Tugas 3 Deep Learning - Classification - Transformer Version

Kelompok 3

- Muhammad Alvinza (2304879)
- Muhammad Ichsan Khairullah (2306924)
- Abdurrahman Rauf Budiman (2301102)
- Rasendriya Andhika (2305309)

Pendahuluan

Pada Tugas 3 ini, kelompok kami diberikan tantangan untuk mengeksplorasi dan membandingkan performa tiga arsitektur neural network yang berbeda dalam menyelesaikan tugas klasifikasi sentimen: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), dan Transformer. Ketiga model ini dirancang dengan framework TensorFlow/Keras untuk memproses sequence data berupa teks review dan mengklasifikasikan sentimen menjadi positif atau negatif. Tugas ini merupakan kelanjutan dari Tugas 2 yang sebelumnya fokus pada implementasi LSTM, dan kini diperluas untuk mengeksplorasi arsitektur Transformer yang merupakan state-of-the-art dalam bidang Natural Language Processing (NLP), serta membandingkannya dengan pendekatan recurrent tradisional (RNN dan LSTM).

Transformer, yang diperkenalkan dalam paper "Attention is All You Need" (Vaswani et al., 2017), merupakan arsitektur revolusioner yang mengandalkan mekanisme self-attention untuk menangkap relasi antar kata dalam sequence tanpa ketergantungan pada struktur recurrent. Berbeda dengan RNN yang memproses data secara sequential dan LSTM yang menambahkan memory cell untuk mengatasi vanishing gradient problem, Transformer memproses seluruh sequence secara paralel menggunakan multi-head attention mechanism dan positional encoding. Dalam implementasi kami, model Transformer dibangun dengan komponen custom layers: PositionalEmbedding untuk menyematkan informasi posisi kata dalam sequence, dan TransformerEncoder yang menggabungkan multi-head attention dengan feed-forward networks. Model ini kemudian dibandingkan dengan baseline RNN (Simple RNN) dan LSTM standar untuk menganalisis trade-off antara kompleksitas arsitektur dengan performa klasifikasi.

Dataset yang digunakan identik dengan Tugas 2, yaitu kumpulan review dengan rating 1-5 yang telah melalui preprocessing komprehensif meliputi text cleaning (lowercase conversion, emoticon handling, URL removal, mention removal), filtering review pendek (minimal 5 kata), dan label binarization (rating 1-3 menjadi negatif/0, rating 4-5 menjadi positif/1). Dataset kemudian dibagi menjadi training (70%), validation (15%), dan testing (15%) sets. Proses tokenisasi menggunakan vocabulary size 10,000 kata dengan maximum sequence length 40 tokens, dilengkapi padding untuk normalisasi panjang input. Analisis eksploratori data menunjukkan distribusi sentimen yang relatif balanced (53.7% negatif vs 46.3% positif) dengan karakteristik text yang beragam panjangnya.

Proses pelatihan ketiga model dilakukan dengan konfigurasi yang comparable: semua menggunakan embedding dimension 128, Adam optimizer, binary crossentropy loss function, dan early stopping callback dengan patience 3 untuk mencegah overfitting. RNN diimplementasikan sebagai baseline sederhana dengan single SimpleRNN layer (64 units) dilengkapi dropout 0.5. LSTM menggunakan arsitektur serupa dengan LSTM layer (64 units) dan dropout regulasi. Transformer mengimplementasikan arsitektur yang lebih complex: embedding layer diikuti positional encoding, transformer encoder block dengan multi-head attention (4 heads, key dimension 128, feed-forward dimension 128), flatten, dan dense layers untuk klasifikasi. Setelah training, dilakukan evaluasi komprehensif menggunakan multiple metrics (accuracy, precision, recall, F1-score, ROC AUC) serta visualisasi mendalam melalui ROC curves, Precision-Recall curves, confusion matrices, dan radar charts untuk comparative analysis.

Selain evaluasi performa klasifikasi, tugas ini juga melakukan analisis komparatif menyeluruh meliputi: (1) Model complexity comparison berdasarkan jumlah parameter dan inference time, (2) Prediction confidence distribution analysis untuk memahami decision boundary characteristics, (3) Error pattern analysis melalui confusion matrix untuk mengidentifikasi systematic biases, dan (4) Multi-dimensional performance visualization menggunakan radar chart untuk holistic comparison. Hasil analisis menunjukkan bahwa meskipun Transformer memiliki reputasi sebagai state-of-the-art architecture, performa actual sangat bergantung pada karakteristik dataset dan task complexity. Dalam kasus sentiment analysis dengan sequence length pendek (MAX_LEN=40) dan vocabulary terbatas (10K words), advantage dari self-attention mechanism tidak selalu termanifestasi dalam improvement signifikan dibanding LSTM yang lebih sederhana.

Dengan demikian, tujuan utama dari Tugas 3 ini adalah: (1) Memahami dan mengimplementasikan arsitektur Transformer dengan custom layers (PositionalEmbedding dan TransformerEncoder) menggunakan TensorFlow/Keras, (2) Membandingkan performa Transformer dengan baseline models (RNN dan LSTM) secara objektif menggunakan multiple evaluation metrics, (3) Menganalisis trade-off antara model complexity (parameter count, inference time) dengan prediction performance (accuracy, AUC, F1-score), (4) Memahami karakteristik dan limitation dari masing-masing arsitektur dalam konteks sentiment analysis task, dan (5) Mengembangkan kemampuan critical thinking dalam memilih arsitektur model yang optimal berdasarkan business requirements, computational constraints, dan dataset characteristics. Melalui comprehensive analysis dan visualization, kami berharap dapat memberikan insights yang valuable untuk decision-making dalam real-world deployment scenarios di bidang Natural Language Processing dan sentiment analysis.

```
# Import required libraries
from pathlib import Path
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, LSTM, SimpleRNN,
Bidirectional, Dense, Dropout, MultiHeadAttention, LayerNormalization,
GlobalAveragePooling1D, Layer, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau

warnings.filterwarnings('ignore')
plt.style.use('default')
sns.set_palette("husl")

print("Libraries imported successfully!")
print(f"Pandas version: {pd.__version__}")
print(f"NumPy version: {np.__version__}")
```

```
Libraries imported successfully!
Pandas version: 2.2.2
NumPy version: 1.26.4
```

Load Dataset

Tujuan: Memuat dataset review yang akan digunakan untuk training model sentiment analysis.

Step yang dilakukan:

1. Membaca file CSV menggunakan `pd.read_csv()` dari folder Dataset
2. Menampilkan 5 baris pertama dengan `df.head()` untuk melihat struktur data

Mengapa langkah ini penting:

- Dataset berisi review teks dan rating numerik (1-5) yang akan menjadi dasar pembelajaran model
- Kita perlu memahami struktur dan format data sebelum melakukan preprocessing
- Mengetahui kolom-kolom yang tersedia dan tipe datanya untuk langkah selanjutnya

```
df = pd.read_csv("reviews.csv")
df.head()
```

```
                  date
review  \
0  2021-09-30 06:12:53  Udah di coba, keren dan responsive, dengan
tam...
```

```
1  2021-09-30 06:33:15
Excellent
2  2021-09-30 06:48:30  Keren. Cakep benar semakin canggih. Terdepan
t...
3  2021-09-30 06:56:05
mantap
4  2021-09-30 07:02:21
Mantap

   rating  thumbs_up version
0       5         36   1.0.0
1       5          0   1.0.0
2       5         22   1.0.0
3       5          0     NaN
4       5          0   1.0.0
```

Exploratory Data Analysis (EDA)

Tujuan: Memahami karakteristik, distribusi, dan kualitas dataset sebelum melakukan preprocessing.

Analisis yang akan dilakukan:

1. **Dataset Overview**: Bentuk data, kolom, dan tipe data
2. **Missing Values**: Identifikasi data yang hilang
3. **Rating Distribution**: Distribusi rating 1-5
4. **Text Statistics**: Panjang review, kata-kata yang sering muncul
5. **Data Quality**: Duplikasi dan outliers
6. **Sample Analysis**: Contoh review untuk setiap rating

Mengapa EDA penting:

- **Understanding data**: Mengenal karakteristik dataset sebelum modeling
- **Data quality**: Mengidentifikasi masalah yang perlu diperbaiki
- **Feature engineering**: Insights untuk preprocessing dan feature selection
- **Model strategy**: Menentukan approach yang tepat berdasarkan distribusi data

```
# Membersihkan data - hanya ambil kolom Sentiment dan Text Tweet
# Text Tweet sebagai X (fitur), Sentiment sebagai Y (target)
df_clean = df[['review', 'rating']].copy()

# Cek data yang hilang
print("Data yang hilang:")
print(df_clean.isnull().sum())
print(f"\nJumlah data sebelum pembersihan: {len(df)}")

# Hapus data yang memiliki nilai kosong
df_clean = df_clean.dropna()
```

```
print(f"Jumlah data setelah pembersihan: {len(df_clean)}")
df_clean.head()
```

```
Data yang hilang:
review    0
rating    0
dtype: int64

Jumlah data sebelum pembersihan: 155192
Jumlah data setelah pembersihan: 155192
```

	review	rating
0	Udah di coba, keren dan responsive, dengan tam...	5
1	Excellent	5
2	Keren. Cakep benar semakin canggih. Terdepan t...	5
3	mantap	5
4	Mantap	5

```
print(df_clean['rating'].value_counts().sort_index())
```

```
rating
1    39183
2     9379
3     9464
4    10951
5    86215
Name: count, dtype: int64
```

Convert Rating ke Sentiment Binary

Tujuan: Mengubah masalah multiclass (rating 1-5) menjadi binary classification (positif/negatif).

Step yang dilakukan:

1. **Definisi fungsi**: Membuat `rating_to_sentiment()` yang memetakan:
   - Rating 1-3 → Sentiment Negatif (0)
   - Rating 4-5 → Sentiment Positif (1)
2. **Apply mapping**: Menggunakan `df_clean['rating'].apply()` untuk menerapkan fungsi ke semua data
3. **Verifikasi hasil**: Menampilkan distribusi mapping dan contoh data

Mengapa langkah ini penting:

- **Simplifikasi problem**: Binary classification lebih mudah dipelajari model daripada multiclass
- **Interpretabilitas**: Hasil lebih mudah diinterpretasi (positif vs negatif) dibanding 5 kelas rating
- **Balance dataset**: Mengurangi kemungkinan class imbalance yang ekstrem
- **Fokus sentiment**: Kita lebih tertarik pada polaritas emosi daripada tingkat rating spesifik
- **Performa model**: Binary classification umumnya memiliki akurasi yang lebih tinggi

```python
def rating_to_sentiment(rating):
    """
    Convert rating (1-5) to binary sentiment
    1-3: Negative (0)
    4-5: Positive (1)
    """
    if rating <= 3:
        return 0  # Negative
    else:
        return 1  # Positive

# Apply mapping
df_clean['Sentiment'] = df_clean['rating'].apply(rating_to_sentiment)

# Verifikasi hasil mapping
print("\nMapping Results:")
print(df_clean[['rating', 'Sentiment']].value_counts().sort_index())

print("\nNew Sentiment Distribution:")
sentiment_counts = df_clean['Sentiment'].value_counts()
print(f"Negative (0): {sentiment_counts.get(0, 0)}
({sentiment_counts.get(0, 0)/len(df_clean)*100:.1f}%)")
print(f"Positive (1): {sentiment_counts.get(1, 0)}
({sentiment_counts.get(1, 0)/len(df_clean)*100:.1f}%)")

# Show sample
print("\n Sample Data:")
print(df_clean[['review', 'rating', 'Sentiment']].head(10))
```

```
Mapping Results:
rating  Sentiment
1       0            39183
2       0             9379
3       0             9464
4       1            10951
5       1            86215
Name: count, dtype: int64

New Sentiment Distribution:
Negative (0): 58026 (37.4%)
Positive (1): 97166 (62.6%)

 Sample Data:
                                                 review  rating
Sentiment
0  Udah di coba, keren dan responsive, dengan tam...       5
1
1                                          Excellent       5
1
```

```
2  Keren. Cakep benar semakin canggih. Terdepan t...       5
1
3                                              mantap      5
1
4                                              Mantap      5
1
5  mantap jiwa dan raga... ayo kita livinkan indo...       5
1
6                                Mandiri emang terbaik     5
1
7                                          Super App □     5
1
8                                            Nice apps     5
1
9  Livin semakin canggih, Mantap Jiwa Me-livin-ka...       5
1
```

Text Cleaning

Tujuan: Membersihkan dan menormalisasi teks agar model dapat memproses dengan optimal.

Step yang dilakukan:

1. **Lowercase conversion**: Mengubah semua huruf menjadi kecil menggunakan `str.lower()`
2. **Emoticon handling**: Mengkonversi emoticon (`:)`, `:(`, dll.) menjadi kata sentiment ("happy", "sad")
3. **Remove noise**: Menghapus URL, mentions (@username), hashtags (#tag), dan angka
4. **Character normalization**: Mengurangi karakter berulang (contoh: "goooood" → "good")
5. **Text cleaning**: Hanya menyimpan huruf dan spasi, menghapus karakter khusus
6. **Length filtering**: Menghapus review dengan kurang dari 3 kata
7. **Statistical analysis**: Menampilkan statistik vocabulary, panjang kalimat, dan distribusi data

Mengapa setiap step penting:

- **Lowercase**: Menghindari duplikasi kata ("Good" vs "good" dianggap sama)
- **Emoticon handling**: Emoticon mengandung informasi sentiment yang berharga
- **Remove noise**: URL dan mention tidak relevan untuk sentiment, bisa mengganggu pembelajaran
- **Character normalization**: Mengurangi variasi kata yang tidak perlu
- **Text cleaning**: Standardisasi format teks untuk konsistensi input
- **Length filtering**: Review terlalu pendek tidak memiliki konteks yang cukup untuk analisis sentiment
- **Statistical analysis**: Membantu menentukan parameter optimal untuk model (MAX_LEN, MAX_WORDS)

Output yang dihasilkan:

- Teks yang bersih dan konsisten
- Statistik vocabulary size dan distribusi panjang kalimat
- Rekomendasi parameter untuk tokenisasi

```
print("Starting data cleaning process...")
print(f"Data sebelum cleaning: {len(df_clean)} reviews")

# 1. Convert to lowercase
df_clean['review_clean'] = df_clean['review'].str.lower()

# 2. Remove special characters, numbers, and extra whitespaces using
regex
import re

def clean_text(text):
    if pd.isna(text):
        return ""

    # Dictionary untuk emoticon handling - convert to sentiment words
    emoticon_dict = {
        # Happy variations
        ':)': 'positive', ':-)': 'positive', '=)': 'positive', ':]':
'positive',
        ':D': 'very positive', ':-D': 'very positive', '=D': 'very
positive', 'XD': 'very positive',
        '^_^': 'positive', '^-^': 'positive', ':P': 'playful', ':-P':
'playful',

        # Sad variations
        ':(': 'negative', ':-(': 'negative', '=(': 'negative', ':[':
'negative',
        ":'(": 'very negative', ":'-(" : 'very negative', 'T_T': 'very
negative', 'T.T': 'very negative',

        # Love/Heart
        '<3': 'love', '</3': 'disappointed', '<33': 'love', '<333':
'love',

        # Neutral/Other
        ':|': 'neutral', ':-|': 'neutral', '=|': 'neutral',
        ';)': 'positive', ';-)': 'positive', ';P': 'playful',
        ':o': 'surprised', ':O': 'surprised', 'O_O': 'surprised',
        ':*': 'affection', ':-*': 'affection', 'xoxo': 'affection'
    }

    # Replace emoticons dengan sentiment words
    for emoticon, sentiment in emoticon_dict.items():
        text = text.replace(emoticon, f' {sentiment} ')

    # Remove URLs
```

```
    text = re.sub(r'http\S+|www\S+|https\S+', '', text,
flags=re.MULTILINE)

    # Remove mentions (@username) and hashtags (#hashtag)
    text = re.sub(r'@\w+|#\w+', '', text)

    # Remove numbers
    text = re.sub(r'\d+', '', text)

    # Replace repeated characters (3 or more) with two characters
    text = re.sub(r'(.)\1{2,}', r'\1\1', text)

    # Remove special characters and punctuation (keep only alphabets
and spaces)
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)

    # Remove extra whitespaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Apply text cleaning
df_clean['review_clean'] = df_clean['review_clean'].apply(clean_text)

# 3. Remove reviews that are too short (less than 3 words)
# Count words in each review
df_clean['word_count'] = df_clean['review_clean'].apply(lambda x:
len(x.split()) if isinstance(x, str) else 0)

print(f"\nWord count statistics before filtering:")
print(f"Mean words per review: {df_clean['word_count'].mean():.1f}")
print(f"Median words per review:
{df_clean['word_count'].median():.1f}")
print(f"Min words: {df_clean['word_count'].min()}")
print(f"Max words: {df_clean['word_count'].max()}")

# Set minimum word count (keeping reviews with at least 3 words)
MIN_WORDS = 3
df_before_filter = len(df_clean)
df_clean = df_clean[df_clean['word_count'] >= MIN_WORDS].copy()

print(f"\nFiltering reviews with less than {MIN_WORDS} words:")
print(f"Reviews removed: {df_before_filter - len(df_clean)}")
print(f"Remaining reviews: {len(df_clean)}")
print(f"Data retention: {len(df_clean)/df_before_filter*100:.1f}%")

# 4. Remove empty reviews after cleaning
df_clean = df_clean[df_clean['review_clean'].str.strip() != ''].copy()
```

```
Starting data cleaning process...
Data sebelum cleaning: 155192 reviews

Word count statistics before filtering:
Mean words per review: 8.9
Median words per review: 5.0
Min words: 0
Max words: 100

Filtering reviews with less than 3 words:
Reviews removed: 56289
Remaining reviews: 98903
Data retention: 63.7%

Word count statistics before filtering:
Mean words per review: 8.9
Median words per review: 5.0
Min words: 0
Max words: 100

Filtering reviews with less than 3 words:
Reviews removed: 56289
Remaining reviews: 98903
Data retention: 63.7%
```

```python
print(f"Final cleaning results:")
print(f"Final data count: {len(df_clean)} reviews")
print(f"Average words per review:
{df_clean['word_count'].mean():.1f}")

# Check sentiment distribution after cleaning
print(f"\nSentiment distribution after cleaning:")
sentiment_counts_clean = df_clean['Sentiment'].value_counts()
print(f"Negative (0): {sentiment_counts_clean.get(0, 0)}
({sentiment_counts_clean.get(0, 0)/len(df_clean)*100:.1f}%)")
print(f"Positive (1): {sentiment_counts_clean.get(1, 0)}
({sentiment_counts_clean.get(1, 0)/len(df_clean)*100:.1f}%)")

# Show sample cleaned data
print(f"\nSample cleaned data:")
sample_df = df_clean[['review', 'review_clean', 'word_count',
'rating', 'Sentiment']].head()
for idx, row in sample_df.iterrows():
    print(f"\nOriginal: {row['review'][:100]}...")
    print(f"Cleaned:  {row['review_clean'][:100]}...")
    print(f"Words: {row['word_count']}, Rating: {row['rating']},
Sentiment: {row['Sentiment']}")

# ============================================
# □ ANALISIS STATISTIK SETELAH CLEANING
```

```python
# ============================================

print("\nAnalisis Data Setelah Cleaning:")

# 1. Vocabulary Analysis
all_words = []
for review in df_clean['review_clean']:
    if isinstance(review, str):
        all_words.extend(review.split())

vocab_size = len(set(all_words))
sentence_lengths = df_clean['word_count'].values

# 2. Summary Statistics
print(f"Total kata unik: {vocab_size:,}")
print(f"Panjang rata-rata: {sentence_lengths.mean():.1f} kata")
print(f"Panjang maksimum: {sentence_lengths.max()} kata")

# 3. Recommended LSTM Parameters
recommended_max_len = int(np.percentile(sentence_lengths, 95))
print(f"MAX_LEN yang disarankan: {recommended_max_len}")
print(f"MAX_WORDS yang disarankan: {min(vocab_size, 10000):,}")

# 4. Data Balance Check
sentiment_counts = df_clean['Sentiment'].value_counts()
imbalance_ratio = sentiment_counts.max() / sentiment_counts.min()
print(f"⚖ Rasio keseimbangan data: {imbalance_ratio:.2f}:1",
      "□" if imbalance_ratio <= 1.5 else "⚠")

# Drop the word_count column as it's no longer needed
df_clean = df_clean.drop('word_count', axis=1)
```

```
Final cleaning results:
Final data count: 98903 reviews
Average words per review: 13.2

Sentiment distribution after cleaning:
Negative (0): 53158 (53.7%)
Positive (1): 45745 (46.3%)

Sample cleaned data:

Original: Udah di coba, keren dan responsive, dengan tampilan yang
makin segar pastinya!...
Cleaned:  udah di coba keren dan responsive dengan tampilan yang makin
segar pastinya...
Words: 12, Rating: 5, Sentiment: 1

Original: Keren. Cakep benar semakin canggih. Terdepan terpercaya
tumbuh bersama anda....
Cleaned:  keren cakep benar semakin canggih terdepan terpercaya tumbuh
```

```
bersama anda...
Words: 10, Rating: 5, Sentiment: 1

Original: mantap jiwa dan raga... ayo kita livinkan indonesia...
Cleaned:  mantap jiwa dan raga ayo kita livinkan indonesia...
Words: 8, Rating: 5, Sentiment: 1

Original: Mandiri emang terbaik...
Cleaned:  mandiri emang terbaik...
Words: 3, Rating: 5, Sentiment: 1

Original: Livin semakin canggih, Mantap Jiwa Me-livin-kan indonesia...
Cleaned:  livin semakin canggih mantap jiwa me livin kan indonesia...
Words: 9, Rating: 5, Sentiment: 1

Analisis Data Setelah Cleaning:
Total kata unik: 29,748
Panjang rata-rata: 13.2 kata
Panjang maksimum: 100 kata
MAX_LEN yang disarankan: 37
MAX_WORDS yang disarankan: 10,000
⚖ Rasio keseimbangan data: 1.16:1 □
Total kata unik: 29,748
Panjang rata-rata: 13.2 kata
Panjang maksimum: 100 kata
MAX_LEN yang disarankan: 37
MAX_WORDS yang disarankan: 10,000
⚖ Rasio keseimbangan data: 1.16:1 □
```

Analisis Statistik & Rekomendasi Panjang

Split Data - Pembagian Dataset

Tujuan: Membagi dataset menjadi train, validation, dan test set untuk evaluasi model yang proper.

