratory data analysis, preprocessing, tokenization, padding, modeling, training, evaluation,* dan *prediction.*



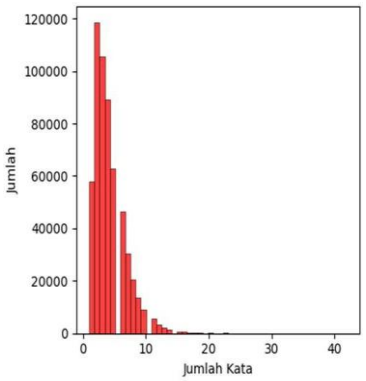
Gambar 1. *Flowchart* motodologi penelitian

Data set yang digunakan adalah kumpulan *review* makanan pada *platform* Amazon yang diunduh melalui Kaggle dengan nama *Amazon fine food review*. Data set ini berisi 568.454 ulasan dari 256.059 pengguna terhadap 74.258 produk yang dikumpulkan dalam periode Oktober 1999 sampai Oktober 2012. Data set ini tergolong data lama, namun hal ini tidak menjadi kendala, karena fokus penelitian ini pada arsitektur model untuk menghasilkan ringkasan.

Data set *Amazon fine food review* dipilih sebagai data untuk melatih model karena didasarkan pada distribusi panjang katanya yang tergolong pendek, sehingga pada saat pelatihan model lebih efisien dan cepat secara komputasi. Sebagian besar isi data set *Amazon fine food review* adalah ulasan yang terdiri dari jumlah kata di bawah seratus hingga dua ratus kata. Namun, ada juga sedikit ulasan dengan panjang di atas seribu kata. Selain itu ada beberapa kata yang dominan dalam dataset, seperti: "gluten *free*", "*dog food*", "*cat food*", "*grocery store*", "*highly recommend*", "*much better*", "*good*", "*great product*".



Gambar 2. Distribusi jumlah kata dalam ulasan



Gambar 3. Distribusi jumlah kata dalam ringkasan

Gambar 4. *WorldCould* kata yang sering muncul

Ulasan ditulis pengguna menggunakan bahasa Inggris, sehingga memudahkan proses *preprocessing* dengan bantuan pustaka NLTK. Selain itu, ulasan ini bergaya bebas dan bervariasi dengan panjang yang tidak terstruktur secara eksplisit, sehingga sesuai dengan penelitian dalam bidang peringkasan teks otomatis khususnya teknik abstraktif. Data set diunduh dengan format CSV yang diolah dengan menggunakan bahasa pemrograman Python.

3 HASIL DAN PEMBAHASAN

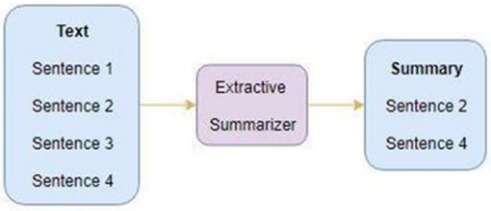
3.1 Peringkasan Teks Otomatis

Peringkasan teks otomatis merupakan sebuah proses penting dalam pemrosesan bahasa alami yang bertujuan untuk mengubah sejumlah besar data teks menjadi lebih ringkas, jelas dan mudah dipahami (Supriyono *et al.*, 2024). Dengan adanya peringkasan teks otomatis, pemrosesan informasi dalam jumlah besar lebih cepat dan efisien dibandingkan dengan sebelumnya, karena proses peringkasan yang sebelumnya memerlukan tenaga manual kini dapat diganti dengan yang otomatis.

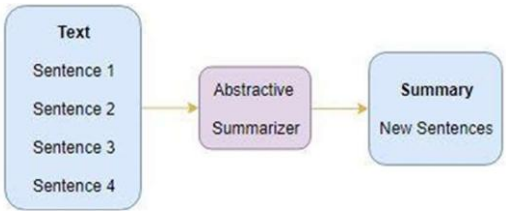
3.2 Strategi Peringkasan Teks Otomatis

Peringkasan teks otomatis memiliki dua strategi, yaitu ekstraktif dan abstraktif. Peringkasan teks dengan strategi ekstraktif adalah memilih kalimat penting dari dalam sumber kemudian disusun jadi satu menjadi ringkasan, kalimat yang dipilih tetap dalam bentuk aslinya tanpa modifikasi apa pun (Tsai *et al.*, 2020). Strategi ini digunakan untuk memilih bagian teks dari sumber yang paling mencolok tanpa perubahan yang besar pada teks atau kalimat asli dalam sumber.

Berbeda dengan abstraktif yang akan menghasilkan ringkasan berupa teks baru berdasarkan pemahaman dalam sumber namun tetap pada intinya (Luo *et al.*, 2024). Dengan demikian, strategi ini melakukan penyusunan ulang suatu teks dengan menggunakan kalimat baru yang tidak ada dalam sumber namun berdasarkan pemahaman ada inti sumber. Perbedaan keduanya dapat dilihat pada Gambar 5 dan 6.



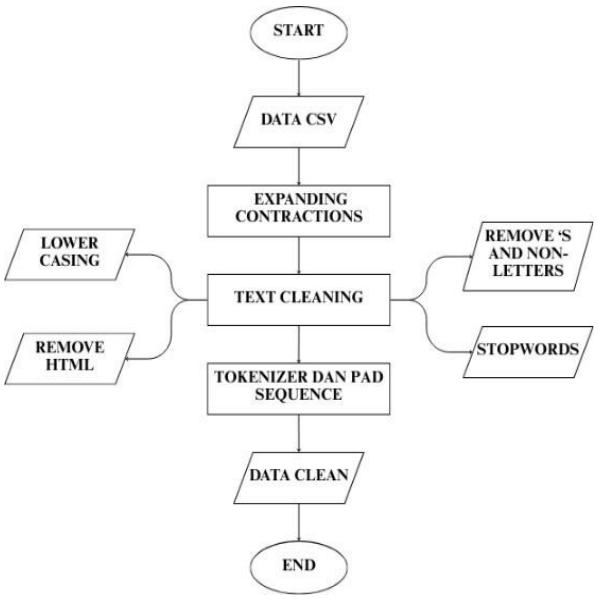
Gambar 5. Strategi Ekstraktif (diadaptasi dari Akhter & Mehra (2022))