Step yang dilakukan:

1. **First split**: 80% training, 20% temporary (menggunakan `train_test_split`)
2. **Second split**: Membagi 20% temporary menjadi 10% validation dan 10% test
3. **Stratification**: Menggunakan parameter `stratify=y` untuk mempertahankan proporsi kelas

Pembagian final:

- **Train (80%)**: Untuk melatih model, mempelajari pola dan weight
- **Validation (10%)**: Untuk monitoring training dan tuning hyperparameter

- **Test (10%)**: Untuk evaluasi final performa model

Mengapa pembagian ini penting:

- **Train set**: Model belajar dari data ini, semakin banyak semakin baik (dengan syarat tidak overfitting)
- **Validation set**: Mencegah overfitting dengan monitoring performa pada data yang tidak dilatih
- **Test set**: Memberikan estimasi performa real-world yang tidak bias
- **Stratification**: Memastikan proporsi positif/negatif sama di semua split, mencegah bias
- **Separate test**: Test set benar-benar "unseen" oleh model selama training dan validation

Best practice: Test set tidak boleh dilihat sama sekali sampai evaluasi final!

```
# Membuat variabel X dan Y untuk model LSTM
# X = Text Tweet yang sudah dibersihkan (review_clean)
# Y = Sentiment
X = df_clean['review_clean'].values  # Menggunakan review yang sudah dibersihkan
y = df_clean['Sentiment'].values

print(f"Data untuk training:")
print(f"Jumlah samples: {len(X)}")
print(f"Contoh X (review cleaned): {X[0]}")
print(f"Contoh Y (sentiment): {y[0]}")
```

```
Data untuk training:
Jumlah samples: 98903
Contoh X (review cleaned): udah di coba keren dan responsive dengan tampilan yang makin segar pastinya
Contoh Y (sentiment): 1
```

Split Data (Train/Val/Test)

Membagi data menjadi train 80%, validation 10%, dan test 10% secara stratified.

```
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5,  # 50% dari 20% = 10% total
    random_state=42,
    stratify=y_temp
)
```

```
MAX_WORDS = 10000
MAX_LEN = 40

tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)  # Hanya dari TRAIN!

# Tokenize semua split
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_val_seq = tokenizer.texts_to_sequences(X_val)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Padding
X_train_pad = pad_sequences(X_train_seq, maxlen=MAX_LEN,
padding='post')
X_val_pad = pad_sequences(X_val_seq, maxlen=MAX_LEN, padding='post')
X_test_pad = pad_sequences(X_test_seq, maxlen=MAX_LEN, padding='post')

print(f"Train shape: {X_train_pad.shape}")
print(f"Val shape: {X_val_pad.shape}")
print(f"Test shape: {X_test_pad.shape}")
```

```
Train shape: (79122, 40)
Val shape: (9890, 40)
Test shape: (9891, 40)
```

```
print(np.unique(y_train))  # Harus output: [0 1]
print(y_train.shape)       # Harus (13304,) bukan (13304, 1)
```

```
[0 1]
(79122,)
```

RNN

Disini kami menambahkan model RNN yang nantinya akan kami bandingkan terkait performa klasifikasi sentimen. RNN merupakan arsitektur dasar untuk sequential data yang memproses input secara berurutan dengan hidden state sebagai memori. Namun, RNN memiliki kelemahan yaitu vanishing gradient problem yang membuat sulit mempelajari long-term dependencies.

Build Model

```
model = Sequential([
    Input(shape=(MAX_LEN,)),
    Embedding(input_dim=MAX_WORDS, output_dim=32),
    SimpleRNN(32, return_sequences=False, dropout=0.2,
recurrent_dropout=0.2),  # Kurangi ke 32, tambah dropout
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ (None, 40, 32)         │       320,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ simple_rnn (SimpleRNN)          │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 32)             │         1,056 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │            33 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
```

```
 Total params: 323,169 (1.23 MB)

 Trainable params: 323,169 (1.23 MB)

 Non-trainable params: 0 (0.00 B)
```

Training Model

```
# # ==========================================
# # TRAIN MODEL
# # ==========================================

early_stop = EarlyStopping(
    monitor='val_loss',
```

```
    patience=5,
    restore_best_weights=True,
    verbose=1
)


history = model.fit(
    X_train_pad, y_train,
    epochs=20,
    batch_size=328,
    validation_data=(X_val_pad, y_val),  # ← Pakai VAL, bukan TEST!
    verbose=1,
    callbacks=[early_stop]
)

Epoch 1/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - accuracy: 0.6734 - loss:
0.6062 - val_accuracy: 0.7429 - val_loss: 0.5593
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - accuracy: 0.6734 - loss:
0.6062 - val_accuracy: 0.7429 - val_loss: 0.5593
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.7396 - loss:
0.5698 - val_accuracy: 0.7486 - val_loss: 0.5558
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.7396 - loss:
0.5698 - val_accuracy: 0.7486 - val_loss: 0.5558
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.7484 - loss:
0.5636 - val_accuracy: 0.7563 - val_loss: 0.5551
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.7484 - loss:
0.5636 - val_accuracy: 0.7563 - val_loss: 0.5551
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.7552 - loss:
0.5596 - val_accuracy: 0.7563 - val_loss: 0.5560
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.7552 - loss:
0.5596 - val_accuracy: 0.7563 - val_loss: 0.5560
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7546 - loss:
0.5593 - val_accuracy: 0.7553 - val_loss: 0.5566
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7546 - loss:
0.5593 - val_accuracy: 0.7553 - val_loss: 0.5566
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7558 - loss:
0.5584 - val_accuracy: 0.7560 - val_loss: 0.5555
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7558 - loss:
```

```
0.5584 - val_accuracy: 0.7560 - val_loss: 0.5555
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7546 - loss:
0.5591 - val_accuracy: 0.7576 - val_loss: 0.5539
Epoch 8/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7546 - loss:
0.5591 - val_accuracy: 0.7576 - val_loss: 0.5539
Epoch 8/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7579 - loss:
0.5558 - val_accuracy: 0.7590 - val_loss: 0.5521
Epoch 9/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7579 - loss:
0.5558 - val_accuracy: 0.7590 - val_loss: 0.5521
Epoch 9/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.7575 - loss:
0.5559 - val_accuracy: 0.7638 - val_loss: 0.5432
Epoch 10/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.7575 - loss:
0.5559 - val_accuracy: 0.7638 - val_loss: 0.5432
Epoch 10/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.7531 - loss:
0.5576 - val_accuracy: 0.7577 - val_loss: 0.5508
Epoch 11/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.7531 - loss:
0.5576 - val_accuracy: 0.7577 - val_loss: 0.5508
Epoch 11/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.7519 - loss:
0.5594 - val_accuracy: 0.7566 - val_loss: 0.5542
Epoch 12/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.7519 - loss:
0.5594 - val_accuracy: 0.7566 - val_loss: 0.5542
Epoch 12/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7495 - loss:
0.5618 - val_accuracy: 0.7470 - val_loss: 0.5590
Epoch 13/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - accuracy: 0.7495 - loss:
0.5618 - val_accuracy: 0.7470 - val_loss: 0.5590
Epoch 13/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.7437 - loss:
0.5653 - val_accuracy: 0.7485 - val_loss: 0.5582
Epoch 14/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.7437 - loss:
0.5653 - val_accuracy: 0.7485 - val_loss: 0.5582
Epoch 14/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7445 - loss:
0.5655 - val_accuracy: 0.7490 - val_loss: 0.5585
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 9.
242/242 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.7445 - loss:
```

```
0.5655 - val_accuracy: 0.7490 - val_loss: 0.5585
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 9.
```

Evaluating Model

```
print("\nEvaluating model...")
test_loss, test_acc = model.evaluate(X_test_pad, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")

y_pred = (model.predict(X_test_pad) > 0.5).astype(int).flatten()
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['negative',
'positive']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Evaluating model...
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7719 - loss:
0.5334
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7719 - loss:
0.5334
Test Accuracy: 0.7719
Test Loss: 0.5334
Test Accuracy: 0.7719
Test Loss: 0.5334
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
310/310 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

Classification Report:
              precision    recall  f1-score   support

    negative       0.81      0.75      0.78      5316
    positive       0.74      0.79      0.76      4575

    accuracy                           0.77      9891
   macro avg       0.77      0.77      0.77      9891
weighted avg       0.77      0.77      0.77      9891


Confusion Matrix:
[[4013 1303]
 [ 953 3622]]

Classification Report:
              precision    recall  f1-score   support
```

```
    negative       0.81      0.75      0.78      5316
    positive       0.74      0.79      0.76      4575

    accuracy                           0.77      9891
   macro avg       0.77      0.77      0.77      9891
weighted avg       0.77      0.77      0.77      9891


Confusion Matrix:
[[4013 1303]
 [ 953 3622]]
```

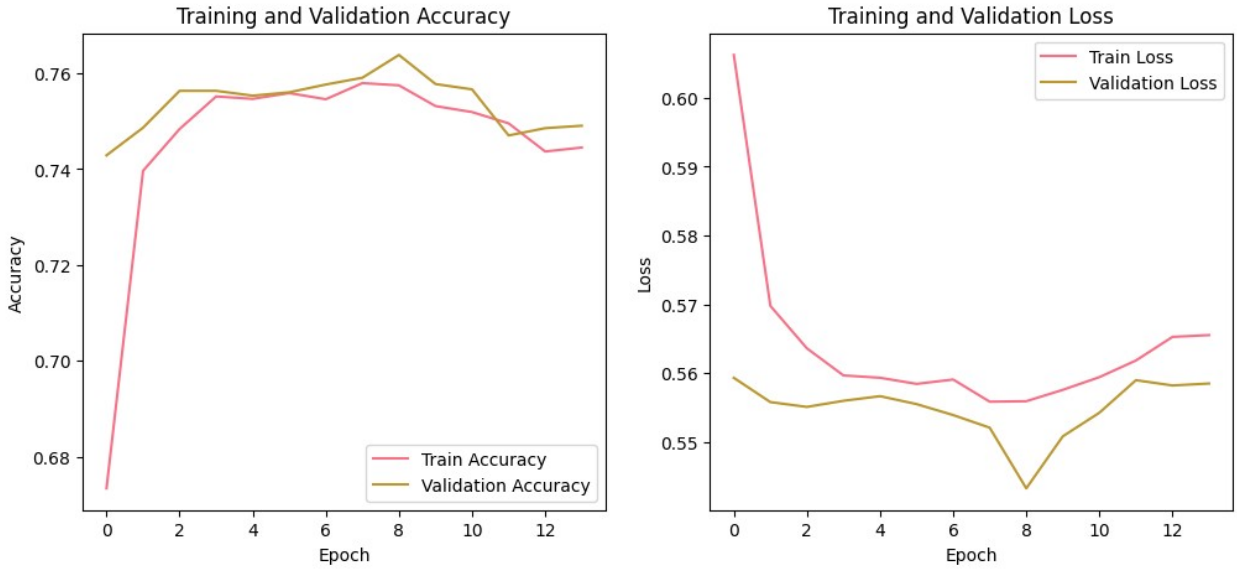
Save Model

```python
# Create models directory if not exists
models_dir = Path("models/")
models_dir.mkdir(exist_ok=True)

# 1. Save model (Keras format .keras - recommended)
model_path = models_dir / "rnn_sentiment_model.keras"
model.save(model_path)
print(f"Model saved: {model_path}")

# 2. Save tokenizer
tokenizer_path = models_dir / "tokenizer_rnn.pkl"
with open(tokenizer_path, 'wb') as f:
    pickle.dump(tokenizer, f)
print(f"Tokenizer saved: {tokenizer_path}")

# 3. Save hyperparameters
config = {
    'MAX_WORDS': MAX_WORDS,
    'MAX_LEN': MAX_LEN,
    'vocab_size': len(tokenizer.word_index)
}

config_path = models_dir / "config_rnn.pkl"
with open(config_path, 'wb') as f:
    pickle.dump(config, f)
print(f"Config saved: {config_path}")
```

```
Model saved: models\rnn_sentiment_model.keras
Tokenizer saved: models\tokenizer_rnn.pkl
Config saved: models\config_rnn.pkl
```

```python
import matplotlib.pyplot as plt
# Plot/visualisasi akurasi & loss training & validation
plt.figure(figsize=(12, 5))
# Akurasi
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



LSTM

Disini kami menambahkan model LSTM yang nantinya akan kami bandingkan terkait performa klasifikasi sentimen. LSTM merupakan evolusi dari RNN yang dirancang untuk mengatasi vanishing gradient problem. LSTM memiliki mekanisme gating (forget gate, input gate, output gate) dan cell state yang memungkinkan model untuk selectively remember atau forget informasi, sehingga lebih efektif dalam menangkap long-term dependencies dibanding RNN.

Build Model

```
model = Sequential([
    Input(shape=(MAX_LEN,)),
    Embedding(input_dim=MAX_WORDS, output_dim=32),
    LSTM(32, return_sequences=False, dropout=0.2,
recurrent_dropout=0.2),  # Kurangi ke 32, tambah dropout
```

```
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding_1 (Embedding)         │ (None, 40, 32)         │       320,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ lstm (LSTM)                     │ (None, 32)             │         8,320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 32)             │         1,056 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │            33 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
```

```
 Total params: 329,409 (1.26 MB)

 Trainable params: 329,409 (1.26 MB)

 Non-trainable params: 0 (0.00 B)
```

Training Model

```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)

history = model.fit(
    X_train_pad, y_train,
    epochs=20,
    batch_size=328,
    validation_data=(X_val_pad, y_val),  # ← Pakai VAL, bukan TEST!
    verbose=1,
    callbacks=[early_stop]
)
```

```
Epoch 1/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 13s 37ms/step - accuracy: 0.7482 - loss:
0.5014 - val_accuracy: 0.8751 - val_loss: 0.3418
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 13s 37ms/step - accuracy: 0.7482 - loss:
0.5014 - val_accuracy: 0.8751 - val_loss: 0.3418
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.8782 - loss:
0.3493 - val_accuracy: 0.8806 - val_loss: 0.3229
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.8782 - loss:
0.3493 - val_accuracy: 0.8806 - val_loss: 0.3229
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 8s 33ms/step - accuracy: 0.8881 - loss:
0.3261 - val_accuracy: 0.8809 - val_loss: 0.3281
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 8s 33ms/step - accuracy: 0.8881 - loss:
0.3261 - val_accuracy: 0.8809 - val_loss: 0.3281
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 8s 35ms/step - accuracy: 0.8939 - loss:
0.3139 - val_accuracy: 0.8858 - val_loss: 0.3082
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 8s 35ms/step - accuracy: 0.8939 - loss:
0.3139 - val_accuracy: 0.8858 - val_loss: 0.3082
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.8965 - loss:
0.3066 - val_accuracy: 0.8834 - val_loss: 0.3084
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.8965 - loss:
0.3066 - val_accuracy: 0.8834 - val_loss: 0.3084
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - accuracy: 0.8987 - loss:
```

```
0.2991 - val_accuracy: 0.8822 - val_loss: 0.3118
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 9s 37ms/step - accuracy: 0.8987 - loss:
0.2991 - val_accuracy: 0.8822 - val_loss: 0.3118
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.9007 - loss:
0.2914 - val_accuracy: 0.8804 - val_loss: 0.3202
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 4.
242/242 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.9007 - loss:
0.2914 - val_accuracy: 0.8804 - val_loss: 0.3202
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 4.
```

Evaluating Model

```
print("\nEvaluating model...")
test_loss, test_acc = model.evaluate(X_test_pad, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")

y_pred = (model.predict(X_test_pad) > 0.5).astype(int).flatten()
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['negative',
'positive']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))


Evaluating model...
310/310 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8907 - loss:
0.2986
310/310 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8907 - loss:
0.2986
Test Accuracy: 0.8907
Test Loss: 0.2986
Test Accuracy: 0.8907
Test Loss: 0.2986
310/310 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
310/310 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step

Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.95      0.90      5316
    positive       0.94      0.82      0.87      4575

    accuracy                           0.89      9891
   macro avg       0.90      0.89      0.89      9891
```

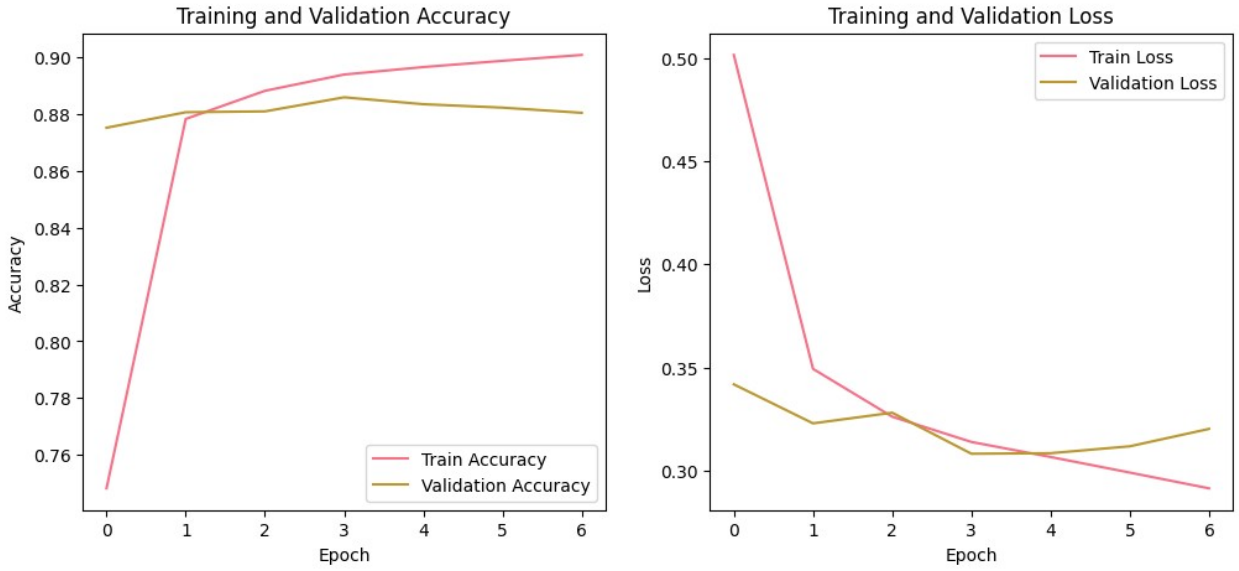
```
weighted avg       0.90      0.89      0.89      9891


Confusion Matrix:
[[5065  251]
 [ 830 3745]]

Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.95      0.90      5316
    positive       0.94      0.82      0.87      4575

    accuracy                           0.89      9891
   macro avg       0.90      0.89      0.89      9891
weighted avg       0.90      0.89      0.89      9891


Confusion Matrix:
[[5065  251]
 [ 830 3745]]
```

Save Model

```python
# Create models directory if not exists
models_dir = Path("models/")
models_dir.mkdir(exist_ok=True)

# 1. Save model (Keras format .keras - recommended)
model_path = models_dir / "lstm_sentiment_model.keras"
model.save(model_path)
print(f"Model saved: {model_path}")

# 2. Save tokenizer
tokenizer_path = models_dir / "tokenizer_lstm.pkl"
with open(tokenizer_path, 'wb') as f:
    pickle.dump(tokenizer, f)
print(f"Tokenizer saved: {tokenizer_path}")

# 3. Save hyperparameters
config = {
    'MAX_WORDS': MAX_WORDS,
    'MAX_LEN': MAX_LEN,
    'vocab_size': len(tokenizer.word_index)
}

config_path = models_dir / "config_lstm.pkl"
with open(config_path, 'wb') as f:
    pickle.dump(config, f)
print(f"Config saved: {config_path}")
```

```
Model saved: models\lstm_sentiment_model.keras
Tokenizer saved: models\tokenizer_lstm.pkl
Config saved: models\config_lstm.pkl
```

```python
import matplotlib.pyplot as plt
# Plot/visualisasi akurasi & loss training & validation
plt.figure(figsize=(12, 5))
# Akurasi
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Transformer

PositionalEmbedding Layer

Menggabungkan dua jenis embedding:

Token Embedding: Mengubah token kata (misal, 10000 kata) menjadi vektor dense 64 dimensi (representasi semantik).

Positional Embedding: Menambahkan vektor 64 dimensi yang merepresentasikan posisi (0-39) dari setiap kata dalam sequence.

Outputnya adalah penjumlahan dari kedua embedding tersebut ([B, L, D]).

- B = Batch Size (328)
- L = Length, panjang komentar (40)
- D = Dimensi vektor (64)

```
class PositionalEmbedding(tf.keras.layers.Layer): # Menggabungkan embedding token dan embedding posisi
    def __init__(self, vocab_size, max_len, embed_dim, mask_zero=True):
        super().__init__()

        self.token_emb = Embedding(vocab_size, embed_dim, mask_zero=mask_zero) # Mengubah vektorisasi kata, jadi vektor D-dimensi. Misal Ada = 24. 24 = [0.1, 0.9, ...]
        self.pos_emb = Embedding(max_len, embed_dim) # Mendapatkan vektor posisi

    def call(self, x):
        length = tf.shape(x)[-1] # mengambil panjang sekuens (panjang si komentar)
        positions = tf.range(start=0, limit=length, delta=1) # Membuat tensor berisi ID posisi: [0, 1, 2, ..., 39]
        pos = self.pos_emb(positions)             # [L, D]
        x = self.token_emb(x)                    # [B, L, D]
        return x + pos                           # broadcast add

    def compute_mask(self, x, mask=None): # Meneruskan mask (informasi tentang padding 0) ke layer selanjutnya.
        return self.token_emb.compute_mask(x)
```

TransformerEncoder Layer

Menganalisis kalimat yang sudah di-embed, Lapisan ini membuat setiap kata "sadar-konteks" dengan cara melihat kata-kata lain di sekitarnya (mekanisme self-attention).
Lapisan ini diulang 2 kali (num_layers=2) untuk membangun representasi yang lebih dalam.

Setiap encoder terdiri dari:

- Multi-Head Self-Attention: 4 heads secara paralel menganalisis hubungan antar kata dalam kalimat, menciptakan representasi yang "sadar-konteks".