Gambar 6. Strategi Abstraktif (diadaptasi dari Akhter & Mehra (2022))

3.3 Pre-processing

Pembuatan peringkasan teks otomatis dimulai dengan tahapan *preprocessing.* Menurut Muflikhah *et al.* (2023), tujuan *preprocessing* adalah untuk meningkatkan kualitas data dengan mengeliminasi masalah-masalah yang ada pada data tersebut, sehingga *preprocessing* menjadi langkah utama yang sangat penting. Dengan *preprocessing* yang baik dan benar, akurasi model dapat meningkat karena data masukan menjadi lebih bersih, konsisten, dan bebas dari *noise*.



Gambar 7. *Flowchart preprocessing*

Dalam penelitian ini, *preprocessing* yang diperlukan adalah *expanding contractions* untuk mengubah bentuk kata singkatan menjadi bentuk lengkapnya, seperti *"ain't"* menjadi "*is not*", *"aren't"* menjadi "*are not",* dan *"can't"* menjadi *"cannot",* terlihat pada gambar 8.

Selanjutnya adalah tahap *cleaning text* terlihat pada Gambar 9. Tahapan ini terdiri dari *setting stopwords* dalam bahasa inggris dari NLTK, *lowercase* untuk mengubah huruf besar menjadi huruf kecil supaya tidak ada duplikasi, contohnya seperti "*Jumbo*" menjadi "*jumbo*" dan "*Salted*" menjadi "*salted*". Kemudian menghapus *tag* HTML, menghapus tanda baca dan menghapus kata yang pendek. Setelah tahapan *expanding contractions* dan *cleaning text* dijalankan, data yang awalnya berisi *noise* seperti *tag* HTML dan tanda baca menjadi lebih bersih, terstruktur dan bebas dari *noise* yang dapat mengganggu pemodelan.

Gambar 8. Potongan skrip tahapan *expanding contractions*

Gambar 9. Potongan skrip tahapan *cleaning text*

Tabel 1. Perbedaan sebelum *preprocessing* dan setelah *preprocessing*

Sebelum *preprocessing*	Setelah *preprocessing*
Teks: *Product arrived labeled as Jumbo Salted Peanuts...the peanuts were actually small sized unsalted. Not sure if this was an error or if the vendor intended to represent the product as "Jumbo".*	Teks: *product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended represent product jumbo*
Ringkasan: *Not as Advertised*	Ringkasan: *not as advertised*
Teks: *Great taffy at a great price. There was a wide assortment of yummy taffy. Delivery was very quick. If your a taffy lover, this is a deal.*	Teks: *great taffy great price wide assortment yummy taffy delivery quick taffy lover deal*
Ringkasan: *Great taffy*	Ringkasan: *great taffy*

Selanjutnya data yang telah bersih dilakukan analisis distribusi panjang kata yang dijadikan sebagai landasan untuk menentukan *hyperparameter* batas maksimum panjang teks. Batas maksimum ini digunakan sebagai tetapan untuk memastikan konsistensi panjang masukkan dan keluaran pada saat pelatihan model. Selain itu, batas maksimum juga bertujuan untuk mengurangi beban komputasi dan mempercepat pelatihan model. Setelah dilakukan beberapa percobaan, diperoleh cakupan 63% dari data ulasan dengan panjang teks lebih dari sama dengan 35 kata, dan cakupan 94% dari data ringkasan dengan panjang teks lebih dari sama dengan 8 kata. Untuk membantu model mengetahui batas kalimat saat pelatihan sehingga

decoding dapat menghasilkan ringkasan yang baik, maka ditambahkan *token* khusus di awalan dan di akhiran kata, yaitu *sostok* sebagai awalan (*start token*) dan *eostok* sebagai akhiran (*end token*).

```
#The Maximum Review length
count=0
for i in reviewsData['Cleaned_Text']:
    if(len(i.split())<=35):
        count=count+1
print(count/len(reviewsData['Cleaned_Text']))
```

```
0.630776211084741
```

Gambar 10. Analisis panjang teks ulasan

```
#The Maximum Summary length
count=0
for i in reviewsData['Cleaned_Summary']:
    if(len(i.split())<=8):
        count=count+1
print(count/len(reviewsData['Cleaned_Summary']))
```

```
0.9437386569872959
```

Gambar 11. Analisis panjang teks ringkasan

```
#Adding START and END
cleaned_text =np.array(reviewsData['Cleaned_Text'])
cleaned_summary=np.array(reviewsData['Cleaned_Summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len
       and len(cleaned_text[i].split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

Gambar 12. Potongan skrip `sostok` dan `eostok`

3.4 Tokenization dan *Padding*

Nambiar *et al.* (2021) mengatakan bahwa *tokenization* adalah proses memecah sebuah kalimat menjadi bagian-bagian kecil yang disebut token atau kata. Dengan begitu, *tokenization* merupakan alat untuk menyederhanakan struktur kalimat dengan cara memecah kalimat menjadi bagian-bagian yang lebih kecil. *Tokenization* membantu mengurangi kompleksitas kalimat dengan memisahkan kalimat menjadi unit-unit yang lebih mudah diproses.

Dalam penelitian ini, proses *tokenization* menggunakan "Tokenizer" dari pustaka yang secara langsung dapat membangun *vocabulary* berdasarkan frekuensi kemunculan kata dalam data pelatihan. Jenis yang digunakan adalah *tokenization* kata dengan tujuan memecah teks menjadi potongan kata. Misalkan suatu teks adalah T=[w_1, w_2,…, w_n], maka hasil tokenisasinya adalah sebagai berikut.

$$Token\ (T) = [i_1, i_2, \dots, i_n]\ dengan\ i_j\ \in \mathbb{N} \quad (1)$$

```
#Preparing a Tokenizer for reviews on training data
X_tokenizer = Tokenizer()
X_tokenizer.fit_on_texts(list(X_train))

#Preparing a Tokenizer for summaries on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))
```

Gambar 13. Potongan skrip persiapan *tokenization*

Selanjutnya, dilakukan analisis terhadap kata unik yang jarang muncul *(rare words)* dari ambang batas tertentu *(threshold)*. Dalam studinya, Bystrov *et al.* (2023) mengemukakan bahwa menghapus kata-kata unik yang jarang muncul dapat mempercepat proses komputasi saat pelatihan model. Dalam penelitian ini, diambil *threshold* = 4 untuk teks ulasan dan *threshold* = 6 untuk teks ringkasan. Ini memiliki arti bahwa kata dalam korpus yang jarang

muncul akan dipertahankan apabila muncul minimal sebanyak 4 kali pada teks ulasan, dan 6 kali pada teks ringkasan. Dengan *threshold* sebesar 4 diperoleh pengurangan sebanyak 65,61% kata unik yang jarang muncul namun hanya mengurangi sekitar 4,94% dari teks ulasan. Sedangkan pada teks ringkasan, dengan *threshold* sebesar 6 diperoleh pengurangan sebanyak 80,80% kata unik yang jarang muncul namun hanya mengurangi sekitar 9,10% dari teks ringkasan.

```
#The Maximum Review length
count=0
for i in reviewsData['Cleaned_Text']:
    if(len(i.split())<=35):
        count=count+1
print(count/len(reviewsData['Cleaned_Text']))
```

```
0.630776211084741
```

Gambar 14. Potongan skrip *rare words* pada teks ulasan

```
#The Maximum Summary length
count=0
for i in reviewsData['Cleaned_Summary']:
    if(len(i.split())<=8):
        count=count+1
print(count/len(reviewsData['Cleaned_Summary']))
```

```
0.9437386569872959
```

Gambar 15. Potongan skrip *rare words* pada teks ringkasan

Setelah dilakukan tokenisasi, diperoleh panjang *sequence* yang berbeda-beda. Maka diperlukan *padding* untuk menyamakan panjang masukan sebelum diproses oleh model. *Padding* dilakukan dengan menambahkan token khusus seperti nol di awalan yang disebut *pre-padding* dan di akhiran yang disebut *post-padding.* Hal ini dapat digambarkan dengan:

$$X_{pad}, Y_{pad} = \begin{cases} [0,0\,, \dots, 0, i_1, i_2, \dots, i_n] & jika\ n < L \\ [i_1, i_2, \dots, i_L] & jika\ n \geq L \end{cases} \quad (2)$$

dengan:
0 adalah *padding* yang disisipkan apabila $n < L$
$i_1, i_2, \dots, i_n$ adalah token hasil dari *tokenization*
L adalah panjang maksimum *sequence.*

```
#Tokenizer for reviews
#Preparing a Tokenizer for reviews on training data
X_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
X_tokenizer.fit_on_texts(list(X_train))

#Converting text sequences into integer sequences
X_train_seq = X_tokenizer.texts_to_sequences(X_train)
X_test_seq = X_tokenizer.texts_to_sequences(X_test)

#Padding zero upto maximum length
X_train = pad_sequences(X_train_seq,
                        maxlen = max_text_len,
                        padding = 'post')
X_test = pad_sequences(X_test_seq,
                       maxlen = max_text_len,
                       padding = 'post')

#Size of vocabulary (+1 for padding token)
X_voc   =  X_tokenizer.num_words + 1
```

Gambar 14. Potongan skrip *tokenization* pada data ulasan

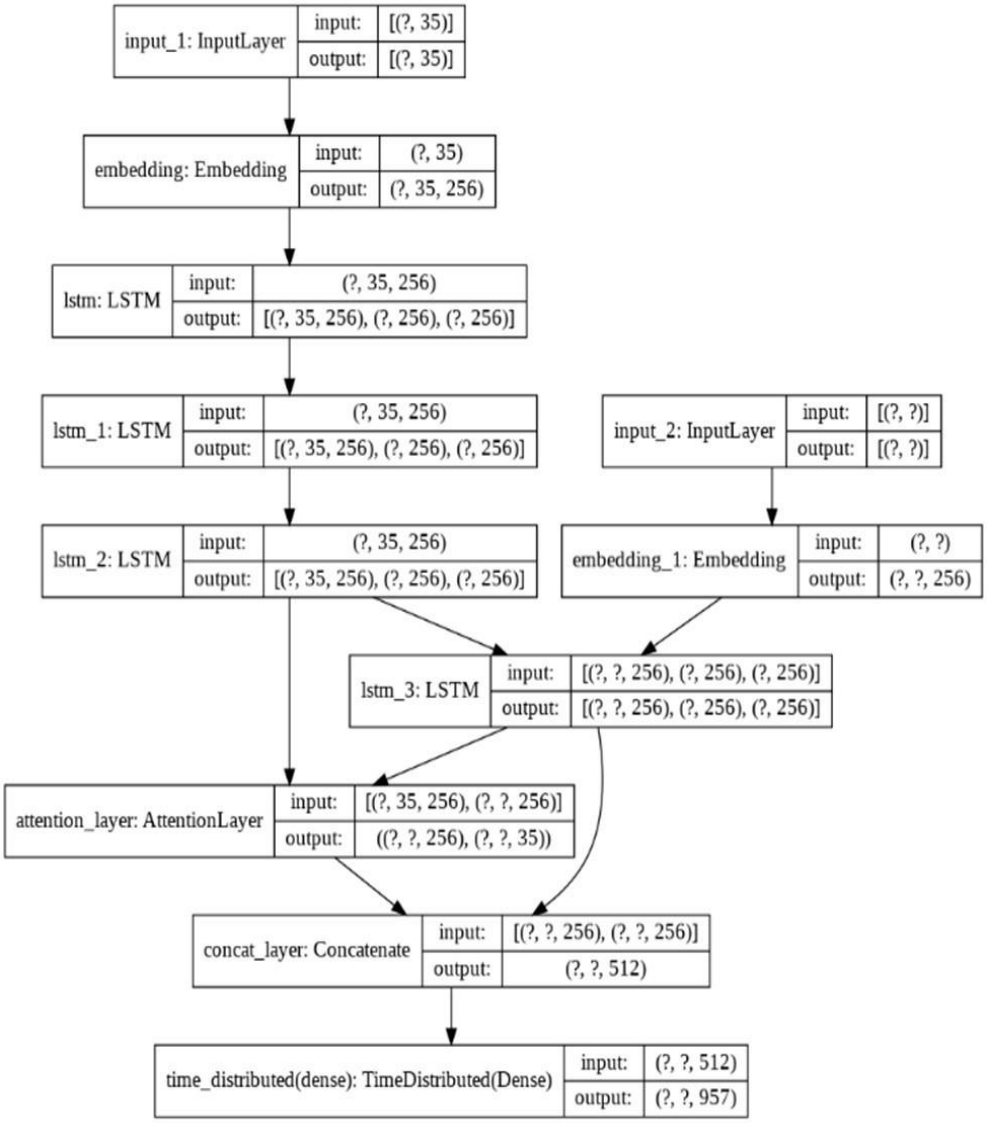
```
#Tokenizer in summary
#Preparing a tokenizer for summaries on training data
y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
y_tokenizer.fit_on_texts(list(y_train))