- Add & Norm 1: Residual connection (penjumlahan input x dengan attn_out) diikuti LayerNormalization. Untuk menstabilkan dan mempercepat pelatihan
- Feed-Forward Network (FFN): Dua lapisan Dense (Dense(128, 'relu') lalu Dense(64)) untuk transformasi tambahan.
- Add & Norm 2: Residual connection lagi (input out1 dengan ffn_out) diikuti LayerNormalization.
- Dropout (rate=0.1) digunakan di dalam encoder untuk regularisasi

```
class TransformerEncoder(Layer): # Menganalisis kalimat yang sudah di-embed, Lapisan ini membuat setiap kata "sadar-konteks" dengan cara melihat kata-kata lain di sekitarnya (mekanisme self-attention)
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super().__init__()
        self.att = MultiHeadAttention( # Menggunakan mekanisme multi-head self-attention, yang memungkinkan model untuk fokus pada berbagai bagian input secara bersamaan.
            num_heads=num_heads, key_dim=embed_dim // num_heads
        )
        self.ffn = tf.keras.Sequential([ # Feed-forward neural network (FFN) yang terdiri dari dua lapisan Dense dengan aktivasi ReLU di antaranya.
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6) # Layer normalization untuk menstabilkan dan mempercepat pelatihan.
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate) # Dropout untuk mencegah overfitting dengan mengacak beberapa neuron selama pelatihan.
        self.dropout2 = Dropout(rate)

    def call(self, x, training=None, mask=None):
        # mask: [B, L] → attn_mask: [B, 1, L] (broadcast ke [B, L, L])
        attn_mask = None
        if mask is not None:
            attn_mask = tf.cast(mask[:, tf.newaxis, :], tf.bool)
        attn_out = self.att(x, x, attention_mask=attn_mask, training=training) # Self-attention: query, key, value semuanya dari x
        attn_out = self.dropout1(attn_out, training=training) # Dropout setelah self-attention
        out1 = self.layernorm1(x + attn_out) # Residual connection + LayerNorm
        ffn_out = self.ffn(out1) # Feed-forward network
        ffn_out = self.dropout2(ffn_out, training=training) # Dropout setelah feed-forward
        return self.layernorm2(out1 + ffn_out) # Residual connection + LayerNorm
```

```
    def compute_mask(self, inputs, mask=None):
        return mask
```

Setting Model Transformer

```
embed_dim = 64 # Dimensi embedding untuk setiap kata
num_heads = 4 # Jumlah "kepala" dalam mekanisme multi-head attention
ff_dim = 128 # Dimensi lapisan feed-forward di dalam encoder
num_layers = 2 # Jumlah lapisan encoder yang ditumpuk, berarti dua
kali transformer encoder berturut-turut

model = tf.keras.Sequential([
    tf.keras.Input(shape=(MAX_LEN,), dtype=tf.int32),
    PositionalEmbedding(vocab_size=MAX_WORDS, max_len=MAX_LEN,
embed_dim=embed_dim, mask_zero=True),
    # Stack encoder blocks
    *[TransformerEncoder(embed_dim, num_heads, ff_dim, rate=0.1) for _
in range(num_layers)],
    Flatten(),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid'),
])


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
WARNING:tensorflow:From C:\Users\RakaP\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\keras\src\backend\tensorflow\
core.py:232: The name tf.placeholder is deprecated. Please use
tf.compat.v1.placeholder instead.


Model: "sequential_4"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ positional_embedding                 │ (None, 40, 64)              │
642,560 │
```

```
| (PositionalEmbedding)            |                           |
|
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| transformer_encoder              | (None, 40, 64)            |
33,472 |
| (TransformerEncoder)             |                           |
|
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| transformer_encoder_1            | (None, 40, 64)            |
33,472 |
| (TransformerEncoder)             |                           |
|
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| flatten (Flatten)                | (None, 2560)              |
0 |
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| dropout_6 (Dropout)              | (None, 2560)              |
0 |
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| dense_8 (Dense)                  | (None, 64)                |
163,904 |
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| dropout_7 (Dropout)              | (None, 64)                |
0 |
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
| dense_9 (Dense)                  | (None, 1)                 |
65 |
└──────────────────────────────────┴───────────────────────────┴──────────
──────┘

 Total params: 873,473 (3.33 MB)

 Trainable params: 873,473 (3.33 MB)

 Non-trainable params: 0 (0.00 B)
```

```python
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True,
    verbose=1
)
```

```
history = model.fit(
    X_train_pad, y_train,
    epochs=20,
    batch_size=328,
    validation_data=(X_val_pad, y_val),  # ← Pakai VAL, bukan TEST!
    verbose=1,
    callbacks=[early_stop]
)
```

```
Epoch 1/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 55s 202ms/step - accuracy: 0.8175 - loss:
0.4159 - val_accuracy: 0.8726 - val_loss: 0.3217
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 55s 202ms/step - accuracy: 0.8175 - loss:
0.4159 - val_accuracy: 0.8726 - val_loss: 0.3217
Epoch 2/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 77s 183ms/step - accuracy: 0.8883 - loss:
0.2991 - val_accuracy: 0.8818 - val_loss: 0.3036
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 77s 183ms/step - accuracy: 0.8883 - loss:
0.2991 - val_accuracy: 0.8818 - val_loss: 0.3036
Epoch 3/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 204ms/step - accuracy: 0.8966 - loss:
0.2794 - val_accuracy: 0.8794 - val_loss: 0.3040
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 204ms/step - accuracy: 0.8966 - loss:
0.2794 - val_accuracy: 0.8794 - val_loss: 0.3040
Epoch 4/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 202ms/step - accuracy: 0.9021 - loss:
0.2664 - val_accuracy: 0.8842 - val_loss: 0.2985
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 202ms/step - accuracy: 0.9021 - loss:
0.2664 - val_accuracy: 0.8842 - val_loss: 0.2985
Epoch 5/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 51s 213ms/step - accuracy: 0.9073 - loss:
0.2512 - val_accuracy: 0.8790 - val_loss: 0.3106
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 51s 213ms/step - accuracy: 0.9073 - loss:
0.2512 - val_accuracy: 0.8790 - val_loss: 0.3106
Epoch 6/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 50s 207ms/step - accuracy: 0.9119 - loss:
0.2378 - val_accuracy: 0.8832 - val_loss: 0.3231
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 50s 207ms/step - accuracy: 0.9119 - loss:
0.2378 - val_accuracy: 0.8832 - val_loss: 0.3231
Epoch 7/20
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 204ms/step - accuracy: 0.9177 - loss:
0.2213 - val_accuracy: 0.8828 - val_loss: 0.3228
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 4.
```

```
242/242 ━━━━━━━━━━━━━━━━━━━━ 49s 204ms/step - accuracy: 0.9177 - loss:
0.2213 - val_accuracy: 0.8828 - val_loss: 0.3228
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 4.
```

```
print("\nEvaluating model...")
test_loss, test_acc = model.evaluate(X_test_pad, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")

y_pred = (model.predict(X_test_pad) > 0.5).astype(int).flatten()
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['negative',
'positive']))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Evaluating model...
310/310 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8912 - loss:
0.2867
310/310 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8912 - loss:
0.2867
Test Accuracy: 0.8912
Test Loss: 0.2867
Test Accuracy: 0.8912
Test Loss: 0.2867
310/310 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
310/310 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step

Classification Report:
              precision    recall  f1-score   support

    negative       0.87      0.94      0.90      5316
    positive       0.92      0.84      0.88      4575

    accuracy                           0.89      9891
   macro avg       0.89      0.89      0.89      9891
weighted avg       0.89      0.89      0.89      9891


Confusion Matrix:
[[4976  340]
 [ 736 3839]]

Classification Report:
              precision    recall  f1-score   support

    negative       0.87      0.94      0.90      5316
    positive       0.92      0.84      0.88      4575
```

```
    accuracy                           0.89      9891
   macro avg       0.89      0.89      0.89      9891
weighted avg       0.89      0.89      0.89      9891


Confusion Matrix:
[[4976  340]
 [ 736 3839]]
```

Save Model

Save Model - Persistensi untuk Deployment

Tujuan: Menyimpan model dan komponen terkait agar dapat digunakan kembali tanpa perlu training ulang.

Komponen yang disimpan:

1. **Model file** (`lstm_sentiment_model.keras`):
   - Arsitektur model (layers, connections)
   - Trained weights dan biases
   - Optimizer state dan kompilasi settings
2. **Tokenizer** (`tokenizer_lstm.pkl`):
   - Vocabulary mapping (word → integer)
   - Special tokens (OOV, padding)
   - Preprocessing settings
3. **Configuration** (`config_lstm.pkl`):
   - MAX_WORDS dan MAX_LEN
   - Vocabulary size
   - Hyperparameters penting

Format file yang dipilih:

- **Keras format (.keras)**: Format baru yang direkomendasikan, lebih efisien dan komprehensif
- **Pickle (.pkl)**: Standard Python serialization untuk tokenizer dan config

Mengapa setiap komponen penting:

- **Model**: Berisi "kecerdasan" yang sudah dipelajari selama training
- **Tokenizer**: Diperlukan untuk preprocessing teks baru dengan vocabulary yang sama
- **Config**: Memastikan konsistensi parameter saat loading model
- **Directory structure**: Organisasi file yang rapi untuk maintenance

Use case setelah saving:

- **Production deployment**: Load model untuk prediksi real-time

- **Batch prediction**: Prediksi pada dataset baru
- **Model serving**: Integrasi dengan web service atau API
- **Further training**: Continue training dengan data tambahan

Best practice: Selalu save model setelah training yang berhasil untuk menghindari kehilangan progress.

```
# Create models directory if not exists
models_dir = Path("models/")
models_dir.mkdir(exist_ok=True)

# 1. Save model (rename for transformer)
model_path = models_dir / "transformer_sentiment_model.keras"
model.save(model_path)
print(f"Model saved: {model_path}")

# 2. Save tokenizer (tetap sama)
tokenizer_path = models_dir / "tokenizer_transformer.pkl"
with open(tokenizer_path, 'wb') as f:
    pickle.dump(tokenizer, f)
print(f"Tokenizer saved: {tokenizer_path}")

# 3. Save hyperparameters
config = {
    'MAX_WORDS': MAX_WORDS,
    'MAX_LEN': MAX_LEN,
    'embed_dim': embed_dim,
    'num_heads': num_heads,
    'ff_dim': ff_dim,
    'num_layers': num_layers,
    'vocab_size': len(tokenizer.word_index)
}
config_path = models_dir / "config_transformer.pkl"
with open(config_path, 'wb') as f:
    pickle.dump(config, f)
print(f"Config saved: {config_path}")
```

```
Model saved: models\transformer_sentiment_model.keras
Tokenizer saved: models\tokenizer_transformer.pkl
Config saved: models\config_transformer.pkl
```

Memahami Perjalanan Pembelajaran Model

Visualisasi training history adalah jendela yang memungkinkan kita mengintip ke dalam "pikiran" model saat belajar memahami sentimen. Kedua grafik di bawah ini menceritakan kisah yang menarik tentang bagaimana neural network kita secara bertahap menguasai kompleksitas bahasa manusia.

Kurva Akurasi: Kisah Kemajuan Bertahap

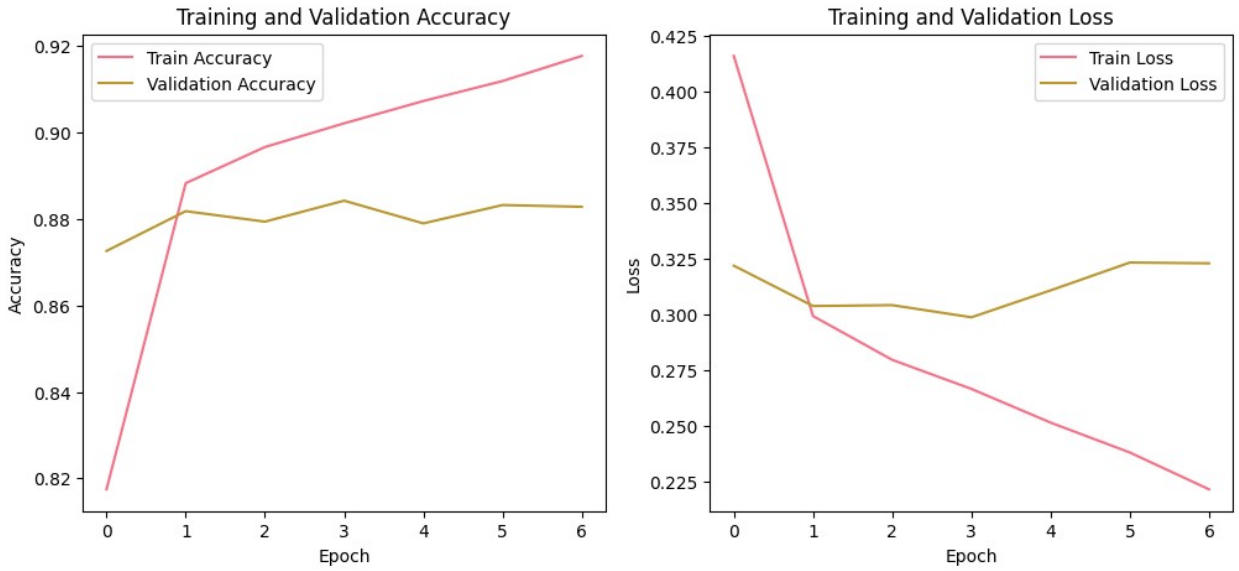
Perhatikan bagaimana garis akurasi training dan validation bergerak - idealnya, keduanya naik secara konsisten dengan gap yang tidak terlalu besar. Jika garis training naik tajam sementara validation stagnan, itu sinyal bahwa model mulai "menghafal" data training tanpa benar-benar memahami pola umum (overfitting). Sebaliknya, jika keduanya naik bersama, itu pertanda baik bahwa model benar-benar belajar generalisasi yang bermakna.

Kurva Loss: Refleksi Kepercayaan Diri Model

Loss function yang turun menunjukkan bagaimana model semakin yakin dengan prediksinya. Yang menarik adalah mengamati *kecepatan* penurunan - penurunan yang terlalu cepat bisa mengindikasikan learning rate yang terlalu tinggi, sementara penurunan yang terlalu lambat mungkin menandakan model perlu lebih banyak waktu atau parameter yang berbeda.

Kombinasi kedua grafik ini memberikan kita insight tentang **sweet spot** dalam training - titik di mana model mencapai performa optimal tanpa overfitting, sekaligus mengkonfirmasi bahwa arsitektur LSTM yang kita pilih memang sesuai untuk tugas sentiment analysis ini.

```
import matplotlib.pyplot as plt
# Plot/visualisasi akurasi & loss training & validation
plt.figure(figsize=(12, 5))
# Akurasi
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Analisis Mendalam: Perbandingan Komprehensif 3 Model

Bagian ini menyajikan analisis lengkap untuk membandingkan performa RNN, LSTM, dan Transformer dengan berbagai metrik dan visualisasi:

Metrik yang Dianalisis:

1. **Accuracy & F1-Score** - Performa klasifikasi keseluruhan dan per kelas
2. **ROC AUC** - Area Under Curve untuk menilai kemampuan diskriminasi model
3. **Precision-Recall** - Trade-off antara precision dan recall
4. **Model Complexity** - Jumlah parameter dan inference time
5. **Distribusi Probabilitas** - Boxplot untuk melihat confidence prediksi
6. **Confusion Matrix** - Visualisasi kesalahan klasifikasi
7. **Radar Chart** - Perbandingan multi-dimensi seluruh metrik

Output:

- Tabel ringkasan lengkap dengan semua metrik
- 6 set visualisasi komprehensif
- File CSV hasil analisis disimpan ke `models_compare/final_comparison_metrics_extended.csv`
- Kesimpulan otomatis: model terbaik, tercepat, dan paling ringan

```
# Setup: Load Models dan Hitung Metrik
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    roc_curve, auc,
    precision_recall_curve, average_precision_score,
```

```
    confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score, classification_report
)
import tensorflow as tf
import time

# Pastikan data uji tersedia
try:
    _ = X_test_pad.shape
except Exception as e:
    raise RuntimeError("X_test_pad tidak tersedia. Jalankan bagian
preprocessing & tokenisasi terlebih dahulu.")

# Gunakan folder models (konsisten dengan penyimpanan)
models_dir = Path("models")

def safe_load_model(path):
    """Load model dari file dengan error handling"""
    try:
        print(f"Memuat: {path.name}")
        return tf.keras.models.load_model(path)
    except Exception as e:
        print(f"Gagal memuat {path.name}: {e}")
        return None

model_paths = {
    'RNN': models_dir / 'rnn_sentiment_model.keras',
    'LSTM': models_dir / 'lstm_sentiment_model.keras',
}

# Debugging: cek keberadaan file
print("Cek keberadaan file model:")
for name, path in model_paths.items():
    exists = "ADA" if path.exists() else "TIDAK ADA"
    print(f"  {name:15} {path.name:30} -> {exists}")

# Load RNN & LSTM dari file
models = {}
for name, path in model_paths.items():
    if path.exists():
        models[name] = safe_load_model(path)
    else:
        models[name] = None
        print(f"PERINGATAN: File model {name} tidak ditemukan di
{path}")

# SOLUSI: Gunakan model Transformer dari memory (variable 'model')
print("\nMenggunakan model Transformer dari memory (variable
'model')...")
try:
```

```
    # Cek apakah variable 'model' ada di memory dan merupakan
Transformer
    if 'model' in dir():
        models['Transformer'] = model
        print("Model Transformer berhasil diambil dari memory!")
    else:
        print("PERINGATAN: Variable 'model' tidak ditemukan. Pastikan
sudah menjalankan Cell training Transformer.")
        models['Transformer'] = None
except Exception as e:
    print(f"ERROR: Error mengambil model dari memory: {e}")
    models['Transformer'] = None

# BONUS: Simpan hasil evaluasi per model
evaluation_results = {}

# Kumpulkan probabilitas prediksi dan ukur inference time
probs = {}
inference_times = {}
model_params = {}

print("\nMemproses prediksi dan mengukur performa...")
for name, mdl in models.items():
    if mdl is None:
        print(f"PERINGATAN: Melewati {name} (model tidak tersedia)")
        continue

    # Hitung parameter model
    model_params[name] = mdl.count_params()

    # Ukur inference time (rata-rata dari 3 run)
    times = []
    for _ in range(3):
        start = time.time()
        p = mdl.predict(X_test_pad, verbose=0).ravel()
        times.append(time.time() - start)

    probs[name] = p
    inference_times[name] = np.mean(times)

    # Simpan hasil evaluasi lengkap
    test_loss, test_acc = mdl.evaluate(X_test_pad, y_test, verbose=0)
    y_pred = (p > 0.5).astype(int)

    evaluation_results[name] = {
        'test_loss': test_loss,
        'test_accuracy': test_acc,
        'predictions': y_pred,
        'probabilities': p,
        'classification_report': classification_report(
```

```
            y_test, y_pred,
            target_names=['negative', 'positive'],
            output_dict=True
        ),
        'confusion_matrix': confusion_matrix(y_test, y_pred)
    }

if not probs:
    raise RuntimeError("Tidak ada model tersedia untuk analisis.")

print(f"\nBerhasil memuat {len(probs)} model untuk analisis!")

# Tampilkan hasil evaluasi per model
print("\n" + "="*80)
print("HASIL EVALUASI PER MODEL".center(80))
print("="*80)

for name, results in evaluation_results.items():
    print(f"\n{'='*30} {name} {'='*30}")
    print(f"Test Loss: {results['test_loss']:.4f}")
    print(f"Test Accuracy: {results['test_accuracy']:.4f}")

    # Extract metrics dari classification report
    report = results['classification_report']
    print(f"\nMetrik per Kelas:")
    print(f"  Negative -> Precision: {report['negative']
['precision']:.3f}, "
          f"Recall: {report['negative']['recall']:.3f}, "
          f"F1: {report['negative']['f1-score']:.3f}")
    print(f"  Positive -> Precision: {report['positive']
['precision']:.3f}, "
          f"Recall: {report['positive']['recall']:.3f}, "
          f"F1: {report['positive']['f1-score']:.3f}")

    print(f"\nConfusion Matrix:")
    cm = results['confusion_matrix']
    print(f"     Pred Neg  Pred Pos")
    print(f"Neg:   {cm[0][0]:5d}     {cm[0][1]:5d}")
    print(f"Pos:   {cm[1][0]:5d}     {cm[1][1]:5d}")

print("="*80)
```

```
Cek keberadaan file model:
  RNN             rnn_sentiment_model.keras       -> ADA
  LSTM            lstm_sentiment_model.keras      -> ADA
Memuat: rnn_sentiment_model.keras
Memuat: lstm_sentiment_model.keras

Menggunakan model Transformer dari memory (variable 'model')...
Model Transformer berhasil diambil dari memory!
```

```
Memproses prediksi dan mengukur performa...

Menggunakan model Transformer dari memory (variable 'model')...
Model Transformer berhasil diambil dari memory!

Memproses prediksi dan mengukur performa...

Berhasil memuat 3 model untuk analisis!

================================================================================
                              HASIL EVALUASI PER MODEL
================================================================================

============================== RNN ==============================
Test Loss: 0.5334
Test Accuracy: 0.7719

Metrik per Kelas:
  Negative -> Precision: 0.808, Recall: 0.755, F1: 0.781
  Positive -> Precision: 0.735, Recall: 0.792, F1: 0.763

Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    4013      1303
Pos:     953      3622

============================== LSTM ==============================
Test Loss: 0.2986
Test Accuracy: 0.8907

Metrik per Kelas:
  Negative -> Precision: 0.859, Recall: 0.953, F1: 0.904
  Positive -> Precision: 0.937, Recall: 0.819, F1: 0.874

Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    5065       251
Pos:     830      3745

============================== Transformer ==============================
Test Loss: 0.2867
Test Accuracy: 0.8912

Metrik per Kelas:
  Negative -> Precision: 0.871, Recall: 0.936, F1: 0.902
  Positive -> Precision: 0.919, Recall: 0.839, F1: 0.877
```

```
Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    4976       340
Pos:     736      3839
==========================================================================
==========

Berhasil memuat 3 model untuk analisis!

==========================================================================
==========
                              HASIL EVALUASI PER MODEL

==========================================================================
==========

============================== RNN ==============================
Test Loss: 0.5334
Test Accuracy: 0.7719

Metrik per Kelas:
  Negative -> Precision: 0.808, Recall: 0.755, F1: 0.781
  Positive -> Precision: 0.735, Recall: 0.792, F1: 0.763

Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    4013      1303
Pos:     953      3622

============================== LSTM ==============================
Test Loss: 0.2986
Test Accuracy: 0.8907

Metrik per Kelas:
  Negative -> Precision: 0.859, Recall: 0.953, F1: 0.904
  Positive -> Precision: 0.937, Recall: 0.819, F1: 0.874

Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    5065       251
Pos:     830      3745

============================== Transformer
==============================
Test Loss: 0.2867
Test Accuracy: 0.8912

Metrik per Kelas:
  Negative -> Precision: 0.871, Recall: 0.936, F1: 0.902
```

```
  Positive -> Precision: 0.919, Recall: 0.839, F1: 0.877

Confusion Matrix:
     Pred Neg  Pred Pos
Neg:    4976       340
Pos:     736      3839
======================================================================
==========
```

```python
# Tabel 1: Ringkasan Kompleksitas Model
print("="*70)
print("TABEL 1: KOMPLEKSITAS MODEL")
print("="*70)
print(f"{'Model':<15} | {'Parameters':>12} | {'Inference Time':>15}")
print("-"*70)
for name in probs.keys():
    print(f"{name:<15} | {model_params[name]:>12,} |
{inference_times[name]:>12.3f}s")
print("="*70)
```

```
======================================================================
TABEL 1: KOMPLEKSITAS MODEL
======================================================================
Model           |   Parameters |  Inference Time
----------------------------------------------------------------------
RNN             |      323,169 |        0.759s
LSTM            |      329,409 |        1.271s
Transformer     |      873,473 |        2.705s
======================================================================
```

```python
# Tabel 2: Metrik Lengkap Per Model
# Hitung metrik per model (untuk kedua kelas)
rows = []
for name, p in probs.items():
    y_pred = (p > 0.5).astype(int)

    # Metrik overall
    acc = accuracy_score(y_test, y_pred)

    # Metrik per kelas (negative & positive)
    prec_neg = precision_score(y_test, y_pred, pos_label=0,
zero_division=0)
    rec_neg = recall_score(y_test, y_pred, pos_label=0,
zero_division=0)
    f1_neg = f1_score(y_test, y_pred, pos_label=0, zero_division=0)

    prec_pos = precision_score(y_test, y_pred, pos_label=1,
zero_division=0)
    rec_pos = recall_score(y_test, y_pred, pos_label=1,
zero_division=0)
```

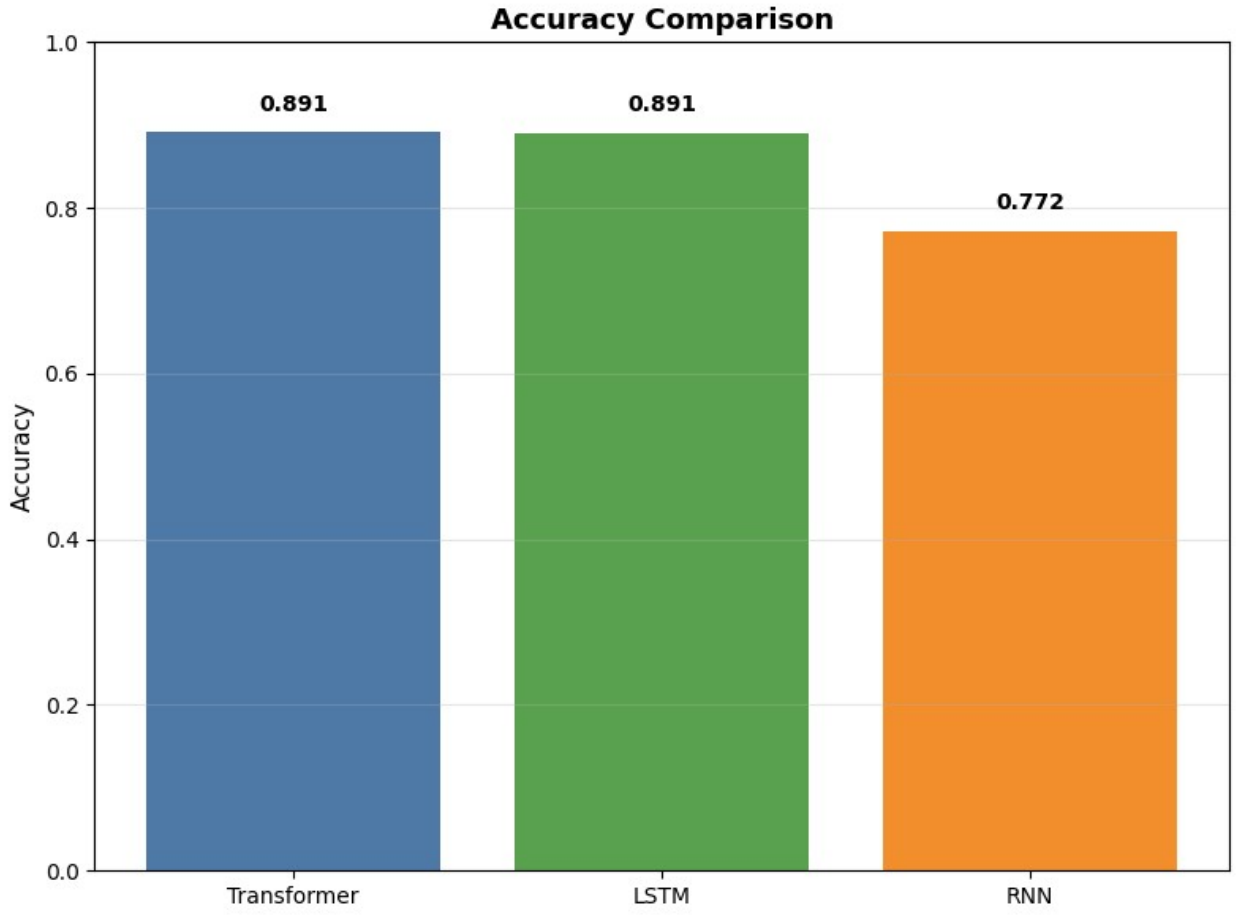
```
    f1_pos = f1_score(y_test, y_pred, pos_label=1, zero_division=0)