#Converting text sequences into integer sequences
y_train_seq = y_tokenizer.texts_to_sequences(y_train)
y_test_seq = y_tokenizer.texts_to_sequences(y_test)

#Padding zero upto maximum length
y_train = pad_sequences(y_train_seq,
                        maxlen=max_summary_len,
                        padding='post')
y_test = pad_sequences(y_test_seq,
                       maxlen=max_summary_len,
                       padding='post')

#size of vocabulary
y_voc = y_tokenizer.num_words +1
```

Gambar 15. Potongan skrip *tokenization* pada data ringkasan

3.1 Membangun model

Dalam penelitian ini, model dibangun dengan Seq2Seq model yang terdiri dari dua bagian utama yaitu *Encoder* dan *Decoder* yang dibangun dengan jaringan LSTM serta ditambahkan mekanisme *attention*. Menurut Niu *et al.* (2021), model *Encoder-Decoder* adalah suatu kerangka berbasis jaringan saraf yang dirancang untuk menangani pemetaan antara masukan dan keluaran yang memiliki struktur kompleks. Dalam bukunya, Paaß & Giesselbach

(2023) menyatakan bahwa tantangan utama dalam Seq2Seq model adalah *decoder* yang harus memperhatikan penyematan token *encoder.* Dengan begitu, *Attention* ditambahkan untuk mengatasi permasalahan dalam pemrosesan teks yang memerlukan perhatian pada bagian-bagian tertentu. Arsitektur yang digunakan dibangun oleh beberapa komponen, yaitu *embedding layer*, *encoder* LSTM bertingkat, *decoder* LSTM, lapisan *attention* oleh Bahdanau, dan *TimeDistributed Dense layer* sebagai keluaran.



Gambar 18. Arsitektur model

Model diawali dengan dua *InputLayer* untuk *encoder* dan *decoder*. *Encoder* berfungsi untuk mengubah kalimat masukan dan mempertahankan konteks dan makna aslinya dengan memanfaatkan lapisan LSTM yang menyimpan informasi dalam bentuk *hidden state* (h_t) dan *cell state* (C_t) sebagai umpan balik internal (Chaurasia *et al.*, 2023). Dalam penelitian ini, e*ncoder* menerima masukan yang berbentuk urutan token dengan panjang 35 token dan mengubahnya menjadi vektor menggunakan *embedding layer* yang berukuran (35, 256). Vektor ini kemudian diteruskan ke tiga buah lapisan LSTM bertingkat (lstm, lstm_1, lstm_2) sebagai *encoder*. Dalam bukunya, Yudistira *et al.* (2023) merumuskan bahwa model LSTM bekerja pada gerbang-gerbang dengan proses di setiap waktu didefinisikan sebagai:

$$
\begin{aligned}
f_t &= \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_i \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t \odot C_t + i_t \tilde{C}_t \\
o_t &= \sigma\,(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= ot \odot tanh(C_t)
\end{aligned}
\tag{3}
$$

dengan:

f_t : *forget gate*
i_t : *input gate*
$\tilde{C}_t$: *candidate cell state*
C_t : *new cell state*
o_t : *output gate*
h_t : *hidden state*
σ : fungsi aktivasi sigmoid
Tanh : fungsi aktivasi hiperbolik
$\odot$: perkalian antar elemen
x_i : masukan saat ini
W dan b: bobot dan bias

Dalam penelitian ini, untuk *encoder* menggunakan *unidirectional* LSTM yang mana model menggunakan 3 LSTM bertingkat satu arah, sehingga keluaran dari setiap layer akan menjadi masukan untuk layer berikutnya. Dengan mengambil formula bentuk *hidden state* (h_t) dan *cell state* (C_t) dari buku Yudistira *et al.* (2023), maka secara matematis untuk layer pertama (lstm) dirumuskan dengan:

$$
{h_t}^{(1)}, {c_t}^{(1)} = LSTM_1\ \left({h_{t-1}}^{(1)}, {c_{t-1}}^{(1)}\right) \tag{4}
$$

dan untuk layer ke dua (lstm_1) dirumuskan dengan:

$$
{h_t}^{(2)}, {c_t}^{(2)} = LSTM_2\ \left({h_t}^{(1)}, {h_{t-1}}^{(2)}, {c_{t-1}}^{(2)}\right) \tag{5}
$$

sedangkan untuk layer ketiga (lstm_2) yang merupakan keluaran akhir *encoder* dirumuskan dengan:

$$
{h_t}^{(3)}, {c_t}^{(3)} = LSTM_2\ \left({h_t}^{(2)}, {h_{t-1}}^{(3)}, {c_{t-1}}^{(3)}\right). \tag{6}
$$

Selanjutnya, informasi dari masukan yang telah disaring oleh *encoder* diteruskan ke *decoder* LSTM (lstm_3). *Decoder* bertugas memprediksi kalimat keluaran secara bertahap kata demi kata (Chaurasia *et al.*, 2023). *Decoder* menerima urutan target yang kemudian diubah menjadi vektor-vektor melalui lapisan *embedding* yang sama dengan *encoder*. Dengan kata lain *decoder* tidak mulai dari nol, melainkan menggunakan *initial state* dari *encoder* layer ketiga (lstm_2) dengan $s_0 = h_{Tx}^{(3)}$, $c_0 = c_{Tx}^{(3)}$. Dengan demikian, saat *decoder* pada waktu t dalam Matematika dirumuskan dengan:

$$
s_t, c_t = LSTM(y_{t-1}, s_{t-1}, c_{t-1}) \tag{7}
$$

Karena model untuk peringkasan teks otomatis yang dibangun dengan Seq2Seq model tergolong lemah, maka diperlukannya *attention*. Dalam hal ini, konsep *attention* memungkinkan model neural network untuk melihat kembali bagian-bagian input yang relevan saat menghasilkan setiap token output, sehingga meningkatkan akurasi penerjemahan maupun ringkasan (Bahdanau *et al.*, 2014). Untuk meningkatkan akurasi pemodelan, ditambahkan mekanisme *attention* sesuai dengan penelitian Bahdanau *et al.* (2014).

Dalam mekanisme *attention* ini, keluaran encoder (hi) diberikan bobot berdasarkan seberapa relevan status tersembunyi terakhir *decoder* (s_{t-1}) kemudian digabungkan menjadi vektor (Bahdanau *et al.*, 2014). Untuk menentukan kata keluaran pada langkah *t*, sistem menghitung *attention weights* (α_{ti}) untuk setiap keluaran *encoder* (h_i) berdasarkan seberapa *score* kecocokan (e_{ti}) vektor h_i dengan status tersembunyi *decoder* s_{t-1}. Dalam penelitian Bahdanau *et al.*, (2014), *score* kecocokan secara matematis ditulis:

$$e_{ti} = score(s_{t-1}, h_i) = v_a^T \tanh W_a s_{t-1} + U_a h_i \quad (8)$$

dengan W_a, U_a, dan v_a adalah bobot pelatihan.

Attention weights dalam Bahdanau *et al.* (2014) dirumuskan dengan:

$$a_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{T} \exp(e_{tk})} \quad (9)$$

di mana rumus ini menggunakan fungsi *softmax* yang dapat mengubah skor e_{ti} menjadi nilai probabilitas. Adapun konteks vektor yang digunakan oleh *decoder* adalah:

$$c_t = \sum_{i=1}^{T} \alpha_{ti} h_i \quad (10)$$

Konteks vektor c_t tersebut adalah hasil gabungan dari semua keluaran dari *encoder* yang telah diberikan bobot α_{ti}. Hal ini menandakan seberapa besar perhatian yang diberikan model terhadap setiap elemen dalam masukan saat menghasilkan keluaran pada waktu ke-t sehingga c_t bertugas sebagai pembantu *decoder* untuk menghasilkan keluaran (Bahdanau *et al.*, 2014).

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 35)]	0	
embedding (Embedding)	(None, 35, 256)	1312768	input_1[0][0]
lstm (LSTM)	[(None, 35, 256), (N	525312	embedding[0][0]
input_2 (InputLayer)	[(None, None)]	0	
lstm_1 (LSTM)	[(None, 35, 256), (N	525312	lstm[0][0]
embedding_1 (Embedding)	(None, None, 256)	244992	input_2[0][0]
lstm_2 (LSTM)	[(None, 35, 256), (N	525312	lstm_1[0][0]
lstm_3 (LSTM)	[(None, None, 256),	525312	embedding_1[0][0] lstm_2[0][1] lstm_2[0][2]
attention_layer (AttentionLayer	((None, None, 256),	131328	lstm_2[0][0] lstm_3[0][0]
concat_layer (Concatenate)	(None, None, 512)	0	lstm_3[0][0] attention_layer[0][0]
time_distributed (TimeDistribut	(None, None, 957)	490941	concat_layer[0][0]

Total params: 4,281,277
Trainable params: 4,281,277
Non-trainable params: 0

Gambar 19. Ringkasan model

```
latent_dim = 256
embedding_dim = 256

# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(X_voc,embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,
                     return_sequences=True,
                     return_state=True,
                     dropout=0.4,
                     recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,
                     return_sequences=True,
                     return_state=True,
                     dropout=0.4,
                     recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3= LSTM(latent_dim,
                    return_state=True,
                    return_sequences=True,
                    dropout=0.4,
                    recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

#Setting up the Decoder using 'encoder_states' as initial state
decoder_inputs = Input(shape=(None,))

#Embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim,
                    return_sequences=True,
                    return_state=True,
                    dropout=0.4,
                    recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

#Attention layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

#Concating Attention input and Decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])

#Dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

#Defining the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Gambar 20. Potongan skrip *tokenization* pada data ulasan

```
#Adding Custom Attention Layer
class AttentionLayer(Layer):
    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert isinstance(input_shape, list)
        self.W_a = self.add_weight(name='W_a',
                                   shape=tf.TensorShape((input_shape[0][2],
                                                         input_shape[0][2])),
                                   initializer='uniform',
                                   trainable=True)
        self.U_a = self.add_weight(name='U_a',
                                   shape=tf.TensorShape((input_shape[1][2],
                                                         input_shape[0][2])),
                                   initializer='uniform',
                                   trainable=True)
        self.V_a = self.add_weight(name='V_a',
                                   shape=tf.TensorShape((input_shape[0][2], 1)),
                                   initializer='uniform',
                                   trainable=True)

        super(AttentionLayer, self).build(input_shape)

    def call(self, inputs, verbose=False):
        """
        inputs: [encoder_output_sequence, decoder_output_sequence]
        """
        assert type(inputs) == list
        encoder_out_seq, decoder_out_seq = inputs
        if verbose:
            print('encoder_out_seq', encoder_out_seq.shape)
            print('decoder_out_seq', decoder_out_seq.shape)

        def energy_step(inputs, states):
            """ Step function for computing energy for a single decoder state """

            assert_msg = "States must be a list. However states {} is of type {}".format(states, type(states))
            assert isinstance(states, list) or isinstance(states, tuple), assert_msg

            """ Some parameters required for shaping tensors"""
            en_seq_len, en_hidden = encoder_out_seq.shape[1], encoder_out_seq.shape[2]
            de_hidden = inputs.shape[-1]

            """ Computing S.Wa where S=[s0, s1, ..., si]"""
            # <= batch_size*en_seq_len, latent_dim
            reshaped_enc_outputs = K.reshape(encoder_out_seq, (-1, en_hidden))
            # <= batch_size*en_seq_len, latent_dim
            W_a_dot_s = K.reshape(K.dot(reshaped_enc_outputs, self.W_a), (-1, en_seq_len, en_hidden))
            if verbose:
                print('Wa.s', W_a_dot_s.shape)

            """ Computing hj.Ua """
            U_a_dot_h = K.expand_dims(K.dot(inputs, self.U_a), 1)  # <= batch_size, 1, latent_dim
            if verbose:
                print('Ua.h', U_a_dot_h.shape)

            """ tanh(S.Wa + hj.Ua) """
            # <= batch_size*en_seq_len, latent_dim
            reshaped_Ws_plus_Uh = K.tanh(K.reshape(W_a_dot_s + U_a_dot_h, (-1, en_hidden)))
            if verbose:
                print('Ws+Uh', reshaped_Ws_plus_Uh.shape)

            """ softmax(va.tanh(S.Wa + hj.Ua)) """
            # <= batch_size, en_seq_len
            e_i = K.reshape(K.dot(reshaped_Ws_plus_Uh, self.V_a), (-1, en_seq_len))
            # <= batch_size, en_seq_len
            e_i = K.softmax(e_i)

            if verbose:
                print('ei', e_i.shape)

            return e_i, [e_i]

        def context_step(inputs, states):
            """ Step function for computing ci using ei """
            # <= batch_size, hidden_size
            c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1), axis=1)
            if verbose:
                print('ci', c_i.shape)
            return c_i, [c_i]

        def create_initial_state(inputs, hidden_size):
            fake_state = K.zeros_like(inputs)  # <= (batch_size, enc_seq_len, latent_dim
            fake_state = K.sum(fake_state, axis=[1, 2])  # <= (batch_size)
            fake_state = K.expand_dims(fake_state)  # <= (batch_size, 1)
            fake_state = K.tile(fake_state, [1, hidden_size])  # <= (batch_size, latent_dim
            return fake_state

        fake_state_c = create_initial_state(encoder_out_seq, encoder_out_seq.shape[-1])
        fake_state_e = create_initial_state(encoder_out_seq, encoder_out_seq.shape[1])  # <= (batch_size, enc_seq_len, latent_dim

        """ Computing energy outputs """
        # e_outputs => (batch_size, de_seq_len, en_seq_len)
        last_out, e_outputs, _ = K.rnn(
            energy_step, decoder_out_seq, [fake_state_e],
        )

        """ Computing context vectors """
        last_out, c_outputs, _ = K.rnn(
            context_step, e_outputs, [fake_state_c],
        )

        return c_outputs, e_outputs

    def compute_output_shape(self, input_shape):
        """ Outputs produced by the layer """
        return [
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[1][2])),
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[0][1]))
        ]
```

Gambar 21. Potongan skrip *tokenization* pada data ringkasan

3.2 Pelatihan model

Setelah arsitektur model dirancang, dilakukan proses pelatihan untuk mengoptimalkan parameter-parameter model. Untuk menghasilkan ringkasan yang akurat, model membutuhkan fungsi objektif dengan probabilitas bersyarat keluaran serta fungsi kerugian (*loss*) yang dapat membantu proses pelatihan yang disebut dengan *likelihood function*. Dalam penelitian Bahdanau *et al.* (2014) hal ini diformulasikan sebagai:

$$P(y) = \prod_{t=1}^{T} P(y_t|y_1, y_2, \dots, y_{t-1}, c) \tag{11}$$

maka,

$$P(y) = \prod_{t=1}^{T} P(y_t|y_{<t}, c) \tag{12}$$

$$\log P(y) = \log \prod_{t=1}^{T} P(y_t|y_{<t}, c) \tag{13}$$

$$\log P(y) = \prod_{t=1}^{T} \log P(y_t|y_{<t}, c) \tag{14}$$

dengan:
y_t : token ke-*t* dari keluaran,
$y_{<t}$: semua token sebelum waktu t,
c : hasil dari *encoder* + *attention*.