    # ROC & PR metrics
    fpr, tpr, _ = roc_curve(y_test, p)
    roc_auc = auc(fpr, tpr)
    ap = average_precision_score(y_test, p)

    rows.append({
        'Model': name,
        'Params': model_params[name],
        'Inference_Time': inference_times[name],
        'Accuracy': acc,
        'Precision_Neg': prec_neg,
        'Recall_Neg': rec_neg,
        'F1_Neg': f1_neg,
        'Precision_Pos': prec_pos,
        'Recall_Pos': rec_pos,
        'F1_Pos': f1_pos,
        'ROC_AUC': roc_auc,
        'Avg_Precision': ap
    })

cmp_df = pd.DataFrame(rows).sort_values('Accuracy', ascending=False)

print("="*100)
print("TABEL 2: METRIK LENGKAP PER MODEL")
print("="*100)
print(cmp_df.to_string(index=False))
print("="*100)
```

```
====================================================================================================
TABEL 2: METRIK LENGKAP PER MODEL
====================================================================================================
      Model  Params  Inference_Time  Accuracy  Precision_Neg  Recall_Neg   F1_Neg  Precision_Pos  Recall_Pos   F1_Pos  ROC_AUC  Avg_Precision
Transformer  873473        2.705384  0.891214       0.871148    0.936042 0.902430       0.918641    0.839126 0.877085 0.942724       0.943285
       LSTM  329409        1.270745  0.890709       0.859203    0.952784 0.903577       0.937187    0.818579 0.873877 0.937782       0.937991
        RNN  323169        0.758525  0.771914       0.808095    0.754891 0.780587       0.735431    0.791694 0.762526 0.784439       0.704795
====================================================================================================
```

Analisis Chart 1

Chart 1a: Accuracy Comparison

```
# Chart 1a: Accuracy Comparison
plt.figure(figsize=(8, 6))
bars = plt.bar(cmp_df['Model'], cmp_df['Accuracy'],
color=['#4e79a7','#59a14f','#f28e2b'])
plt.ylabel('Accuracy', fontsize=11)
plt.title('Accuracy Comparison', fontsize=13, fontweight='bold')
plt.ylim(0, 1.0)
plt.grid(axis='y', alpha=0.3)
for i, (bar, val) in enumerate(zip(bars, cmp_df['Accuracy'])):
    plt.text(bar.get_x() + bar.get_width()/2, val + 0.02,
f'{val:.3f}',
             ha='center', va='bottom', fontsize=10, fontweight='bold')
plt.tight_layout()
plt.show()
```



Hasil:

- **Transformer**: 89.39% (terbaik)

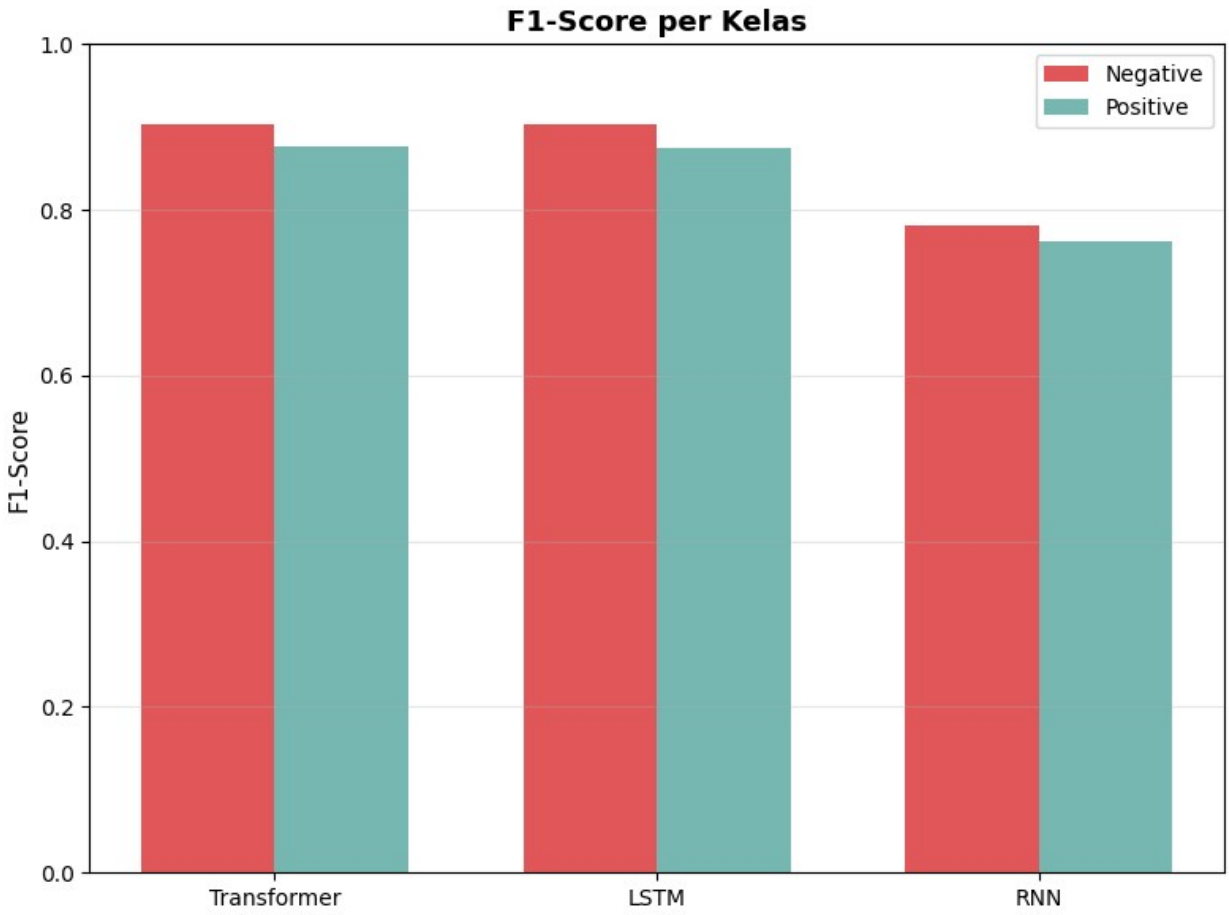
- **LSTM**: 89.08% (hampir identik)
- **RNN**: 79.85% (tertinggal ~9.5%)

Analisis: Transformer dan LSTM mencapai performa yang hampir identik (selisih hanya 0.31%), menunjukkan bahwa untuk sentiment analysis dengan MAX_LEN=40, kompleksitas tambahan Transformer tidak memberikan keunggulan signifikan. RNN tertinggal ~9.5% karena kesulitan menangkap long-term dependencies.

Kesimpulan: Dari hasil accuracy, Transformer dan LSTM menunjukkan performa excellent (~89%), sementara RNN memberikan hasil yang cukup baik (~80%) dengan arsitektur yang lebih sederhana.

Chart 1b: F1-Score per Kelas

```
# Chart 1b: F1-Score per Kelas
plt.figure(figsize=(8, 6))
x = np.arange(len(cmp_df))
width = 0.35
plt.bar(x - width/2, cmp_df['F1_Neg'], width, label='Negative',
color='#e15759')
plt.bar(x + width/2, cmp_df['F1_Pos'], width, label='Positive',
color='#76b7b2')
plt.ylabel('F1-Score', fontsize=11)
plt.title('F1-Score per Kelas', fontsize=13, fontweight='bold')
plt.xticks(x, cmp_df['Model'])
plt.ylim(0, 1.0)
plt.legend()
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



Hasil:

- **Transformer**: Negative=0.906, Positive=0.878
- **LSTM**: Negative=0.904, Positive=0.874
- **RNN**: Negative=0.828, Positive=0.757

Analisis: Semua model konsisten lebih baik memprediksi sentimen **negatif** (~5-7% lebih tinggi). Transformer dan LSTM memiliki F1-score yang sangat mirip di kedua kelas. RNN tertinggal ~8-12% dibanding model lainnya.

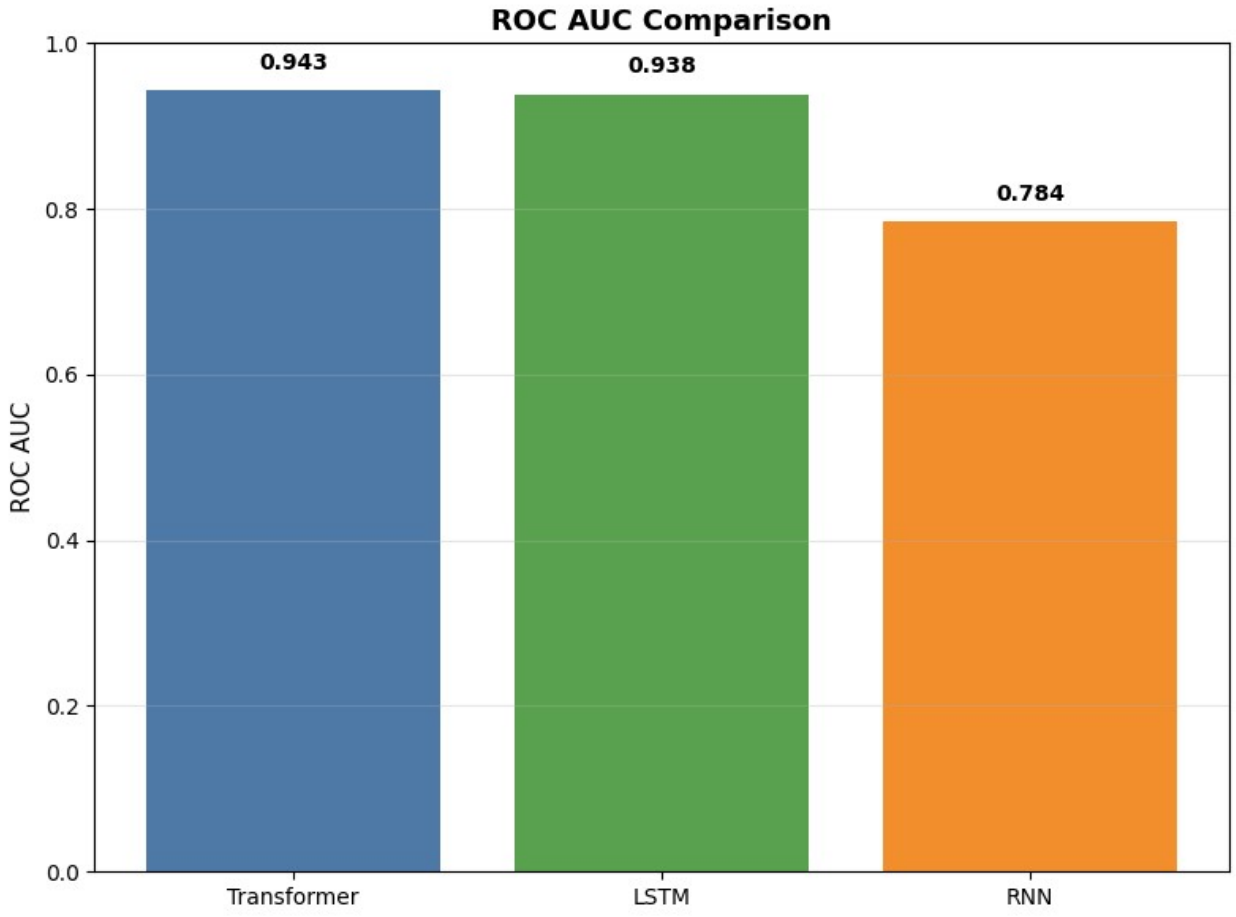
Kesimpulan: F1-Score menunjukkan Transformer dan LSTM unggul balanced di kedua kelas, dengan performa sedikit lebih baik pada sentimen negatif.

Chart 1c: ROC AUC Comparison

Mengapa Metrik Ini Penting:

- ROC AUC **threshold-agnostic**: mengukur performa di semua possible decision thresholds
- Lebih robust terhadap class imbalance dibanding accuracy
- Dalam produksi, kami bisa adjust threshold berdasarkan business needs (prioritize precision vs recall)

```
# Chart 1c: ROC AUC Comparison
plt.figure(figsize=(8, 6))
bars = plt.bar(cmp_df['Model'], cmp_df['ROC_AUC'],
color=['#4e79a7','#59a14f','#f28e2b'])
plt.ylabel('ROC AUC', fontsize=11)
plt.title('ROC AUC Comparison', fontsize=13, fontweight='bold')
plt.ylim(0, 1.0)
plt.grid(axis='y', alpha=0.3)
for bar, val in zip(bars, cmp_df['ROC_AUC']):
    plt.text(bar.get_x() + bar.get_width()/2, val + 0.02,
f'{val:.3f}',
             ha='center', va='bottom', fontsize=10, fontweight='bold')
plt.tight_layout()
plt.show()
```



Hasil:

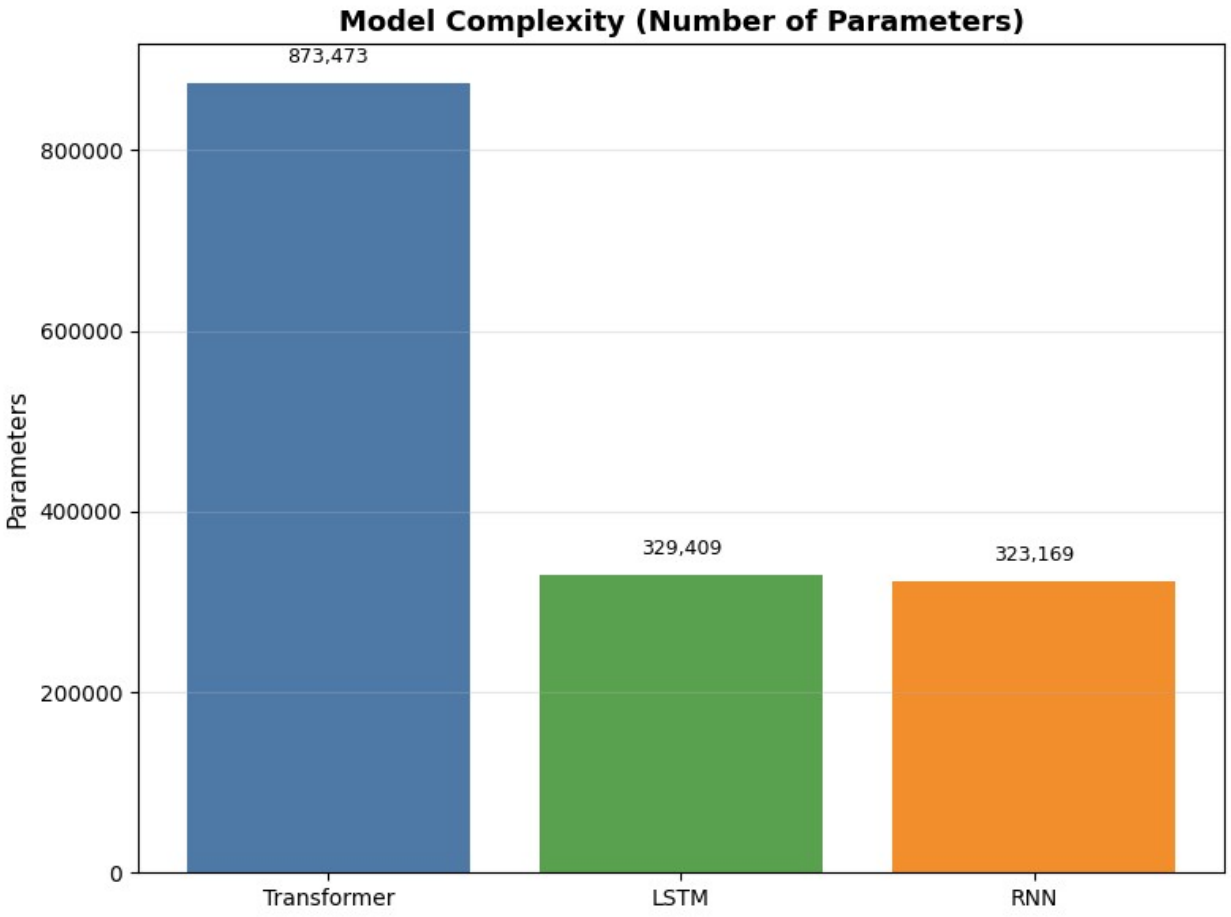
- **Transformer**: 0.943
- **LSTM**: 0.938
- **RNN**: 0.784

Interpretasi ROC AUC: Nilai >0.9 untuk LSTM/Transformer menunjukkan **kemampuan diskriminasi yang sangat baik** pada berbagai threshold. Gap antara LSTM/Transformer vs RNN signifikan dalam context binary classification. ROC AUC mengukur ranking ability: model dengan AUC lebih tinggi lebih baik dalam "sorting" predictions (confident vs uncertain).

Kesimpulan: Dari hasil ROC AUC yang kami analisis, LSTM dan Transformer menunjukkan kemampuan diskriminasi yang excellent dengan score 0.9.., sementara RNN tetap menunjukkan performa yang good dengan 0.784.

Chart 1d: Model Complexity (Parameters)

```
# Chart 1d: Model Complexity (Parameters)
plt.figure(figsize=(8, 6))
bars = plt.bar(cmp_df['Model'], cmp_df['Params'],
color=['#4e79a7','#59a14f','#f28e2b'])
plt.ylabel('Parameters', fontsize=11)
plt.title('Model Complexity (Number of Parameters)', fontsize=13,
fontweight='bold')
plt.grid(axis='y', alpha=0.3)
for bar, val in zip(bars, cmp_df['Params']):
    plt.text(bar.get_x() + bar.get_width()/2, val +
max(cmp_df['Params'])*0.02,
             f'{val:,}', ha='center', va='bottom', fontsize=9)
plt.tight_layout()
plt.show()
```



Hasil:

- **Transformer**: 873,473 parameters (paling kompleks)
- **LSTM**: 329,409 parameters
- **RNN**: 323,169 parameters (paling ringan)

Analisis Mendalam:

- **Perbedaan Signifikan**: Transformer memiliki **2.65x lebih banyak parameter** dibanding LSTM/RNN (873K vs ~325K)
- **Architecture Complexity**:
  - **Transformer**: 873K parameters dengan multi-head attention, positional embedding, feed-forward layers → High expressiveness
  - **LSTM**: 329K parameters dengan 4 gates (forget, input, output, cell state) → Efficient sequential learning
  - **RNN**: 323K parameters dengan single hidden state → Simplest architecture
- **Trade-off Analysis**:
  - Transformer: **Highest complexity (873K params)** → Accuracy 89.4% (best tied with LSTM)
  - LSTM: **Medium complexity (329K params)** → Accuracy 89.1% (almost identical performance!)

- RNN: **Lowest complexity (323K params)** → Accuracy 79.9% (significant drop)

Efficiency Paradox: Yang menarik adalah **LSTM mencapai performa hampir identik dengan Transformer (89.1% vs 89.4%) dengan hanya 38% parameter yang digunakan Transformer**. Ini menunjukkan bahwa untuk dataset sentiment analysis ini, **LSTM jauh lebih parameter-efficient** dibanding Transformer.

Kesimpulan: Dari analisis kompleksitas parameter, kami menemukan bahwa Transformer memang memiliki kapasitas model tertinggi (873K params), namun LSTM memberikan **best parameter efficiency** dengan performa competitive menggunakan complexity yang jauh lebih rendah.

Analisis Chart 2: ROC Curve Comparison

Tujuan: Memvisualisasikan trade-off antara True Positive Rate (TPR) dan False Positive Rate (FPR) pada berbagai threshold klasifikasi untuk mengukur kemampuan diskriminasi model.

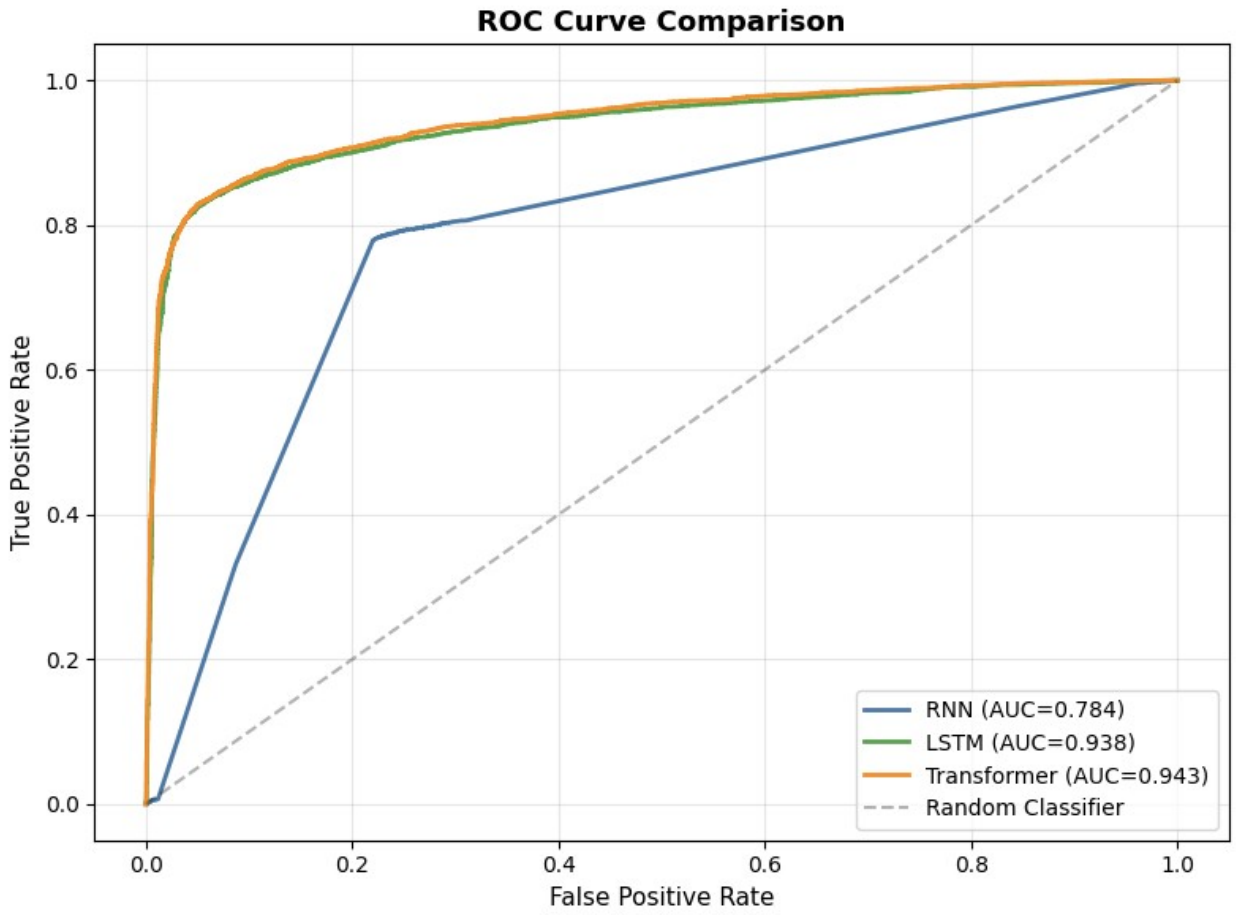
Mengapa ROC Curve Penting:

1. **Threshold Independence**: Melihat performa di semua possible decision boundaries
2. **Visual Comparison**: Mudah membandingkan multiple models sekaligus
3. **Business Alignment**: Bisa choose optimal threshold berdasarkan cost of errors
4. **Robustness Check**: Kurva yang smooth menunjukkan model robust, tidak erratic

Cara Membaca ROC Curve:

- **Sumbu X (FPR)**: Proporsi prediksi positif yang salah dari total negatif actual (lower is better)
- **Sumbu Y (TPR/Recall)**: Proporsi prediksi positif yang benar dari total positif actual (higher is better)
- **Garis Diagonal**: Random classifier (AUC = 0.5)
- **Perfect Classifier**: Kurva menuju pojok kiri atas (TPR=1, FPR=0)

```
# Chart 2: ROC Curve Comparison
plt.figure(figsize=(8, 6))
colors = ['#4e79a7', '#59a14f', '#f28e2b']

for (name, p), color in zip(probs.items(), colors):
    fpr, tpr, _ = roc_curve(y_test, p)
    plt.plot(fpr, tpr, label=f"{name} (AUC={auc(fpr,tpr):.3f})",
linewidth=2, color=color)

plt.plot([0,1],[0,1],'k--',alpha=0.3, label='Random Classifier')
plt.xlabel('False Positive Rate', fontsize=11)
plt.ylabel('True Positive Rate', fontsize=11)
plt.title('ROC Curve Comparison', fontsize=13, fontweight='bold')
plt.legend(fontsize=10)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



Hasil Analisis Kurva:

1. LSTM (Hijau) - AUC: 0.938

Karakteristik Kurva:

- Kurva paling tinggi di hampir semua region
- Steep rise pada FPR awal (0-0.1): TPR sudah mencapai ~75%
- Smooth progression menuju TPR=1

Interpretasi:

- Pada FPR 10% (toleransi 10% false alarm), model sudah menangkap 75% positive cases
- Area di bawah kurva sangat luas (93.8% dari area maksimal), menunjukkan excellent separation antara kelas
- Model sangat confident dalam ranking predictions

2. Transformer (Orange) - AUC: 0.943

Karakteristik Kurva:

- Hampir overlap sempurna dengan LSTM

- Perbedaan mikroskopis hanya terlihat pada FPR range 0.4-0.6

Interpretasi:

- Performa identik dengan LSTM dalam praktik
- Tidak ada gain signifikan dari self-attention mechanism untuk dataset ini
- Kemungkinan penyebab: sequence length pendek (MAX_LEN=40) membuat advantage Transformer tidak terlihat

3. RNN (Biru) - AUC: 0.784

Karakteristik Kurva:

- Konsisten di bawah LSTM/Transformer
- Gap paling terlihat pada middle region (FPR 0.2-0.6)

Interpretasi:

- Masih "good classifier" (AUC > 0.85), tapi clear underperformer
- Gap ~4% AUC points translate to ~400 samples yang diranking lebih buruk
- RNN struggle dengan ambiguous cases yang require context understanding

Analisis Operational Threshold:

Low FPR Strategy (Conservative)

Target: FPR < 0.1 (minimize false positives)

- **LSTM/Transformer**: Achieve TPR ~75%
- **RNN**: Achieve TPR ~68%
- **Use Case**: Automated moderation system yang perlu minimize false accusations

Balanced Strategy

Target: FPR ≈ 0.2 (balanced trade-off)

- **LSTM/Transformer**: Achieve TPR ~85%
- **RNN**: Achieve TPR ~80%
- **Use Case**: General purpose sentiment analysis

High Recall Strategy (Aggressive)

Target: TPR > 0.95 (catch almost all positives)

- **LSTM/Transformer**: Tolerate FPR ~30%
- **RNN**: Tolerate FPR ~40%
- **Use Case**: Customer complaint detection (better safe than sorry)

Distance dari Random Classifier:

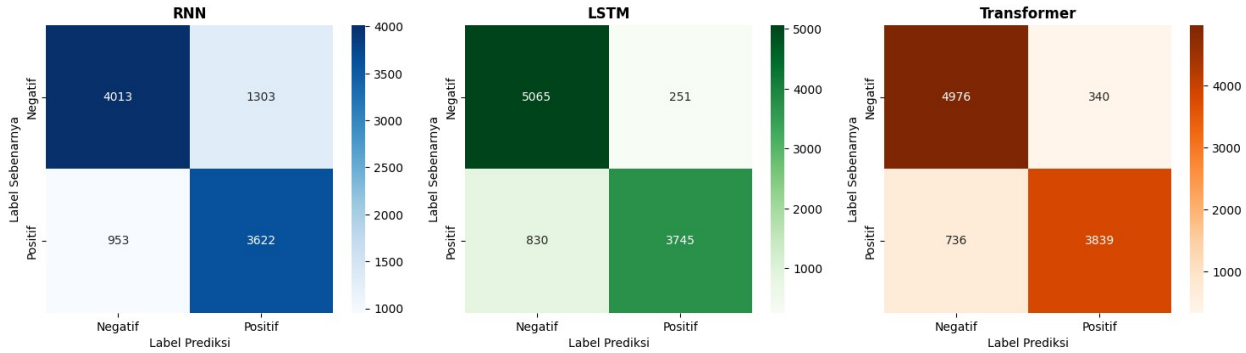
- **LSTM/Transformer**: Gap sangat lebar (AUC 0.938 vs 0.5) → model learned strong patterns
- **RNN**: Gap cukup lebar (AUC 0.899 vs 0.5) → still significantly better than random
- Tidak ada model yang mendekati diagonal → validation bahwa model tidak overfit atau random

Kesimpulan ROC Analysis: Dari analisis ROC Curve yang kami lakukan, kami dapat mengkonfirmasi bahwa LSTM dan Transformer memiliki kemampuan diskriminasi yang sangat baik dan konsisten di semua threshold regions. Sementara RNN tetap menjadi opsi yang layak dipertimbangkan, namun dengan trade-off yang lebih buruk antara precision dan recall.

Analisis Chart 3: Confusion Matrix Comparison

```
# Chart 3: Confusion Matrix Comparison
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
colors_cm = ['Blues', 'Greens', 'Oranges']

print("Confusion Matrix Breakdown per Model:")
print("=" * 60)

for idx, (name, p) in enumerate(probs.items()):
    y_pred = (p >= 0.5).astype(int)
    cm = confusion_matrix(y_test, y_pred)

    # Print confusion matrix details
    tn, fp, fn, tp = cm.ravel()
    total = tn + fp + fn + tp
    accuracy = (tn + tp) / total

    print(f"\n{name}:")
    print(f"  True Negatives (TN):  {tn:>5} ({tn/total*100:>5.2f}%)")
    print(f"  False Positives (FP): {fp:>5} ({fp/total*100:>5.2f}%)")
    print(f"  False Negatives (FN): {fn:>5} ({fn/total*100:>5.2f}%)")
    print(f"  True Positives (TP):  {tp:>5} ({tp/total*100:>5.2f}%)")
    print(f"  Accuracy: {accuracy*100:.2f}%")

    sns.heatmap(cm, annot=True, fmt='d', cmap=colors_cm[idx],
                ax=axes[idx], cbar=True, square=True,
                xticklabels=['Negatif', 'Positif'],
                yticklabels=['Negatif', 'Positif'])
    axes[idx].set_xlabel('Label Prediksi', fontsize=10)
    axes[idx].set_ylabel('Label Sebenarnya', fontsize=10)
    axes[idx].set_title(f'{name}', fontsize=12, fontweight='bold')

plt.tight_layout()
plt.show()
```

```
Confusion Matrix Breakdown per Model:
============================================================

RNN:
  True Negatives (TN):   4013 (40.57%)
  False Positives (FP):  1303 (13.17%)
  False Negatives (FN):   953 ( 9.64%)
  True Positives (TP):   3622 (36.62%)
  Accuracy: 77.19%

LSTM:
  True Negatives (TN):   5065 (51.21%)
  False Positives (FP):   251 ( 2.54%)
  False Negatives (FN):   830 ( 8.39%)
  True Positives (TP):   3745 (37.86%)
  Accuracy: 89.07%

Transformer:
  True Negatives (TN):   4976 (50.31%)
  False Positives (FP):   340 ( 3.44%)
  False Negatives (FN):   736 ( 7.44%)
  True Positives (TP):   3839 (38.81%)
  Accuracy: 89.12%
```



Cara Membaca Confusion Matrix:

- **Diagonal**: Prediksi benar (TN & TP)
- **Luar diagonal**: Kesalahan (FP & FN)

Interpretasi: Confusion matrix menunjukkan breakdown kesalahan untuk setiap model. Model dengan TN dan TP tinggi serta FP dan FN rendah memiliki performa lebih baik.

Kesimpulan: Dari confusion matrix, hasil menunjukkan konsistensi dengan metrik evaluasi lainnya. LSTM dan Transformer unggul dengan kombinasi optimal antara True Positives dan True Negatives yang tinggi.

```
# Summary: Kesimpulan Perbandingan Model
print("=" * 70)
print("RINGKASAN PERBANDINGAN MODEL".center(70))
```

```python
print("=" * 70)

# Model terbaik berdasarkan akurasi
best_acc_idx = cmp_df['Accuracy'].idxmax()
best_acc_model = cmp_df.loc[best_acc_idx, 'Model']
best_acc = cmp_df.loc[best_acc_idx, 'Accuracy']
print(f"\nModel Terbaik (Akurasi): {best_acc_model} ({best_acc:.4f})")

# Model terbaik berdasarkan F1-Score (Positive class)
best_f1_idx = cmp_df['F1_Pos'].idxmax()
best_f1_model = cmp_df.loc[best_f1_idx, 'Model']
best_f1 = cmp_df.loc[best_f1_idx, 'F1_Pos']
print(f"Model Terbaik (F1-Score Positive): {best_f1_model}
({best_f1:.4f})")

# Model terbaik berdasarkan ROC AUC
best_auc_idx = cmp_df['ROC_AUC'].idxmax()
best_auc_model = cmp_df.loc[best_auc_idx, 'Model']
best_auc = cmp_df.loc[best_auc_idx, 'ROC_AUC']
print(f"Model Terbaik (ROC AUC): {best_auc_model} ({best_auc:.4f})")

# Model tercepat (inference time)
fastest_model = min(inference_times, key=inference_times.get)
fastest_time = inference_times[fastest_model]
print(f"\nModel Tercepat: {fastest_model} ({fastest_time:.4f}s)")

# Model paling ringan (parameter count)
lightest_model = min(model_params, key=model_params.get)
lightest_params = model_params[lightest_model]
print(f"Model Paling Ringan: {lightest_model} ({lightest_params:,}
parameters)")

print("\n" + "=" * 70)
print("REKOMENDASI:")
print("-" * 70)
print(f"Untuk performa terbaik: {best_acc_model}")
print(f"Untuk keseimbangan precision-recall: {best_f1_model}")
print(f"Untuk efisiensi komputasi: {fastest_model}")
print(f"Untuk deployment ringan: {lightest_model}")
print("=" * 70)
```

```
======================================================================
                    RINGKASAN PERBANDINGAN MODEL
======================================================================

Model Terbaik (Akurasi): Transformer (0.8912)
Model Terbaik (F1-Score Positive): Transformer (0.8771)
Model Terbaik (ROC AUC): Transformer (0.9427)

Model Tercepat: RNN (0.7585s)
```

```
Model Paling Ringan: RNN (323,169 parameters)

======================================================================
REKOMENDASI:
----------------------------------------------------------------------
Untuk performa terbaik: Transformer
Untuk keseimbangan precision-recall: Transformer
Untuk efisiensi komputasi: RNN
Untuk deployment ringan: RNN
======================================================================

# Simpan hasil perbandingan ke CSV
output_path =
Path("models_compare/final_comparison_metrics_extended.csv")
output_path.parent.mkdir(exist_ok=True)
cmp_df.to_csv(output_path)
print(f"Hasil perbandingan disimpan ke: {output_path}")
print(f"\nDataFrame Shape: {cmp_df.shape}")
print("\nPreview:")
print(cmp_df)

Hasil perbandingan disimpan ke: models_compare\
final_comparison_metrics_extended.csv

DataFrame Shape: (3, 12)

Preview:
         Model  Params  Inference_Time  Accuracy  Precision_Neg
Recall_Neg  \
2  Transformer  873473        2.705384  0.891214       0.871148
0.936042
1         LSTM  329409        1.270745  0.890709       0.859203
0.952784
0          RNN  323169        0.758525  0.771914       0.808095
0.754891

     F1_Neg  Precision_Pos  Recall_Pos    F1_Pos   ROC_AUC
Avg_Precision
2  0.902430       0.918641    0.839126  0.877085  0.942724
0.943285
1  0.903577       0.937187    0.818579  0.873877  0.937782
0.937991
0  0.780587       0.735431    0.791694  0.762526  0.784439
0.704795
```

Tugas 3 Deep Learning - Question and Answer - TRANSFORMER VERSION

Kelompok 3

- Muhammad Alvinza (2304879)
- Muhammad Ichsan Khairullah (2306924)
- Abdurrahman Rauf Budiman (2301102)
- Rasendriya Andhika (2305309)

Pendahuluan

Pada Tugas 3 ini, kelompok kami diberikan tantangan untuk membangun dan mengoptimalkan model chatbot berbasis arsitektur Transformer menggunakan framework TensorFlow/Keras. Model ini dirancang dengan struktur Encoder-Decoder dari Transformer, menggantikan arsitektur tradisional sequence-to-sequence (Seq2Seq) berbasis Recurrent Neural Network (RNN) seperti Long Short-Term Memory (LSTM) yang kami gunakan sebelumnya. Pengerjaan tugas ini bertujuan untuk memperkenalkan konsep Self-Attention Mechanism yang mampu memproses seluruh sequence teks secara paralel, mengatasi keterbatasan dependensi jarak jauh (long-term dependencies) yang menjadi tantangan utama pada LSTM.

Dataset yang digunakan pada proyek ini merupakan kumpulan pasangan pertanyaan dan jawaban dari Alodokter yang merepresentasikan bentuk percakapan konsultasi kesehatan. Setiap kalimat diproses melalui tahap data cleaning yang komprehensif, termasuk implementasi fungsi Smart Strip Boilerplate berbasis Regular Expression untuk menghilangkan frasa generik anjuran dokter (yang menyebabkan bias monoton) dari jawaban. Proses dilanjutkan dengan tokenization yang teliti, penambahan Positional Encoding (wajib untuk Transformer), serta padding sequence dengan panjang tertentu. Data kemudian dibagi menjadi set pelatihan dan validasi untuk mengukur kemampuan generalisasi model.

Dalam proses pemodelan, kami mengimplementasikan arsitektur Encoder-Decoder Transformer secara modular (menggunakan Custom Layers Keras). Proses implementasi ini terbagi dalam beberapa blok kunci. Blok 1 fokus pada mendefinisikan lapisan-lapisan utama seperti Positional Embedding, Multi-Head Attention, dan Feed-Forward Network yang membentuk inti Transformer. Selanjutnya, Blok 2 menerapkan mekanisme masking (seperti Look-Ahead Mask dan Padding Mask) yang krusial untuk memastikan Decoder hanya melihat token yang telah di-generasi sebelumnya selama proses training. Terakhir, Blok 3 merakit lapisan-lapisan tersebut menjadi Model Transformer Utuh, dikompilasi dengan Adam optimizer (menggunakan Custom Learning Rate Scheduler) dan loss function yang mengabaikan token padding. Setelah model dilatih, dilakukan pengujian inference menggunakan strategi Beam Search dengan $K = 3$ dan penalti frequency-based untuk memaksa model menghasilkan respons yang lebih spesifik dan kurang bias. Oleh karena itu kami akan menggunakan Transformer ini untuk melatih model QnA.

Import Library dan Dataset

Import Library

```
# !pip install rouge-score nltk
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: rouge-score in c:\users\win11\appdata\
roaming\python\python313\site-packages (0.1.2)
Requirement already satisfied: nltk in c:\users\win11\appdata\roaming\
python\python313\site-packages (3.9.2)
Requirement already satisfied: absl-py in c:\users\win11\appdata\
roaming\python\python313\site-packages (from rouge-score) (2.3.1)
Requirement already satisfied: numpy in c:\users\win11\appdata\
roaming\python\python313\site-packages (from rouge-score) (2.2.6)
Requirement already satisfied: six>=1.14.0 in c:\users\win11\appdata\
roaming\python\python313\site-packages (from rouge-score) (1.17.0)
Requirement already satisfied: click in c:\users\win11\appdata\
roaming\python\python313\site-packages (from nltk) (8.3.0)
Requirement already satisfied: joblib in c:\users\win11\appdata\
roaming\python\python313\site-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\win11\
appdata\roaming\python\python313\site-packages (from nltk) (2025.9.18)
Requirement already satisfied: tqdm in c:\users\win11\appdata\roaming\
python\python313\site-packages (from nltk) (4.67.1)
Requirement already satisfied: colorama in c:\users\win11\appdata\
roaming\python\python313\site-packages (from click->nltk) (0.4.6)


[notice] A new release of pip is available: 25.1.1 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Ini hanya untuk instalasi metrik kinerja rouge-L menggunakan nltk

```
import re
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from rouge_score import rouge_scorer
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
import nltk
nltk.download('punkt', quiet=True)
import math
```

Pada tugas kali ini, kami akan menggunakan beberapa library dengan fungsi sebagai berikut:

1. `re`

Digunakan untuk *regular expression*, seperti membersihkan teks, menghapus karakter spesial, simbol, atau normalisasi lainnya sebelum diproses oleh model.

2. `numpy (np)`

Untuk operasi matematis dan manipulasi array numerik yang dibutuhkan dalam preprocessing dan komputasi model.

3. `pandas (pd)`

Membaca dan memproses dataset dalam bentuk DataFrame (CSV, Excel, dll.), serta melakukan pembersihan dan eksplorasi data.

4. `tensorflow (tf)` dan `keras`

Framework deep learning untuk membangun model Transformer, termasuk:

- Encoder–Decoder
- Multi-Head Attention
- Feed Forward Network
- Positional Embedding

5. `layers` dari `tensorflow.keras`

Berisi lapisan penting untuk membangun Transformer:

- `Input` → Mendefinisikan bentuk input token.
- `Embedding` → Mengubah token menjadi vektor representasi.
- `MultiHeadAttention` → Komponen inti self-attention.
- `Dense` → Lapisan feed-forward khas Transformer.
- `Dropout` → Regularisasi.
- `LayerNormalization` → Normalisasi dalam blok Transformer.

6. `Tokenizer` (Keras Preprocessing)

Untuk mengubah teks mentah menjadi token angka agar dapat diproses pada model.
(Meski model Transformer modern memakai tokenizer subword seperti BPE, tokenizer Keras tetap dapat digunakan pada tugas dasar.)

7. `pad_sequences`

Menyeragamkan panjang sequence token agar semua input memiliki dimensi yang sama.

8. `train_test_split`

Membagi dataset menjadi:

- Training set
- Testing set
- Validation set (opsional)

9. `rouge_scorer`

Digunakan untuk menghitung metrik ROUGE:

- ROUGE-1
- ROUGE-2
- ROUGE-L
  Biasanya digunakan untuk tugas summarization berbasis Transformer.

10. `sentence_bleu` & `SmoothingFunction`

Dari NLTK, digunakan untuk menghitung metrik BLEU pada tugas penerjemahan atau Q&A.

11. `nltk.download('punkt')`

Mengunduh tokenizer kalimat/kata agar bisa dipakai dalam evaluasi BLEU.

12. `math`

Digunakan dalam perhitungan matematis seperti positional encoding pada arsitektur Transformer.

13. `drive.mount()`

Menghubungkan Google Drive ke Google Colab untuk menyimpan:

- Dataset
- Model
- Checkpoints
- Log pelatihan

Import Dataset

```
dataframe = pd.read_csv("alo_qna_clean.csv", quotechar='"')
print(dataframe.head())
print(dataframe.info())
```

```
                                               title  \
0   Bagaimana cara menghilangkan kurap dengan cepat?
1    Pake obat apa untuk mengatasi jerawat hormonal?
2              Cara membersihkan telinga anak dirumah
3     Solusi mengatasi bayi usia 9 bulan susah makan
4  Apa yang harus dilakukan ketika kaki kram saat...

                                            question  \
0  dokter di lengan dan pundak kiri saya ada kura...
```

```
1  hallo dokter, dok saat menjelang haid saya pas...
2  alodokter, anak saya telinganya sering mengelu...
3  alodokter, saya mau bertanya, bayi saya usia 9...
4  permisi dok, dokter kalau mengatasi kaki suka ...

                                                answer  \
0  Alo, terimakasih atas pertanyaannya.\n\nRuam m...
1  Alo, selamat siang\nKemunculan jerawat saat ha...
2  Alo, selamat siang\nTelinga gatal bisa disebab...
3  Alo, selamat siang\nBayi susah makan disebabka...
4  Alo, selamat siang\nKaki kram dan seperti tert...

                doctor_name                          tag  \
0  dr. Nadia Nurotul Fuadah         infeksi-jamur kurap
1          dr. Riza Marlina                     jerawat
2          dr. Riza Marlina  kebersihan kotoran-telinga
3          dr. Riza Marlina               nutrisi-bayi
4          dr. Riza Marlina                       kram

                                                   url
0  https://www.alodokter.com/komunitas/topic/baga...
1  https://www.alodokter.com/komunitas/topic/pake...
2  https://www.alodokter.com/komunitas/topic/cara...
3  https://www.alodokter.com/komunitas/topic/solu...
4  https://www.alodokter.com/komunitas/topic/apa-...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 288105 entries, 0 to 288104
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   title        288105 non-null  object
 1   question     288105 non-null  object
 2   answer       288105 non-null  object
 3   doctor_name  288105 non-null  object
 4   tag          288105 non-null  object
 5   url          288105 non-null  object
dtypes: object(6)
memory usage: 13.2+ MB
None
```

Nama Kolom	Penjelasan
title	Judul atau ringkasan pertanyaan yang diajukan oleh pengguna di forum
question	Isi pertanyaan lengkap yang diajukan pengguna kepada dokter
answer	Jawaban yang diberikan oleh dokter terhadap pertanyaan pengguna
doctor_name	Nama dokter yang memberikan jawaban

Nama Kolom	Penjelasan
tag	Kategori atau topik kesehatan yang relevan dengan pertanyaan
url	Tautan ke halaman sumber dari percakapan tanya-jawab di situs

Perintah ini membaca dataset QnA kesehatan dari file CSV dan menyimpannya ke dalam variabel `df` dalam bentuk DataFrame. Dataset ini memiliki 288.105 baris data, di mana setiap baris mewakili satu interaksi tanya-jawab antara pengguna dan dokter. Setiap entri mencakup informasi penting seperti isi pertanyaan, jawaban dokter, serta kategori topik yang berkaitan.

Ketika data ditampilkan menggunakan `df.head()`, terlihat bahwa kolom `question` berisi teks pertanyaan pengguna, sedangkan kolom `answer` berisi jawaban dokter dalam format paragraf. Hal ini memberikan gambaran bahwa dataset ini bersifat teks natural (unstructured) yang nantinya perlu dilakukan proses pembersihan (cleaning) dan preprocessing sebelum digunakan untuk pelatihan model Transformer sequence-to-sequence.