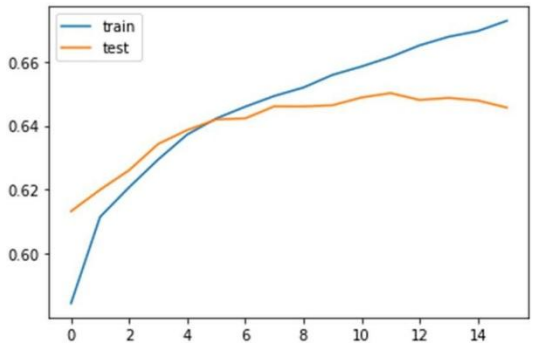
Dalam penelitian ini, fungsi kerugian di atas diimplementasikan ke dalam fungsi *loss* pada Keras yaitu *sparse_categorical_crossentropy*. Seperti yang dijelaskan oleh Goodfellow *et al.* (2016), maksimum likelihood menjadi peminimalan *negative log-likelihood* (NLL), atau secara setara, peminimalan *cross-entropy*. Dengan begitu, fungsi objektif *log-likelihood* di atas diubah menjadi bentuk *negative log-likelihood*, sehingga bentuk formulanya menjadi:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t|y_{<t}, c) \tag{15}$$

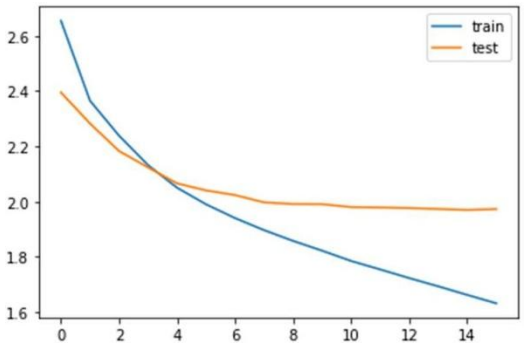
sedangkan untuk algoritma optimasi dipilih menggunakan RMSProp yang efektif untuk RNN seperti LSTM (Matrenin *et al.*, 2020). Model ini mendapatkan nilai akurasi (*accuracy*) sebesar sekitar 67.27% pada data pelatihan di *epoch* ke-16, dan akurasi sebesar sekitar 64.56% pada data validasi di *epoch* yang sama. Untuk fungsi kerugian (*loss*) mendapatkan nilai sebesar sekitar 1.6314 pada data pelatihan dan sekitar 1.9727 pada data validasi di epoch ke-16.

```
#Traing
model.compile(optimizer='rmsprop' ,
              loss='sparse_categorical_crossentropy' ,
              metrics=['accuracy'])
```

Gambar 22. Potongan skrip kompilasi model



Gambar 23. Grafik akurasi



Gambar 24. Grafik kerugian

3.3 Evaluasi dan Prediksi

Metode evaluasi dilakukan dengan menggunakan metrik BLEU (*Bilingual Evaluation Understudy*) untuk membandingkan hasil prediksi keluaran model dengan referensi manusia, dalam penelitian ini menggunakan teks ringkasan asli dalam data set. BLEU *score* menghitung kesamaan antara n-gram pada kalimat hasil prediksi (*candidate*) dengan kalimat referensi (*reference*) (Papineni *et al.*, 2002).

Misalkan *c* adalah panjang dari kalimat hasil prediksi (*candidate*) dan *r* adalah panjang efektif dari kalimat referensi (*reference*). Dalam penelitian Papineni *et al.* (2002), untuk menghitung penalti singkat (brevity penalty, BP), digunakan rumus:

$$BP = \begin{cases} 1 & jika\ c > r \\ e^{(1-\frac{r}{c})} & jika\ c \leq r \end{cases} \tag{16}$$

Untuk skor BLEU (Papineni *et al.*, 2002) dihitung dengan mengalikan penalti singkat ini dengan rata-rata geometrik dari presisi n-gram, yaitu:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{17}$$

dimana p_n adalah presisi untuk n-gram ke-*n*, dan w_n adalah bobot untuk n-gram tersebut (Papineni *et al.*, 2002). Dalam implementasi standar, digunakan $N = 4$ dan bobot yang seragam, yaitu $w_n = \frac{1}{N}$. Dalam penelitian Papineni *et al.* (2002), persamaan BLEU menjadi:

$$\log BLEU = \min\left(1 - \frac{r}{c}, 0\right) + \sum_{n=1}^{N} w_n \log p_n \tag{18}$$

Dalam penelitian ini, implementasi rumus BLEU *score* diterapkan dengan bahasa pemrograman Python menggunakan fungsi *sentence*_bleu() dari pustaka NLTK dengan pengaturan bobot *weights* yang merepresentasikan bobot kumulatif untuk n-gram dari 1 hingga 4. Untuk menghitung BLEU-1, yaitu hanya mempertimbangkan kemunculan 1-gram dengan bobot penuh pada $w_1 = 1$ dan sisanya nol. Untuk BLEU-2 dengan bobot rata untuk 1-gram dan 2-gram (komulatif) dengan $w_1 = 0{,}5$ dan $w_2 = 0{,}5$. Kemudian untuk BLEU-3 dengan total bobot = 1 (dibagi rata ke 3 n-gram), sedangkan untuk implementasi BLEU-4 komulatif secara langsung. Secara matematis, penjabaran BLEU (Papineni *et al.*, 2002) diformulakan sebagai:

$$BLEU_1 = BP \cdot \exp(\log p_1) = BP \cdot \exp p_1 \tag{19}$$

$$BLEU_2 = BP \cdot \exp(\frac{1}{2}(\log p_1 + \log p_2)) \quad (20)$$

$$BLEU_3 = BP \cdot \exp(\frac{1}{3}(\log p_1 + \log p_2 + \log p_3)) \quad (21)$$

$$BLEU_4 = BP \cdot \exp(\frac{1}{4}(\log p_1 + \log p_2 + \log p_3 + \log p_4)) \quad (22)$$

```
#cumulative BLEU scores
from nltk.translate.bleu_score import sentence_bleu
for i in range(0,1000):
  reference = seq2summary(y_test[i])
  candidate = decode_sequence(X_test[i].reshape(1, max_text_len))

print('Cumulative 1-gram: %f' % sentence_bleu(reference, candidate, weights=(1, 0, 0, 0)))
print('Cumulative 2-gram: %f' % sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0)))
print('Cumulative 3-gram: %f' % sentence_bleu(reference, candidate, weights=(0.33, 0.33, 0.33, 0)))
print('Cumulative 4-gram: %f' % sentence_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25)))
```

Gambar 25. Potongan skrip BLEU *score* komulatif

Tabel 2. Tabel hasil metode evaluasi BLEU *score* kumulatif

Jenis	*Score*
BLEU-1	0.428571
BLEU-2	0.654654
BLEU-3	0.756080
BLEU-4	0.809107

Nilai BLEU berada pada rentang nol hingga satu, di mana saat nilai mendekati satu, maka menunjukkan bahwa ringkasan semakin mendekati referensi (Mastropaolo *et al.*, 2023). Dengan kata lain, saat nilai BLEU mendekati satu, maka ringkasan dianggap koheren dan relevan. Dalam penelitian ini, *score* BLEU-1 hingga BLEU-4 terus meningkat dengan BLEU-4 sebesar 0.809107, ini menandakan bahwa model mampu menghasilkan ringkasan yang koheren dan relevan. Adapun hasil prediksi model ditunjukkan pada Tabel 3.