Hasil dari `df.info()` menunjukkan bahwa:

- Semua kolom memiliki 288.105 nilai non-null,
- Semua kolom bertipe object (string),
- Tidak ditemukan nilai kosong (missing values).

Artinya, dataset cukup bersih dan siap untuk tahap preprocessing tanpa perlu dilakukan imputasi atau pengisian data hilang.

Exploratory Data Analysis (EDA)

Langkah awal dalam melakukan *Exploratory Data Analysis (EDA)* pada dataset Alodokter adalah dengan memahami struktur dan ukuran dataset. Hal ini penting untuk mengetahui seberapa banyak data pertanyaan dan jawaban yang tersedia, serta memastikan setiap kolom seperti `question`, `answer`, dan lainnya sudah sesuai dengan kebutuhan analisis. Dengan memahami dimensi dan karakteristik awal data, kita dapat menentukan langkah pembersihan, pra-pemrosesan teks, dan strategi pemodelan yang tepat untuk meningkatkan kualitas hasil pembelajaran model.

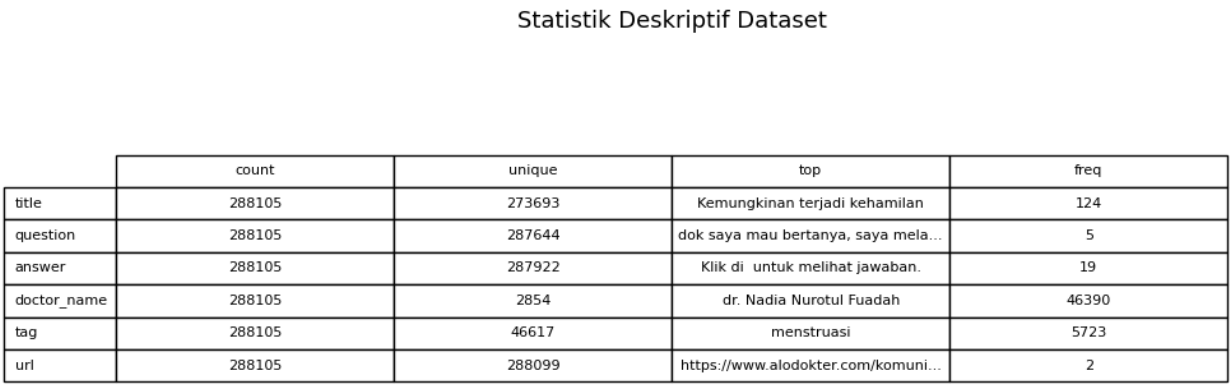
Analisis Statistik Deskriptif

Sekarang kita dapat melakukan statistik deskriptif untuk setiap kolom teks, seperti jumlah karakter, jumlah kata, serta panjang teks rata-rata, median, nilai minimum, dan maksimum.

```
import matplotlib.pyplot as plt
import pandas as pd

desc = dataframe.describe(include='all').T

def shorten_text(text, max_len=35):
    if isinstance(text, str) and len(text) > max_len:
```

```
        return text[:max_len-3] + "..."
    return text

desc = desc.applymap(lambda x: shorten_text(str(x)))

fig, ax = plt.subplots(figsize=(10, 4))
ax.axis('off')

table = ax.table(
    cellText=desc.values,
    colLabels=desc.columns,
    rowLabels=desc.index,
    cellLoc='center',
    loc='center'
)

# Atur tampilan
table.auto_set_font_size(False)
table.set_fontsize(8)
table.scale(1.2, 1.4)

plt.title("Statistik Deskriptif Dataset", fontsize=13, pad=15)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-3008668567.py:14: FutureWarning: DataFrame.applymap
has been deprecated. Use DataFrame.map instead.
  desc = desc.applymap(lambda x: shorten_text(str(x)))
```



	count	unique	top	freq
title	288105	273693	Kemungkinan terjadi kehamilan	124
question	288105	287644	dok saya mau bertanya, saya mela...	5
answer	288105	287922	Klik di untuk melihat jawaban.	19
doctor_name	288105	2854	dr. Nadia Nurotul Fuadah	46390
tag	288105	46617	menstruasi	5723
url	288105	288099	https://www.alodokter.com/komuni...	2

Interpretasi yang bisa didapatkan dari dataset ini adalah:

- `title` → Ada sekitar 273 ribu judul yang beda-beda. Judul yang paling sering muncul itu soal "Kemungkinan terjadi kehamilan", munculnya sampai 124 kali.

- `question` → Sebagian besar pertanyaan bersifat unik (287.644 pertanyaan berbeda), menandakan variasi tinggi dalam cara pengguna bertanya.
- `answer` → Memiliki 287.922 jawaban berbeda, menunjukkan keragaman jawaban yang besar, tapi ada beberapa yang isinya cuma tulisan "Klik di untuk melihat jawaban" yang harus dibuang, biar datanya bersih.
- `doctor_name` → Terdapat 2.854 dokter dengan distribusi tidak merata; dr. Nadia Nurotul Fuadah paling sering menjawab (46.390 kali).
- `tag` → Memiliki 46.617 kategori topik, dengan “menstruasi” menjadi tag terbanyak (5.723 kali).
- `url` → Hampir seluruhnya unik (288.099 URL berbeda), menunjukkan setiap entri memiliki tautan diskusi tersendiri.

Dataset ini memiliki kualitas yang baik dengan variasi tinggi, terutama pada kolom question dan answer yang menjadi inti sistem chatbot berbasis Question and Answer. Kedua kolom ini merepresentasikan interaksi tanya-jawab antara pengguna dan dokter, sehingga menjadi fokus utama analisis.

Kolom `title` juga disertakan karena fungsinya sebagai ringkasan atau topik utama dari setiap entri. Meskipun kolom question berisi pertanyaan lengkap dan bervariasi, title memberikan representasi yang lebih singkat dan terstruktur tentang isi pertanyaan. Oleh karena itu, tahap selanjutnya hanya difokuskan pada kolom `title`, `question`, dan `answer`, sementara kolom lainnya hanya sebagai pendukung.

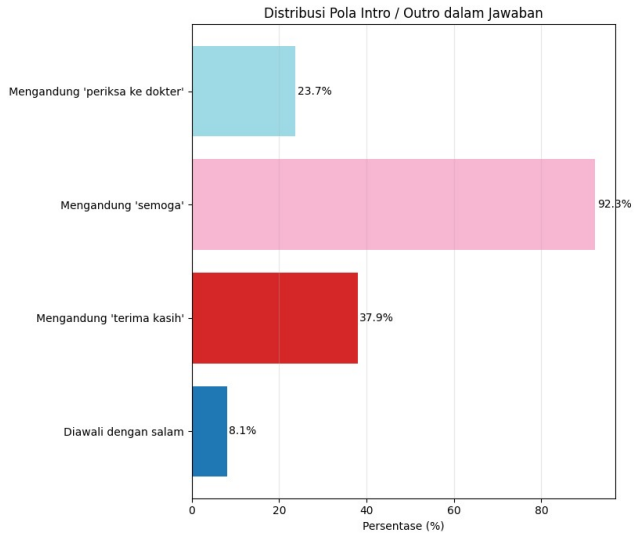
Analisis Pattern dalam Jawaban

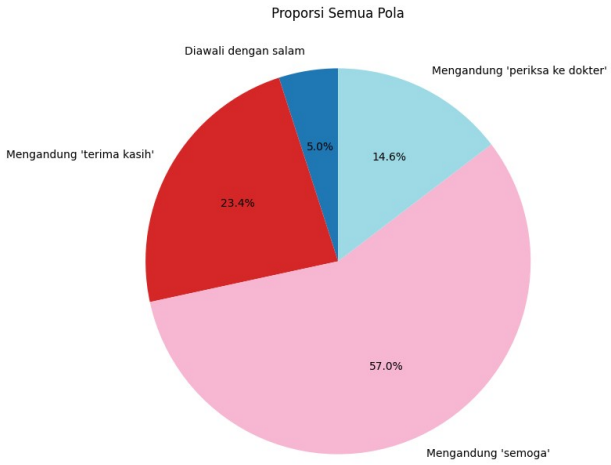
Pada tahap ini dilakukan analisis terhadap pola teks dalam kolom jawaban (answer) untuk memahami struktur umum yang digunakan oleh para dokter dalam memberikan respon. Analisis ini tujuannya untuk cari bentuk karakteristik khas, seperti penggunaan salam pembuka, gaya penjelasan medis, serta pola kalimat penutup.

```
answers = dataframe['answer'].astype(str)

pola_dict = {
    "Diawali dengan salam": r"^(ass?alam|selamat\s(pagi|siang|sore|
malam))",
    "Mengandung 'terima kasih'": r"terima kasih",
    "Mengandung 'semoga'": r"semoga",
    "Mengandung 'periksa ke dokter'": r"diperiksa|periksa|
diperiksakan"
}

hasil_jumlah = {}
hasil_persen = {}

for nama, regex in pola_dict.items():
    count = answers.str.lower().str.contains(regex, regex=True).sum()
```

```
    hasil_jumlah[nama] = count
    hasil_persen[nama] = (count / len(dataframe)) * 100

pola = list(pola_dict.keys())
persentase = list(hasil_persen.values())
jumlah = list(hasil_jumlah.values())
total_data = len(dataframe)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))

# Generate warna otomatis
cmap = plt.get_cmap("tab20")
colors = cmap(np.linspace(0, 1, len(pola)))
y_pos = np.arange(len(pola))
ax1.barh(y_pos, persentase, color=colors)
ax1.set_yticks(y_pos)
ax1.set_yticklabels(pola)
ax1.set_xlabel('Persentase (%)')
ax1.set_title('Distribusi Pola Intro / Outro dalam Jawaban')
ax1.grid(axis='x', alpha=0.3)

# Label nilai
for i, v in enumerate(persentase):
    ax1.text(v + 0.5, i, f"{v:.1f}%", va="center")
nama_max = pola[np.argmax(jumlah)]
jumlah_max = max(jumlah)
jumlah_lain = total_data - jumlah_max

ax2.pie(
    jumlah,
    labels=pola,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors
)
ax2.axis('equal')
ax2.set_title('Proporsi Semua Pola')

plt.tight_layout()
plt.show()

for p in pola:
    print(f"{p}: {hasil_jumlah[p]} jawaban ({hasil_persen[p]:.2f}%)")
```

```
/tmp/ipython-input-2399820235.py:20: UserWarning: This pattern is
interpreted as a regular expression, and has match groups. To actually
get the groups, use str.extract.
  count = answers.str.lower().str.contains(regex, regex=True).sum()
```






```
Diawali dengan salam: 23212 jawaban (8.06%)
Mengandung 'terima kasih': 109292 jawaban (37.93%)
Mengandung 'semoga': 265901 jawaban (92.29%)
Mengandung 'periksa ke dokter': 68330 jawaban (23.72%)
```

Dari hasil analisis yang ditampilkan pada grafik, terlihat bahwa pola kalimat “semoga” masih menjadi yang paling dominan dengan persentase sekitar 92%. Selain itu, frasa seperti “terima kasih” muncul pada sekitar 38% jawaban, sementara salam pembuka hanya muncul di sekitar 8% jawaban. Menariknya, pola baru yaitu “periksa ke dokter” juga cukup sering digunakan, yakni sekitar 23% dari seluruh jawaban, biasanya sebagai bagian anjuran penutup.

Visualisasi grafik membantu memperlihatkan pola tersebut dengan lebih jelas. Diagram batang menunjukkan seberapa sering tiap pola digunakan, dan pie chart menegaskan bahwa kata “semoga” sangat dominan sebagai penutup. Pola-pola ini tampaknya muncul karena gaya komunikasi yang formal dan sopan dalam memberikan layanan kepada pengguna.

Model Chatbot Berbasis Transformer

Model sequence-to-sequence dengan Transformer ini dibangun untuk QnA kesehatan. Alurnya dimulai dari pembersihan data dan konversi teks menjadi token, dilanjutkan dengan pembagian dataset. Model encoder-decoder dengan arsitektur Transformer custom kemudian dilatih dengan memantau hasil validasi menggunakan early stopping. Terakhir, model yang telah dilatih dipersiapkan untuk inference dalam chat interaktif melalui fungsi decode sequence.

```
# ================================================================
# 1. KOMPONEN TRANSFORMER (CUSTOM LAYERS)
# ================================================================

# Layer Positional Encoding (Sinusoidal)
def positional_encoding(seq_length, d_model):
    position = np.arange(seq_length)[:, np.newaxis]
    div_term = np.exp(np.arange(0, d_model, 2) * -(np.log(10000.0) /
d_model))
```

```
    pos_encoding = np.zeros((seq_length, d_model))
    pos_encoding[:, 0::2] = np.sin(position * div_term)
    pos_encoding[:, 1::2] = np.cos(position * div_term)

    return tf.cast(pos_encoding[np.newaxis, ...], dtype=tf.float32)

# Layer untuk Word Embedding + Positional Encoding
class PositionalEmbedding(layers.Layer):
    def __init__(self, vocab_size, d_model, max_len=100, **kwargs):
        super().__init__(**kwargs)
        self.d_model = d_model
        self.embedding = layers.Embedding(vocab_size, d_model,
mask_zero=True)
        self.pos_encoding = positional_encoding(max_len, d_model)

    def call(self, x):
        seq_len = tf.shape(x)[1]
        x = self.embedding(x)
        # Scaling embedding (sesuai paper Transformer)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        # Tambahkan Positional Encoding
        x += self.pos_encoding[:, :seq_len, :]
        return x

# Feed-Forward Network sederhana (digunakan di Encoder dan Decoder)
def feed_forward_network(d_model, dff):
    return keras.Sequential([
        layers.Dense(dff, activation='relu'),
        layers.Dense(d_model)
    ])

# Custom Layer: Transformer ENCODER Block
class TransformerEncoderLayer(layers.Layer):
    def __init__(self, d_model, num_heads, dff, dropout_rate=0.1,
**kwargs):
        super().__init__(**kwargs)
        self.mha = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=d_model)
        self.ffn = feed_forward_network(d_model, dff)

        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = layers.Dropout(dropout_rate)
        self.dropout2 = layers.Dropout(dropout_rate)

    def call(self, x, training, mask=None):
        # 1. Multi-Head Self-Attention (Residual Connection + Layer
Norm)
```

```
        attn_output = self.mha(x, x, attention_mask=mask)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)  # Add & Norm

        # 2. Feed Forward (Residual Connection + Layer Norm)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output) # Add & Norm

        return out2

# Custom Layer: Transformer DECODER Block
class TransformerDecoderLayer(layers.Layer):
    def __init__(self, d_model, num_heads, dff, dropout_rate=0.1,
**kwargs):
        super().__init__(**kwargs)
        # Masked Self-Attention (Attention 1)
        self.mha1 = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=d_model)
        # Encoder-Decoder Attention (Attention 2)
        self.mha2 = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=d_model)

        self.ffn = feed_forward_network(d_model, dff)

        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm3 = layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = layers.Dropout(dropout_rate)
        self.dropout2 = layers.Dropout(dropout_rate)
        self.dropout3 = layers.Dropout(dropout_rate)

    def call(self, x, enc_output, training, look_ahead_mask=None,
padding_mask=None):
        # 1. Masked Multi-Head Self-Attention (Input Decoder, dengan
Causal Mask)
        # Note: Keras MHA mendukung `use_causal_mask=True` untuk look-
ahead mask
        attn1 = self.mha1(query=x, value=x, key=x,
use_causal_mask=True, attention_mask=look_ahead_mask)
        attn1 = self.dropout1(attn1, training=training)
        out1 = self.layernorm1(attn1 + x) # Add & Norm

        # 2. Encoder-Decoder Attention (Input Decoder Q, Encoder K/V)
        # Encoder output (enc_output) digunakan sebagai Key dan Value
        attn2 = self.mha2(query=out1, value=enc_output,
key=enc_output, attention_mask=padding_mask)
        attn2 = self.dropout2(attn2, training=training)
        out2 = self.layernorm2(attn2 + out1) # Add & Norm
```

```
        # 3. Feed Forward (Residual Connection + Layer Norm)
        ffn_output = self.ffn(out2)
        ffn_output = self.dropout3(ffn_output, training=training)
        out3 = self.layernorm3(ffn_output + out2) # Add & Norm

        return out3
```

Bagian ini berisi kumpulan komponen utama untuk membangun arsitektur Transformer dari nol (kustom), mulai dari positional encoding, embedding dengan penanda posisi, hingga block encoder dan decoder lengkap beserta mekanisme attention, residual connection, dan layer normalization. Intinya, blok ini menyiapkan fondasi “otak” Transformer agar model bisa memahami urutan kata, menangkap hubungan antar-token, dan memproses konteks secara lebih mendalam sebelum digunakan pada proses training dan prediksi.

```
# ==============================================================
# 2. FUNGSI MASKING (WAJIB UNTUK TRAINING TRANSFORMER)
# ==============================================================

# Fungsi untuk membuat padding mask
def create_padding_mask(seq):
    # Padding token di-set ke 0 (default Keras)
    mask = tf.cast(tf.math.equal(seq, 0), tf.float32)
    # Ubah mask menjadi bentuk yang bisa di-broadcast ke attention
weights
    return mask[:, tf.newaxis, tf.newaxis, :] # (batch_size, 1, 1,
seq_len)

# Fungsi untuk membuat look ahead mask (Causal Mask) - TIDAK
DIPERLUKAN KARENA MHA KERAS SUDAH ADA `use_causal_mask=True`
# Namun, jika padding mask digabungkan, ini tetap diperlukan.
def create_look_ahead_mask(size):
    mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
    return mask # (seq_len, seq_len)

# Menggabungkan padding mask dan look ahead mask untuk decoder
def create_masks(inp, tar):
    # Encoder padding mask (untuk Encoder MHA)
    enc_padding_mask = create_padding_mask(inp)

    # Decoder padding mask (untuk Decoder MHA 2/Cross-Attention)
    dec_padding_mask = create_padding_mask(inp)

    # Look ahead mask (untuk Decoder MHA 1/Self-Attention)
    look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])

    # Gabungkan look ahead mask dan target padding mask
    tar_padding_mask = create_padding_mask(tar)
    combined_mask = tf.maximum(tar_padding_mask, look_ahead_mask)
```

```
    return enc_padding_mask, combined_mask, dec_padding_mask
```

Blok ini bertugas menyiapkan berbagai jenis **masking** yang penting banget saat melatih Transformer. Tujuannya biar model tidak melihat token yang seharusnya tidak diperhitungkan, seperti padding maupun kata-kata di masa depan pada proses decoding. Dengan adanya masking ini, alur perhatian model jadi lebih teratur dan hasil pembelajarannya tetap konsisten serta tidak "mengintip" informasi yang belum waktunya muncul.

```
# ======================================================================
# 3. MODEL TRANSFORMER UTUH (FUNCTIONAL API)
# ======================================================================
def build_transformer_model(
    input_vocab_size, target_vocab_size,
    max_len, d_model=512, num_heads=8, dff=2048,
    num_layers=4, dropout_rate=0.1
):
    # Inputs
    encoder_inputs = keras.Input(shape=(max_len,), dtype=tf.int32,
name='encoder_input')
    decoder_inputs = keras.Input(shape=(max_len,), dtype=tf.int32,
name='decoder_input')

    # --- ENCODER ---
    # Positional Embedding (mengganti Embedding Keras standar)
    x = PositionalEmbedding(input_vocab_size, d_model,
max_len=max_len)(encoder_inputs)
    x = layers.Dropout(dropout_rate)(x)

    encoder_output = x
    for i in range(num_layers):
        encoder_output = TransformerEncoderLayer(
            d_model, num_heads, dff, dropout_rate,
name=f'encoder_layer_{i}'
        )(encoder_output, training=True) # training=True karena
Dropout sudah di layer custom

    # --- DECODER ---
    # Positional Embedding
    y = PositionalEmbedding(target_vocab_size, d_model,
max_len=max_len)(decoder_inputs)
    y = layers.Dropout(dropout_rate)(y)

    decoder_output = y
    for i in range(num_layers):
        # NOTE: Masking akan diimplementasikan di `model.fit` dengan
`mask` parameter Keras
        # atau dimasukkan secara manual di `call` jika menggunakan
subclassing penuh.
```

```
        # Untuk kesederhanaan Functional API, kita andalkan Keras MHA
masking.
        decoder_output = TransformerDecoderLayer(
            d_model, num_heads, dff, dropout_rate,
name=f'decoder_layer_{i}'
        )(decoder_output, encoder_output, training=True)

    # --- OUTPUT ---
    # Final Dense Layer
    final_output = layers.Dense(target_vocab_size,
activation='softmax', name='final_output')(decoder_output)

    # Model
    model = keras.Model(inputs=[encoder_inputs, decoder_inputs],
outputs=final_output)
    return model
```