Tabel 3. Tabel perbandingan ringkasan hasil prediksi model dengan teks ringkasan asli

No	Teks ulasan	Teks ringkasan asli	Hasil prediksi
1	*thing buy lot stores much say love loves bought numerous times*	*great product great price*	*great product*
2	*tasty satisfying meat protien sticks perfect meal snack addition protien diet pepper flavor adds something missing regular flavor variety snacks recommended*	*meat of*	*delicious*
3	*chips great delicious healthier regular deli store chips first purchased supermarket found delicious found amazon cheaper ordered variety flavor flavors liking con addicting warned purchased amazon*	*delicious*	*great chips*
4	*always happy newman products coffee exception found priced love fact organic*	*coffee*	*great coffee*

4 SIMPULAN

Dengan menggunakan *unidirectional* LSTM yaitu 3 LSTM bertingkat satu arah pada *encoder,* dan pada *decoder* menggunakan *unidirectional* LSTM satu lapis yang menggunakan initial state dari encoder (state_h, state_c) serta menambahkan mekanisme *attention*, model berhasil mendapatkan akurasi sebesar 67,2%. Sedangkan untuk metrik BLEU berhasil mendapatkan skor komulatif BLEU-1 sebesar 0.428571, BLEU-2 sebesar 0.654654, BLEU-3 sebesar 0.756080, dan BLEU-4 sebesar 0.809107.

DAFTAR PUSTAKA

Akhter, B., & Mehra, M. (2022). A study of implementation of deep learning techniques for text summarization. *International Journal of Innovative Research in Engineering and Management*, *9*(2), 18–28.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate*.

Bystrov, V., Naboka-Krell, V., Staszewska-Bystrova, A., & Winker, P. (2023). Analysing the Impact of Removing Infrequent Words on Topic Quality in LDA Models. *ArXiv Preprint*, 1–21.

Chaurasia, S., Dasgupta, D., & Regunathan, R. (2023). T5LSTM-RNN based Text Summarization Model for Behavioral Biology Literature. *Procedia Computer Science*, *218*, 585–593. https://doi.org/10.1016/j.procs.2023.01.040

Febriana, A., Nur, Y., & Mariah, M. (2023). Pengaruh ulasan produk, kemudahan, dan keamanan terhadap keputusan pembelian produk secara online pada Shopee di Kota Makassar. *Jurnal Malomo : Manajemen Dan Akuntansi*, *1*(3), 281–292.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Luo, M., Xue, B., & Niu, B. (2024). A comprehensive survey for automatic text summarization: Techniques, approaches and perspectives. *Neurocomputing*, *603*, 128280. https://doi.org/10.1016/j.neucom.2024.128280

Mastropaolo, A., Ciniselli, M., Di Penta, M., & Bavota, G. (2023). *Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization*.

Matrenin, P. V., Manusov, V. Z., Khalyasmaa, A. I., Antonenkov, D. V., Eroshenko, S. A., & Butusov, D. N. (2020). Improving Accuracy and Generalization Performance of Small-Size Recurrent Neural Networks Applied to Short-Term Load Forecasting. *Mathematics*, *8*(12), 2169. https://doi.org/10.3390/math8122169

Muflikhah, L., Mahmudy, W. F., & Kurnianingtyas, D. (2023). *Machine learning: Cetakan Pertama*. UB Press.

Nambiar, S. K., Peter S, D., & Idicula, S. M. (2021). Attention based Abstractive Summarization of Malayalam Document. *Procedia Computer Science*, *189*, 250–257. https://doi.org/10.1016/j.procs.2021.05.088

Niu, Z., Zhong, G., & Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, *452*, 48–62. https://doi.org/10.1016/j.neucom.2021.03.091

Paaß, G., & Giesselbach, S. (2023). *Foundation Models for Natural Language Processing -- Pre-trained Language Models Integrating Media*.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, 311–318. https://doi.org/10.3115/1073083.1073135

Setyawan, C., Benarkah, N., & Prasetyo, V. R. (2021). Automatic Text Summarization Berdasarkan Pendekatan Statistika pada Dokumen Berbahasa Indonesia. *KELUWIH: Jurnal Sains Dan Teknologi*, *2*(1), 9–15. https://doi.org/10.24123/saintek.v2i1.4045

Shakil, H., Farooq, A., & Kalita, J. (2024). Abstractive text summarization: State of the art, challenges, and improvements. *Neurocomputing*, *603*, 128255. https://doi.org/10.1016/j.neucom.2024.128255

Supriyono, S., Wibawa, A. P., Suyono, S., & Kurniawan, F. (2024). A survey of text summarization: Techniques, evaluation and challenges. *Natural Language Processing Journal*, *7*, 100070. https://doi.org/10.1016/j.nlp.2024.100070

Tsai, C.-F., Chen, K., Hu, Y.-H., & Chen, W.-K. (2020). Improving text summarization of online hotel reviews with review helpfulness and sentiment. *Tourism Management*, *80*, 104122. https://doi.org/10.1016/j.tourman.2020.104122

Yudistira, N., Alfiansih, L. M. D., Andriyani, N. I., Essayem, W., Maulida, N., Maghfiroh, N. A., & Nurdian, I. W. (2023). *Prediksi deret waktu menggunakan deep learning*. UB Press.