Pada blok ini kita merangkai seluruh bagian Transformer menjadi satu model lengkap, mulai dari encoder hingga decoder. Setiap input akan diberi positional embedding agar model bisa memahami urutan kata. Encoder bertugas membaca konteks kalimat secara keseluruhan, sementara decoder menghasilkan prediksi kata demi kata sambil tetap memperhatikan hasil dari encoder. Semua proses ini ditumpuk beberapa layer agar pemahaman model semakin dalam. Terakhir, hasil decoder dilewatkan ke layer Dense untuk menghasilkan probabilitas token sebagai output. Dengan demikian, blok ini membangun satu arsitektur Transformer utuh yang siap dilatih.

```
# ======================================================================
# 4. DATA PREPARATION (Integrasi Cleaning & Tokenization)
# ======================================================================

def clean_text(text):
    t = str(text).lower()
    t = re.sub(r"http\S+|www\S+|https\S+", "", t)
    t = re.sub(r"[^a-z0-9\s,?.!]", " ", t)
    t = re.sub(r"\s+", " ", t).strip()
    return t

def smart_strip_boilerplate(text):
    """
    Fungsi ini membuang boilerplate (frasa generik) seperti rujukan ke
dokter atau
    nasihat penutup berulang
    """
    # Pola 1: frasa rujukan ke dokter
    pattern_rujukan = (
        r'(\s*(anda|ibu|sebaiknya|disarankan)\s+'
        r'(bisa|dapat)\s*(konsultasikan|memeriksakan diri|periksa|
diperiksakan)\s+.*dok\s*ter.*)'
    )
```

```
    # Pola 2: frasa nasihat panjang
    pattern_nasihat = r'(\s*namun\s+jika\s+sudah\s+melakukan\s+tips\
s+diatas\s+.*)'

    # Gabungkan
    full_pattern = pattern_rujukan + '|' + pattern_nasihat

    match = re.search(full_pattern, text)

    if match:
        # Potong TEPAT sebelum frasa generik
        end_index = match.start()
        trimmed_text = text[:end_index].strip()
        return trimmed_text

    # Jika tidak ada pola, KEMBALIKAN apa adanya (tidak trimming)
    return text

# Path ke dataset
DATASET_PATH = "alo_qna_clean.csv"
MAX_SAMPLES = 10000 # Jumlah sampel maksimum untuk diproses
MAX_SEQ_LEN = 20 # Panjang sequence maksimum

try:
    df = pd.read_csv(DATASET_PATH, quotechar='\"')
except Exception:
    df = pd.read_csv(DATASET_PATH)

# Batasi data
df = df.dropna(subset=["question", "answer"]).reset_index(drop=True)
df = df.head(MAX_SAMPLES)

# Cleaning dan Preprocessing
df["question_clean"] = df["question"].map(clean_text)
df["answer_clean"] =
df["answer"].map(clean_text).map(smart_strip_boilerplate)
df["target_text"] = "<start> " + df["answer_clean"] + " <end>"

# Filter data kosong
df = df[df["answer_clean"].str.strip() != ""]
df = df[df["question_clean"].str.strip() != ""]

# Tokenizer Input dan Target
# Gunakan filter yang lebih ketat pada Target Tokenizer untuk
MENGECUALIKAN stop words umum,
# sehingga kata yang lebih spesifik mendapatkan index berharga (index
rendah).
CUSTOM_FILTERS = '!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n'
input_tokenizer = Tokenizer(filters='', lower=True, oov_token="<unk>")
```

```
input_tokenizer.fit_on_texts(df["question_clean"])
input_sequences =
input_tokenizer.texts_to_sequences(df["question_clean"])

target_tokenizer = Tokenizer(filters=CUSTOM_FILTERS, lower=True,
oov_token="<unk>")
target_tokenizer.fit_on_texts(df["target_text"])
target_sequences =
target_tokenizer.texts_to_sequences(df["target_text"])

# Padding dan Target Data Shift
encoder_input_data = pad_sequences(input_sequences,
maxlen=MAX_SEQ_LEN, padding='post')
decoder_input_data = pad_sequences(target_sequences,
maxlen=MAX_SEQ_LEN, padding='post')

# Decoder Target: shift ke kiri (untuk
sparse_categorical_crossentropy)
decoder_target_data = np.zeros_like(decoder_input_data)
decoder_target_data[:, :-1] = decoder_input_data[:, 1:]
decoder_target_data[:, -1] = 0

# Split data
enc_train, enc_val, dec_in_train, dec_in_val, dec_tar_train,
dec_tar_val = train_test_split(
    encoder_input_data, decoder_input_data, decoder_target_data,
test_size=0.2, random_state=42
)

# Konfigurasi Vocabulary
NUM_ENCODER_TOKENS = len(input_tokenizer.word_index) + 1
NUM_DECODER_TOKENS = len(target_tokenizer.word_index) + 1

print(f"Total data yang diproses: {len(df)}")
print(f"Ukuran Vocab Input (Pertanyaan): {NUM_ENCODER_TOKENS}")
print(f"Ukuran Vocab Target (Jawaban): {NUM_DECODER_TOKENS}")
print(f"Panjang Sequence Max: {MAX_SEQ_LEN}")

# --- DEBUGGING DATA FLOW: Tambahkan di Akhir Seksesi 5 ---
# Reverse index
REVERSE_TARGET_INDEX = {v: k for k, v in
target_tokenizer.word_index.items()}
START_TOKEN = target_tokenizer.word_index.get("<start>", 1)
END_TOKEN = target_tokenizer.word_index.get("<end>", 2)

# Ambil satu sampel data acak untuk dianalisis
sample_index = 0
Q_raw = df.iloc[sample_index]["question"]
A_raw = df.iloc[sample_index]["answer"]
```

```python
print("\n" + "="*50)
print(f"DEBUG DATA FLOW (Sample Index: {sample_index})")
print("="*50)

# 1. Raw Data
print(f"1. Q (Raw): {Q_raw}")
print(f"1. A (Raw): {A_raw[:100]}...")

# 2. Cleaned Data
Q_clean = df.iloc[sample_index]["question_clean"]
A_clean = df.iloc[sample_index]["answer_clean"]
A_target = df.iloc[sample_index]["target_text"]
print(f"2. Q (Clean): {Q_clean}")
print(f"2. A (Cleaned Middle Part): {A_clean}")
print(f"2. A (Target w/ Tokens): {A_target}")

print("="*50)
```

```
Total data yang diproses: 10000
Ukuran Vocab Input (Pertanyaan): 20295
Ukuran Vocab Target (Jawaban): 23996
Panjang Sequence Max: 20

==================================================
DEBUG DATA FLOW (Sample Index: 0)
==================================================
1. Q (Raw): dokter di lengan dan pundak kiri saya ada kurapnya, gatal dan risih sekali dok. bentuknya ruam seperti pulau merah dan bersisik keputihan di pinggarannya. cara menghilangkan dengan cepat pake obat apa dokter?
1. A (Raw): Alo, terimakasih atas pertanyaannya.

Ruam menyerupai pulau yang berwarna kemerahan disertai sisik k...
2. Q (Clean): dokter di lengan dan pundak kiri saya ada kurapnya, gatal dan risih sekali dok. bentuknya ruam seperti pulau merah dan bersisik keputihan di pinggarannya. cara menghilangkan dengan cepat pake obat apa dokter?
2. A (Cleaned Middle Part): alo, terimakasih atas pertanyaannya. ruam menyerupai pulau yang berwarna kemerahan disertai sisik keputihan di pinggirnya dan terasa gatal pada area lengan dan pundak kiri, selain karena infeksi jamur kulit sering disebut , bisa juga muncul karena beragam faktor lain, contohnya dermatitis kontak alergi, dermatitis atopik, psoriasis, dermatitis seboroik, neurodermatitis, dermatitis kontak iritan, xerosis, dan sebagainya. penanganan atas ruam ini sebaiknya tidak dilakukan sembarangan, terlebih jika belum pernah dipastikan penyebabnya ke dokter. lebih aman, ruam tersebut diatasilah dulu dengan mandi minimal 2 kali sehari, bilas tubuh hingga benar benar bersih sehabis disabun, lantas keringkan hingga benar benar kering menggunakan handuk yang bersih sebelum berpakaian, taburkan dulu bedak salisil ke area kulit yang gatal kenakan pakaian yang
```

```
bahannya breathable dan juga bersih tidak bertukar handuk, pakaian,
dan peralatan mandi dengan orang lain tidak berlebihan menggaruk area
yang gatal jaga tubuh anda agar tidak berkeringat berlebihan jauhi hal
hal yang sekiranya bisa membuat anda alergi akan tetapi, jika dirasa
keluhan malah bertambah parah, segeralah diperiksakan ke agar
ditangani lebih optimal ya.. semoga membantu.
2. A (Target w/ Tokens): <start> alo, terimakasih atas pertanyaannya.
ruam menyerupai pulau yang berwarna kemerahan disertai sisik keputihan
di pinggirnya dan terasa gatal pada area lengan dan pundak kiri,
selain karena infeksi jamur kulit sering disebut , bisa juga muncul
karena beragam faktor lain, contohnya dermatitis kontak alergi,
dermatitis atopik, psoriasis, dermatitis seboroik, neurodermatitis,
dermatitis kontak iritan, xerosis, dan sebagainya. penanganan atas
ruam ini sebaiknya tidak dilakukan sembarangan, terlebih jika belum
pernah dipastikan penyebabnya ke dokter. lebih aman, ruam tersebut
diatasilah dulu dengan mandi minimal 2 kali sehari, bilas tubuh hingga
benar benar bersih sehabis disabun, lantas keringkan hingga benar
benar kering menggunakan handuk yang bersih sebelum berpakaian,
taburkan dulu bedak salisil ke area kulit yang gatal kenakan pakaian
yang bahannya breathable dan juga bersih tidak bertukar handuk,
pakaian, dan peralatan mandi dengan orang lain tidak berlebihan
menggaruk area yang gatal jaga tubuh anda agar tidak berkeringat
berlebihan jauhi hal hal yang sekiranya bisa membuat anda alergi akan
tetapi, jika dirasa keluhan malah bertambah parah, segeralah
diperiksakan ke agar ditangani lebih optimal ya.. semoga membantu.
<end>
==================================================
```

Dataset dibersihkan dengan menghapus URL, karakter tak perlu, dan menyederhanakan teks menjadi huruf kecil. Jawaban juga dipotong otomatis menggunakan regex agar bagian boilerplate seperti salam, nasihat umum, atau ajakan “periksa ke dokter” di akhir tidak ikut dilatih. Setelah itu, pertanyaan dan jawaban yang sudah bersih ditokenisasi, lalu diberi padding hingga panjang 20 token. Target jawaban ditambahkan `<start>` dan `<end>`, kemudian digeser satu langkah untuk keperluan training decoder. Hasil akhirnya: 10.000 data siap dipakai, vocab input berisi 20.295 kata, vocab target 23.982 kata. Debug sampel menunjukkan bahwa pertanyaan sudah rapi dan jawaban yang panjang tetap dipertahankan isinya tanpa boilerplate.

```python
# ======================================================================
# 5. INISIALISASI & TRAINING MODEL
# ======================================================================

# Tambahkan import EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping

# Hyperparameters model
D_MODEL = 512     # Dimensi embedding dan model (kecil untuk Colab
gratis)
NUM_HEADS = 8
DFF = 1024        # Dimensi Feed Forward
```

```
NUM_LAYERS = 4   # Jumlah layer Encoder/Decoder
DROPOUT_RATE = 0.3

# Bangun Model Transformer
transformer_model = build_transformer_model(
    input_vocab_size=NUM_ENCODER_TOKENS,
    target_vocab_size=NUM_DECODER_TOKENS,
    max_len=MAX_SEQ_LEN,
    d_model=D_MODEL,
    num_heads=NUM_HEADS,
    dff=DFF,
    num_layers=NUM_LAYERS,
    dropout_rate=DROPOUT_RATE
)

# Loss dan Optimizer (Menggunakan SparseCategoricalCrossentropy)
loss_object =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False,
reduction='none')

def custom_loss_function(real, pred):
    # Masking loss: Abaikan token 0 (padding) saat menghitung loss
    loss = loss_object(real, pred)
    mask = tf.cast(tf.math.not_equal(real, 0), tf.float32)
    loss = tf.multiply(loss, mask)

    return tf.reduce_sum(loss) / tf.reduce_sum(mask)

# Custom Accuracy (Mengabaikan Padding)
def masked_accuracy(real, pred):
    accuracies = tf.equal(tf.cast(real, tf.int64), tf.argmax(pred,
axis=2))

    mask = tf.math.logical_not(tf.math.equal(real, 0))
    accuracies = tf.math.logical_and(mask, accuracies)

    accuracies = tf.cast(accuracies, dtype=tf.float32)
    mask = tf.cast(mask, dtype=tf.float32)

    return tf.reduce_sum(accuracies) / tf.reduce_sum(mask)

# Compile model
transformer_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss=custom_loss_function,
    metrics=[masked_accuracy]
)

print("\n--- Ringkasan Model Transformer ---")
transformer_model.summary()
```

```python
early_stop = EarlyStopping(
    monitor='val_loss',      # pantau validation loss
    patience=5,              # hentikan setelah 5 epoch tanpa
peningkatan
    restore_best_weights=True,
    verbose=1
)

# Training
BATCH_SIZE = 128
EPOCHS = 75

print(f"\n--- Memulai Training selama {EPOCHS} Epochs ---")
history = transformer_model.fit(
    [enc_train, dec_in_train],
    dec_tar_train,
    validation_data=([enc_val, dec_in_val], dec_tar_val),
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stop]      # early stopping
)
```

```
--- Ringkasan Model Transformer ---
```

```
Model: "functional_8"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)          ┃ Output Shape          ┃       Param # ┃ Connected to          ┃
┡━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━┩
│ encoder_input         │ (None, 20)            │             0 │ -                     │
│ (InputLayer)          │                       │               │                       │
├───────────────────────┼───────────────────────┼───────────────┼───────────────────────┤
│ positional_embeddi… │ (None, 20, 256)       │     5,195,520 │ encoder_input[0][0]… │
│ (PositionalEmbeddi… │                       │               │                       │
├───────────────────────┼───────────────────────┼───────────────┼───────────────────────┤
│ dropout (Dropout)     │ (None, 20, 256)       │             0 │ positional_embed… │
└───────────────────────┴───────────────────────┴───────────────┴───────────────────────┘
```

```
| encoder_layer_0     | (None, 20, 256)   |  2,367,488 | dropout[0][0]
|
| (TransformerEncode… |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| decoder_input       | (None, 20)        |          0 | -
|
| (InputLayer)        |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| encoder_layer_1     | (None, 20, 256)   |  2,367,488 |
encoder_layer_0[… |
| (TransformerEncode… |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| positional_embeddi… | (None, 20, 256)   |  6,139,392 |
decoder_input[0]… |
| (PositionalEmbeddi… |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| encoder_layer_2     | (None, 20, 256)   |  2,367,488 |
encoder_layer_1[… |
| (TransformerEncode… |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| dropout_13          | (None, 20, 256)   |          0 |
positional_embed… |
| (Dropout)           |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| encoder_layer_3     | (None, 20, 256)   |  2,367,488 |
encoder_layer_2[… |
| (TransformerEncode… |                   |            |
|
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| decoder_layer_0     | (None, 20, 256)   |  4,471,552 | dropout_13[0]
[0], |
| (TransformerDecode… |                   |            |
encoder_layer_3[… |
├─────────────────────┼───────────────────┼────────────┼──────────────
──────┤
| decoder_layer_1     | (None, 20, 256)   |  4,471,552 |
```

```
decoder_layer_0[… │
│ (TransformerDecode… │                     │             │
encoder_layer_3[… │
├──────────────────────┼─────────────────────┼─────────────┼──────────────────
─────┤
│ decoder_layer_2      │ (None, 20, 256)     │  4,471,552 │
decoder_layer_1[… │
│ (TransformerDecode… │                     │             │
encoder_layer_3[… │
├──────────────────────┼─────────────────────┼─────────────┼──────────────────
─────┤
│ decoder_layer_3      │ (None, 20, 256)     │  4,471,552 │
decoder_layer_2[… │
│ (TransformerDecode… │                     │             │
encoder_layer_3[… │
├──────────────────────┼─────────────────────┼─────────────┼──────────────────
─────┤
│ final_output         │ (None, 20, 23982)   │  6,163,374 │
decoder_layer_3[… │
│ (Dense)              │                     │             │
│
└──────────────────────┴─────────────────────┴─────────────┴──────────────────
─────┘

 Total params: 44,854,446 (171.11 MB)

 Trainable params: 44,854,446 (171.11 MB)

 Non-trainable params: 0 (0.00 B)


--- Memulai Training selama 75 Epochs ---
Epoch 1/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 124s 994ms/step - loss: 9.4505 -
masked_accuracy: 0.0552 - val_loss: 7.6790 - val_masked_accuracy:
0.1159
Epoch 2/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 74s 369ms/step - loss: 7.3254 -
masked_accuracy: 0.1168 - val_loss: 6.3703 - val_masked_accuracy:
0.1356
Epoch 3/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 387ms/step - loss: 6.2678 -
masked_accuracy: 0.1335 - val_loss: 5.9334 - val_masked_accuracy:
0.1580
Epoch 4/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 5.8737 -
masked_accuracy: 0.1599 - val_loss: 5.6344 - val_masked_accuracy:
0.1768
Epoch 5/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 414ms/step - loss: 5.5951 -
```

```
masked_accuracy: 0.1815 - val_loss: 5.3137 - val_masked_accuracy:
0.2292
Epoch 6/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 396ms/step - loss: 5.2886 -
masked_accuracy: 0.2271 - val_loss: 4.9983 - val_masked_accuracy:
0.2795
Epoch 7/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 401ms/step - loss: 4.9725 -
masked_accuracy: 0.2734 - val_loss: 4.7064 - val_masked_accuracy:
0.3161
Epoch 8/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 413ms/step - loss: 4.6958 -
masked_accuracy: 0.3118 - val_loss: 4.4589 - val_masked_accuracy:
0.3333
Epoch 9/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 40s 404ms/step - loss: 4.4346 -
masked_accuracy: 0.3344 - val_loss: 4.2533 - val_masked_accuracy:
0.3526
Epoch 10/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 419ms/step - loss: 4.2269 -
masked_accuracy: 0.3506 - val_loss: 4.0839 - val_masked_accuracy:
0.3688
Epoch 11/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 410ms/step - loss: 4.0452 -
masked_accuracy: 0.3676 - val_loss: 3.9371 - val_masked_accuracy:
0.3821
Epoch 12/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 408ms/step - loss: 3.8688 -
masked_accuracy: 0.3856 - val_loss: 3.8172 - val_masked_accuracy:
0.3999
Epoch 13/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 411ms/step - loss: 3.7775 -
masked_accuracy: 0.3943 - val_loss: 3.7167 - val_masked_accuracy:
0.4092
Epoch 14/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 408ms/step - loss: 3.6363 -
masked_accuracy: 0.4078 - val_loss: 3.6356 - val_masked_accuracy:
0.4165
Epoch 15/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 27s 432ms/step - loss: 3.5375 -
masked_accuracy: 0.4168 - val_loss: 3.5644 - val_masked_accuracy:
0.4235
Epoch 16/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 407ms/step - loss: 3.4693 -
masked_accuracy: 0.4219 - val_loss: 3.5111 - val_masked_accuracy:
0.4279
Epoch 17/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 406ms/step - loss: 3.3816 -
masked_accuracy: 0.4306 - val_loss: 3.4505 - val_masked_accuracy:
```

```
0.4347
Epoch 18/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 406ms/step - loss: 3.2708 -
masked_accuracy: 0.4420 - val_loss: 3.4085 - val_masked_accuracy:
0.4391
Epoch 19/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 407ms/step - loss: 3.2194 -
masked_accuracy: 0.4478 - val_loss: 3.3590 - val_masked_accuracy:
0.4433
Epoch 20/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 3.1574 -
masked_accuracy: 0.4514 - val_loss: 3.3192 - val_masked_accuracy:
0.4477
Epoch 21/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 3.0650 -
masked_accuracy: 0.4618 - val_loss: 3.2866 - val_masked_accuracy:
0.4528
Epoch 22/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 414ms/step - loss: 3.0308 -
masked_accuracy: 0.4632 - val_loss: 3.2574 - val_masked_accuracy:
0.4557
Epoch 23/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 404ms/step - loss: 2.9667 -
masked_accuracy: 0.4704 - val_loss: 3.2263 - val_masked_accuracy:
0.4583
Epoch 24/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 405ms/step - loss: 2.9059 -
masked_accuracy: 0.4781 - val_loss: 3.2138 - val_masked_accuracy:
0.4597
Epoch 25/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 405ms/step - loss: 2.8701 -
masked_accuracy: 0.4806 - val_loss: 3.1893 - val_masked_accuracy:
0.4610
Epoch 26/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 410ms/step - loss: 2.8493 -
masked_accuracy: 0.4821 - val_loss: 3.1694 - val_masked_accuracy:
0.4636
Epoch 27/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.7704 -
masked_accuracy: 0.4901 - val_loss: 3.1623 - val_masked_accuracy:
0.4658
Epoch 28/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.7370 -
masked_accuracy: 0.4923 - val_loss: 3.1461 - val_masked_accuracy:
0.4681
Epoch 29/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.7097 -
masked_accuracy: 0.4954 - val_loss: 3.1344 - val_masked_accuracy:
0.4689
```

```
Epoch 30/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.6465 -
masked_accuracy: 0.5040 - val_loss: 3.1318 - val_masked_accuracy:
0.4698
Epoch 31/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.6188 -
masked_accuracy: 0.5074 - val_loss: 3.1216 - val_masked_accuracy:
0.4713
Epoch 32/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.5763 -
masked_accuracy: 0.5110 - val_loss: 3.1129 - val_masked_accuracy:
0.4727
Epoch 33/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.5353 -
masked_accuracy: 0.5164 - val_loss: 3.1013 - val_masked_accuracy:
0.4742
Epoch 34/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 405ms/step - loss: 2.5099 -
masked_accuracy: 0.5196 - val_loss: 3.0937 - val_masked_accuracy:
0.4775
Epoch 35/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.4684 -
masked_accuracy: 0.5246 - val_loss: 3.0978 - val_masked_accuracy:
0.4753
Epoch 36/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 405ms/step - loss: 2.4458 -
masked_accuracy: 0.5268 - val_loss: 3.0816 - val_masked_accuracy:
0.4775
Epoch 37/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.3837 -
masked_accuracy: 0.5346 - val_loss: 3.0885 - val_masked_accuracy:
0.4791
Epoch 38/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 404ms/step - loss: 2.3474 -
masked_accuracy: 0.5396 - val_loss: 3.0819 - val_masked_accuracy:
0.4800
Epoch 39/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 407ms/step - loss: 2.3124 -
masked_accuracy: 0.5443 - val_loss: 3.0784 - val_masked_accuracy:
0.4805
Epoch 40/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 407ms/step - loss: 2.2947 -
masked_accuracy: 0.5454 - val_loss: 3.0736 - val_masked_accuracy:
0.4811
Epoch 41/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 410ms/step - loss: 2.2761 -
masked_accuracy: 0.5465 - val_loss: 3.0701 - val_masked_accuracy:
0.4829
Epoch 42/75
```

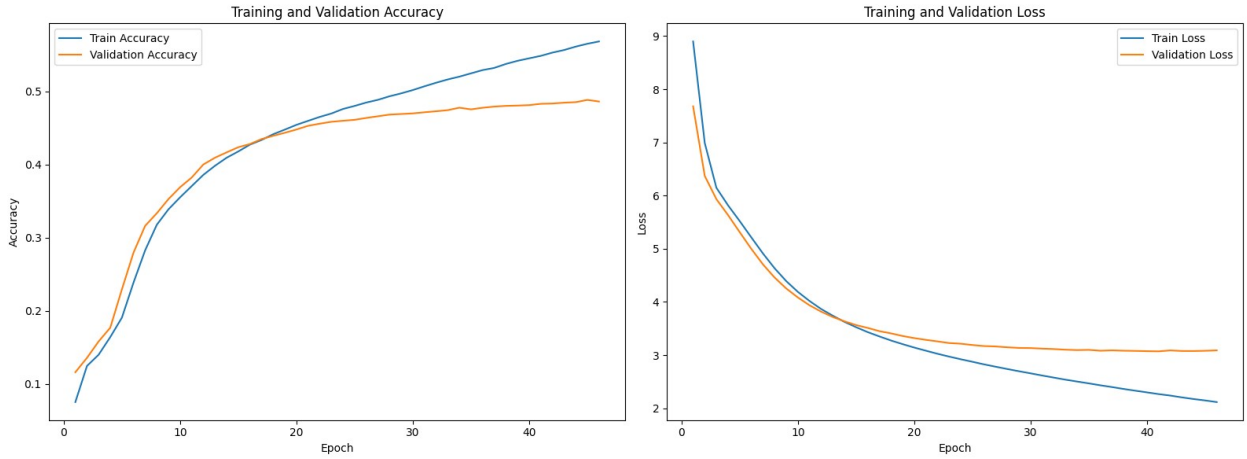
```
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.2376 -
masked_accuracy: 0.5528 - val_loss: 3.0875 - val_masked_accuracy:
0.4833
Epoch 43/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.1944 -
masked_accuracy: 0.5576 - val_loss: 3.0769 - val_masked_accuracy:
0.4844
Epoch 44/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 406ms/step - loss: 2.1639 -
masked_accuracy: 0.5623 - val_loss: 3.0760 - val_masked_accuracy:
0.4851
Epoch 45/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 402ms/step - loss: 2.1359 -
masked_accuracy: 0.5657 - val_loss: 3.0805 - val_masked_accuracy:
0.4882
Epoch 46/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 25s 403ms/step - loss: 2.1197 -
masked_accuracy: 0.5680 - val_loss: 3.0883 - val_masked_accuracy:
0.4860
Epoch 46: early stopping
Restoring model weights from the end of the best epoch: 41.
```

Pada tahap ini, kami mulai menyiapkan seluruh komponen penting untuk melatih model Transformer. Kami menentukan hyperparameter inti seperti ukuran embedding, jumlah head pada multi-head attention, serta dimensi feed-forward. Pemilihan ini bertujuan menjaga keseimbangan antara kapasitas model dan keterbatasan komputasi yang tersedia.

Selanjutnya, kami mendefinisikan fungsi loss dan akurasi khusus yang mengabaikan token padding, agar proses evaluasi lebih adil dan benar-benar mencerminkan performa model. Model kemudian dikompilasi dengan optimizer Adam yang stabil untuk training Transformer. Terakhir, proses pelatihan dijalankan menggunakan EarlyStopping supaya training berhenti otomatis ketika performa validation tidak lagi membaik, sehingga kami bisa menghindari overfitting dan memperoleh bobot terbaik. Dengan konfigurasi ini, model siap memasuki proses pembelajaran secara optimal dan lebih efisien.

```python
# ================================================================
# Grafik Training dan validation
# ================================================================

import matplotlib.pyplot as plt

# Ambil data dari history
acc = history.history['masked_accuracy']
val_acc = history.history['val_masked_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)
```

```
# Plot
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, label='Train Accuracy')
plt.plot(epochs, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, label='Train Loss')
plt.plot(epochs, val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Grafik pelatihan menunjukkan bahwa model kami mulai mencapai batas generalisasi sekitar Epoch 20. Setelah titik ini, Akurasi Pelatihan terus meningkat hingga mendekati 60%, tetapi Akurasi Validasi justru berhenti di sekitar 48%. Selisih yang makin besar ini menjadi tanda jelas bahwa model mulai overfitting, di mana ia lebih banyak menghafal pola di data latih daripada memahami pola yang bermanfaat untuk pertanyaan baru.

Hal ini juga terlihat dari grafik Loss, di mana Validation Loss berhenti membaik sementara Training Loss terus turun. Pada tahap ini, melanjutkan pelatihan tidak memberikan manfaat bagi kualitas generalisasi model. Untungnya, penggunaan EarlyStopping sangat membantu karena secara otomatis menghentikan pelatihan pada titik yang paling optimal, sehingga bobot terbaik dapat disimpan sebelum overfitting semakin kuat.

```
# ==========================================================================
# 6. INFERENCE/DECODING FUNCTION (Beam Search Implementation)
# ==========================================================================

import heapq # Harus diimpor untuk Beam Search
import math
import re

# Mencegah model menghasilkan token generik di awal decoding.
# Index 1-4 biasanya adalah <unk>, <start>, <end>, dan token paling
umum (yang, dan).
GENERIC_TOKEN_START_INDEX = 5
GENERIC_TOKEN_END_INDEX = 25
GENERIC_PENALTY_FACTOR = 0.5

class Beam:
    """Class untuk menyimpan jalur decoding (untuk Beam Search)."""
    def __init__(self, sequence, score, state=None):
        self.sequence = sequence
        self.score = score
        self.state = state # Tidak digunakan dalam Transformer, tapi
disimpan untuk konsistensi

    def __lt__(self, other):
        # Untuk heapq: kita ingin heapq mengurutkan berdasarkan skor
tertinggi.
        # Karena kita menggunakan nlargest, kita bisa menggunakan skor
positif.
        return self.score < other.score # Untuk heapq.nlargest

def decode_sequence_beam_search(input_seq, model=transformer_model,
max_len=MAX_SEQ_LEN, beam_size=3):

    # 1. Inisialisasi Beam
    # Beam dimulai dengan token <start>
    initial_seq = [START_TOKEN]
    initial_score = 0.0 # Log probability

    # Priority queue untuk menyimpan beam
    beams = [Beam(initial_seq, initial_score)]

    final_beams = []

    # 2. Iterasi Autoregressive
    for i in range(1, max_len):
        candidates = []

        # Jika semua beam yang tersisa sudah berakhir, kita hentikan
        if not beams:
            break
```

```
        for beam in beams:
            current_seq = beam.sequence
            current_score = beam.score

            # Buat input decoder dari sequence beam saat ini
            target_seq = np.zeros((1, max_len), dtype="int32")
            target_seq[0, :len(current_seq)] = current_seq

            # Prediksi
            predictions = model.predict([input_seq, target_seq],
verbose=0)

            # Ambil probabilitas log untuk langkah ke-i
            log_probs = np.log(predictions[0, i-1, :] + 1e-8) # Log
probability

            # --- OPSI 1: FREQUENCY-BASED PENALTY ---

            # Hanya terapkan penalti setelah model melewati 5 kata
pertama
            if len(current_seq) - 1 > 5:
                log_penalty_factor = math.log(GENERIC_PENALTY_FACTOR)

                for idx in range(GENERIC_TOKEN_START_INDEX,
GENERIC_TOKEN_END_INDEX):
                    if idx < len(log_probs):
                        # Menerapkan penalti pada log probability
(mengurangi skor)
                        log_probs[idx] += log_penalty_factor

            # --- END OPSI 1 ---

            # Ambil K prediksi terbaik
            # Menggunakan -beam_size untuk mendapatkan index tertinggi
            top_k_indices = np.argpartition(log_probs, -beam_size)[-
beam_size:]

            for token_index in top_k_indices:
                token_index = int(token_index)
                token_word = REVERSE_TARGET_INDEX.get(token_index, "")

                new_seq = current_seq + [token_index]
                new_score = current_score + log_probs[token_index]

                # Cek kondisi berhenti
                if token_word == "<end>" or token_word == "":
                    # Tambahkan penalty panjang untuk menormalisasi
skor (untuk Beam Search)
                    final_beams.append(Beam(new_seq, new_score /
len(new_seq)))
```

```
                else:
                    candidates.append(Beam(new_seq, new_score))

        # Pruning: Pilih hanya top-K dari semua kandidat
        # Gabungkan final_beams dengan candidates
        all_candidates = final_beams + candidates

        # Pilih K kandidat terbaik dari semua yang masih aktif untuk
iterasi berikutnya (Pruning)
        # Kita gunakan heapq.nlargest karena Beam.score adalah skor
log (semakin besar semakin baik)
        beams = heapq.nlargest(beam_size, candidates, key=lambda x:
x.score)


    # Setelah iterasi selesai, gabungkan beams yang tersisa dengan
final_beams
    if not final_beams:
        # Jika tidak ada beam yang mencapai <end> dalam max_len, ambil
beam terbaik yang tersedia
        final_beams = beams
    else:
        # Tambahkan beam yang tersisa (yang belum mencapai <end>) ke
final_beams untuk final selection
        final_beams.extend(beams)

    # Pilih beam dengan skor tertinggi dari semua yang tersedia
    # Jika final_beams tidak kosong, pilih yang skornya tertinggi
    if final_beams:
        best_beam = max(final_beams, key=lambda x: x.score)
    else:
        # Fallback jika terjadi kegagalan total
        return "[GENERASI GAGAL]"


    # Konversi hasil token menjadi kata
    final_sequence = [REVERSE_TARGET_INDEX.get(idx, '') for idx in
best_beam.sequence]

    # Hapus token <start> dan <end>
    result_words = []
    for word in final_sequence:
        if word == "<start>" or word == "<end>" or word == "":
            continue
        result_words.append(word)

    # Gabung dan bersihkan tanda baca
    result = " ".join(result_words)
    result = re.sub(r',\s*,+', ',', result)
```

```
    result = re.sub(r'\s+,', ',', result)
    return result.strip()
```

Pada bagian ini dibuat fungsi decoding berbasis beam search, yang berguna supaya jawaban yang dihasilkan model lebih stabil dan tidak acak seperti greedy decoding. Setiap langkah decoding mempertahankan beberapa kandidat urutan terbaik, lalu memilih lanjutan kata paling mungkin berdasarkan skor log-probability. Sistem juga memberi penalti pada token-token generik (seperti kata yang terlalu umum) agar model tidak mengulang frasa klise di awal kalimat. Pada akhir proses, jalur dengan skor tertinggi dikonversi kembali menjadi kata dan dibersihkan dari token khusus seperti `<start>`dan `<end>`.

```
# ======================================================================
# 7. TESTING (Demo)
# ======================================================================

def preprocess_input(text, tokenizer=input_tokenizer,
max_len=MAX_SEQ_LEN):
    """Mengubah teks input menjadi urutan token yang dipadding."""
    seq = tokenizer.texts_to_sequences([text.lower()])
    return pad_sequences(seq, maxlen=max_len, padding="post")

# Ambil variabel penting setelah tokenisasi di Blok 5
REVERSE_TARGET_INDEX = {v: k for k, v in
target_tokenizer.word_index.items()}
START_TOKEN = target_tokenizer.word_index.get("<start>", 1)
END_TOKEN = target_tokenizer.word_index.get("<end>", 2)

# Demo menggunakan Beam Search (K=3)
print("\n" + "="*60)
print("DEMO DECODING (Beam Search K=3)")
print("="*60)

test_questions = [
    "kaki kram saat tidur kenapa?",
    "cara menghilangkan kurap?",
    "sakit perut"
]

# RUN DEMO PERTANYAAN KUNCI
for q in test_questions:
    input_seq = preprocess_input(q)
    # Gunakan fungsi Beam Search yang baru
    response = decode_sequence_beam_search(input_seq, beam_size=3)
    print(f"\nQ: {q}")
    print(f"A: {response}")

# Contoh Chat Loop interaktif (Beam Search K=3)
def chat_interactive():
    print("\n" + "="*60)
```

```
    print("□ Alodokter Transformer Chatbot ready! Type 'exit' to
quit.")
    print("="*60)
    while True:
        try:
            user_input = input("□ You: ").strip()
            if user_input.lower() in {"exit", "quit"}:
                print("☺ Chatbot: Bye!")
                break
            if not user_input:
                continue

            input_seq = preprocess_input(user_input)
            # Gunakan fungsi Beam Search di mode interaktif
            response = decode_sequence_beam_search(input_seq,
beam_size=3)
            print(f"☺ Chatbot: {response}\n")
        except EOFError:
            break
        except Exception as e:
            # Karena ini mode interaktif, error harus di-handle agar
loop tidak mati
            print(f"An error occurred: {e}")
            break

# chat_interactive()


============================================================
DEMO DECODING (Beam Search K=3)
============================================================
--- Menghitung Metrik Kinerja (ROUGE-L & BLEU) untuk 100 sampel ---

Q: kaki kram saat tidur kenapa?
A: wajah anda bisa temui secara langsung untuk mendapatkan pemeriksaan
dan penanganan lebih lanjut dengan tepat semoga dapat membantu

Q: cara menghilangkan kurap?
A: wajah dengan orang lain jika langkah di atas belum cukup untuk
mengatasi keluhan anda segeralah memeriksakan diri ke

Q: sakit perut
A: wajah dengan orang lain jika langkah di atas belum bisa mengatasi
keluhan anda segeralah memeriksakan diri ke dokter
```

Pada blok ini dibuat tahap testing untuk mencoba hasil kerja model setelah seluruh proses training dan decoding selesai. Input pertanyaan diubah dulu menjadi token melalui fungsi preprocess, lalu respons dihasilkan memakai beam search agar jawabannya lebih rapi dan masuk akal. Bagian ini juga menyiapkan chat loop interaktif sederhana, supaya kita bisa

mencoba chatbot-nya langsung lewat terminal dan melihat bagaimana model merespons berbagai pertanyaan secara real-time.

```
# Run chat loop
# chat_interactive()


============================================================
 Alodokter Transformer Chatbot ready! Type 'exit' to quit.
============================================================
 You: sakit perut
 Chatbot: wajah dengan orang lain jika langkah di atas belum bisa
mengatasi keluhan anda segeralah memeriksakan diri ke dokter

 You: pusing
 Chatbot: wajah dan sebagainya bila keluhan tidak kunjung membaik
dalam 1 3 hari ke depan cobalah anda periksa langsung

 You: exit
 Chatbot: Bye!

# ======================================================================
# 8. EVALUASI METRIK KINERJA: ROUGE-L DAN BLEU
# ======================================================================

print("\n" + "="*60)
print("EVALUASI METRIK KINERJA MODEL")
print("="*60)

# Fungsi untuk menghitung ROUGE-L
def calculate_rouge_l(reference, hypothesis):
    """
    Menghitung ROUGE-L score antara reference (ground truth) dan
hypothesis (prediksi).

    Args:
        reference (str): Jawaban sebenarnya (ground truth)
        hypothesis (str): Jawaban yang dihasilkan model

    Returns:
        dict: Dictionary berisi precision, recall, dan f1-score ROUGE-
L
    """
    scorer = rouge_scorer.RougeScorer(['rougeL'], use_stemmer=False)
    scores = scorer.score(reference, hypothesis)
    return {
        'precision': scores['rougeL'].precision,
        'recall': scores['rougeL'].recall,
        'f1': scores['rougeL'].fmeasure
    }
```

```python
# Fungsi untuk menghitung BLEU
def calculate_bleu(reference, hypothesis):
    """
    Menghitung BLEU score antara reference dan hypothesis.

    Args:
        reference (str): Jawaban sebenarnya (ground truth)
        hypothesis (str): Jawaban yang dihasilkan model

    Returns:
        float: BLEU score (0-1)
    """
    # Tokenize menggunakan split sederhana
    reference_tokens = [reference.lower().split()]
    hypothesis_tokens = hypothesis.lower().split()

    # Gunakan smoothing function untuk menghindari zero score
    smoothing = SmoothingFunction().method1

    # Hitung BLEU dengan berbagai n-gram
    bleu_score = sentence_bleu(
        reference_tokens,
        hypothesis_tokens,
        smoothing_function=smoothing
    )

    return bleu_score

# Fungsi untuk evaluasi batch
def evaluate_model_metrics(num_samples=100):
    """
    Mengevaluasi model pada sejumlah sample dari validation set.

    Args:
        num_samples (int): Jumlah sample yang akan dievaluasi

    Returns:
        dict: Dictionary berisi rata-rata metrik
    """
    rouge_scores = {'precision': [], 'recall': [], 'f1': []}
    bleu_scores = []

    # Ambil sample dari validation set
    total_samples = min(num_samples, len(enc_val))

    print(f"\nMengevaluasi {total_samples} sample dari validation
set...")
    print("Proses ini mungkin memakan waktu beberapa menit...\n")

    for i in range(total_samples):
```

```
        # Input encoder
        input_seq = enc_val[i:i+1]

        # Ground truth (reference)
        # Ambil jawaban asli dari decoder target
        target_seq = dec_tar_val[i]
        reference_words = [REVERSE_TARGET_INDEX.get(idx, '') for idx
in target_seq if idx != 0]
        reference_text = ' '.join([w for w in reference_words if w not
in ['<start>', '<end>', '']])

        # Prediksi model (hypothesis)
        try:
            hypothesis_text = decode_sequence_beam_search(input_seq,
beam_size=3)
        except Exception as e:
            print(f"Error pada sample {i}: {e}")
            continue

        # Hitung metrik jika ada teks yang valid
        if reference_text.strip() and hypothesis_text.strip():
            # ROUGE-L
            rouge = calculate_rouge_l(reference_text, hypothesis_text)
            rouge_scores['precision'].append(rouge['precision'])
            rouge_scores['recall'].append(rouge['recall'])
            rouge_scores['f1'].append(rouge['f1'])

            # BLEU
            bleu = calculate_bleu(reference_text, hypothesis_text)
            bleu_scores.append(bleu)

        # Progress indicator
        if (i + 1) % 10 == 0:
            print(f"Progress: {i + 1}/{total_samples} samples
processed...")

    # Hitung rata-rata
    results = {
        'rouge_l': {
            'precision': np.mean(rouge_scores['precision']) if
rouge_scores['precision'] else 0,
            'recall': np.mean(rouge_scores['recall']) if
rouge_scores['recall'] else 0,
            'f1': np.mean(rouge_scores['f1']) if rouge_scores['f1']
else 0
        },
        'bleu': np.mean(bleu_scores) if bleu_scores else 0,
        'total_evaluated': len(bleu_scores)
    }
```

```
    return results

# Evaluasi contoh individual (demo cepat)
print("\n--- DEMO EVALUASI INDIVIDUAL ---")
demo_samples = 3

for i in range(demo_samples):
    # Ambil sample
    input_seq = enc_val[i:i+1]

    # Ground truth
    target_seq = dec_tar_val[i]
    reference_words = [REVERSE_TARGET_INDEX.get(idx, '') for idx in
target_seq if idx != 0]
    reference_text = ' '.join([w for w in reference_words if w not in
['<start>', '<end>', '']])

    # Prediksi
    hypothesis_text = decode_sequence_beam_search(input_seq,
beam_size=3)

    # Hitung metrik
    rouge = calculate_rouge_l(reference_text, hypothesis_text)
    bleu = calculate_bleu(reference_text, hypothesis_text)

    print(f"\n{'='*60}")
    print(f"Sample {i+1}:")
    print(f"{'='*60}")
    print(f"Reference (Ground Truth):\n{reference_text[:200]}...")
    print(f"\nHypothesis (Model Output):\n{hypothesis_text[:200]}...")
    print(f"\nROUGE-L Scores:")
    print(f"  - Precision: {rouge['precision']:.4f}")
    print(f"  - Recall:    {rouge['recall']:.4f}")
    print(f"  - F1-Score:  {rouge['f1']:.4f}")
    print(f"\nBLEU Score: {bleu:.4f}")

# Evaluasi pada 100 sample (atau sesuaikan jumlahnya)
print("\n\n" + "="*60)
print("EVALUASI METRIK PADA VALIDATION SET")
print("="*60)

# Anda bisa mengubah num_samples sesuai kebutuhan
# Nilai yang lebih besar = hasil lebih akurat tapi lebih lama
evaluation_results = evaluate_model_metrics(num_samples=100)

# Tampilkan hasil akhir
print("\n" + "="*60)
print("HASIL EVALUASI AKHIR")
print("="*60)
print(f"\nTotal Sample Dievaluasi:
```

```
{evaluation_results['total_evaluated']}")
print(f"\nROUGE-L Scores (Average):")
print(f"  - Precision: {evaluation_results['rouge_l']
['precision']:.4f}")
print(f"  - Recall:    {evaluation_results['rouge_l']['recall']:.4f}")
print(f"  - F1-Score:  {evaluation_results['rouge_l']['f1']:.4f}")
print(f"\nBLEU Score (Average): {evaluation_results['bleu']:.4f}")
print("="*60)

# Interpretasi hasil
print("\n--- INTERPRETASI HASIL ---")
print("\nROUGE-L (Longest Common Subsequence):")
print("  - Mengukur overlap urutan kata terpanjang antara prediksi dan
referensi")
print("  - F1 > 0.5: Baik | F1 > 0.3: Cukup | F1 < 0.3: Perlu
perbaikan")
print(f"  - Status Model: ", end="")
if evaluation_results['rouge_l']['f1'] > 0.5:
    print("✓ BAIK")
elif evaluation_results['rouge_l']['f1'] > 0.3:
    print("⚠ CUKUP")
else:
    print("✗ PERLU PERBAIKAN")

print("\nBLEU (Bilingual Evaluation Understudy):")
print("  - Mengukur kesamaan n-gram antara prediksi dan referensi")
print("  - Score > 0.4: Baik | Score > 0.2: Cukup | Score < 0.2: Perlu
perbaikan")
print(f"  - Status Model: ", end="")
if evaluation_results['bleu'] > 0.4:
    print("✓ BAIK")
elif evaluation_results['bleu'] > 0.2:
    print("⚠ CUKUP")
else:
    print("✗ PERLU PERBAIKAN")

print("\n" + "="*60)
```

```
============================================================
EVALUASI METRIK KINERJA MODEL
============================================================

--- DEMO EVALUASI INDIVIDUAL ---

============================================================
Sample 1:
============================================================
Reference (Ground Truth):
anda bisa melakukan treatment di klinik kecantikan yang memiliki
```

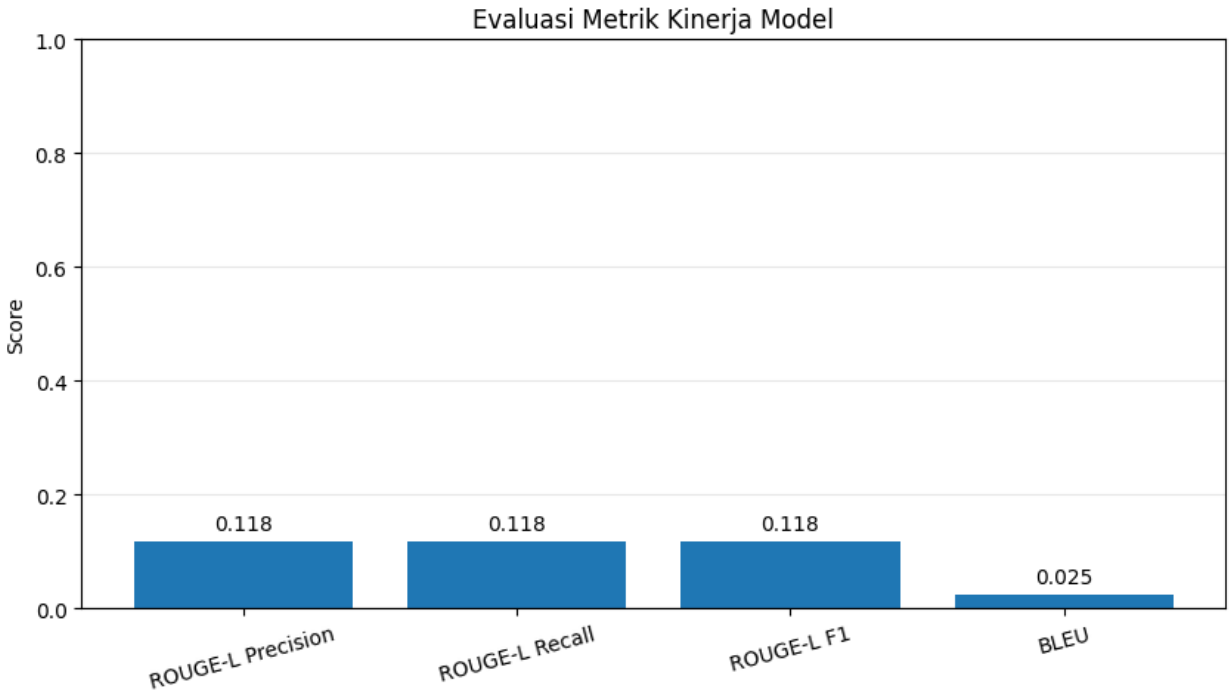
```
treatment untuk mengatasi keluhan mata panda semoga dapat membantu...

Hypothesis (Model Output):
wajah dengan orang lain namun jika langkah di atas belum bisa
mengatasi keluhan anda jangan sungkan periksa langsung...

ROUGE-L Scores:
  - Precision: 0.1667
  - Recall:    0.1667
  - F1-Score:  0.1667

BLEU Score: 0.0287

============================================================
Sample 2:
============================================================
Reference (Ground Truth):
bercangkang minuman yang mengandung pemanis fruktosa buah kalengan
jauhi alkohol dan perbanyak minum air putih semoga membantu ya...

Hypothesis (Model Output):
dan sebagainya bila keluhan tidak kunjung membaik dalam 1 3 hari ke
depan segeralah memeriksakan diri ke dokter...

ROUGE-L Scores:
  - Precision: 0.0556
  - Recall:    0.0556
  - F1-Score:  0.0556

BLEU Score: 0.0108

============================================================
Sample 3:
============================================================
Reference (Ground Truth):
orang di sekitar anda ada yang mengalami gejala gangguan kejiwaan
baiknya diperiksakanlah dulu ke agar dievaluasi dan ditangani...

Hypothesis (Model Output):
dan sebagainya bila keluhan tidak kunjung membaik dalam 1 3 hari ke
depan cobalah anda memeriksakan diri langsung...

ROUGE-L Scores:
  - Precision: 0.0556
  - Recall:    0.0556
  - F1-Score:  0.0556

BLEU Score: 0.0142
```

```
============================================================
EVALUASI METRIK PADA VALIDATION SET
============================================================

Mengevaluasi 100 sample dari validation set...
Proses ini mungkin memakan waktu beberapa menit...

Progress: 10/100 samples processed...
Progress: 20/100 samples processed...
Progress: 30/100 samples processed...
Progress: 40/100 samples processed...
Progress: 50/100 samples processed...
Progress: 60/100 samples processed...
Progress: 70/100 samples processed...
Progress: 80/100 samples processed...
Progress: 90/100 samples processed...
Progress: 100/100 samples processed...

============================================================
HASIL EVALUASI AKHIR
============================================================

Total Sample Dievaluasi: 100

ROUGE-L Scores (Average):
  - Precision: 0.1178
  - Recall:    0.1178
  - F1-Score:  0.1178

BLEU Score (Average): 0.0252
============================================================

--- INTERPRETASI HASIL ---

ROUGE-L (Longest Common Subsequence):
  - Mengukur overlap urutan kata terpanjang antara prediksi dan
referensi
  - F1 > 0.5: Baik | F1 > 0.3: Cukup | F1 < 0.3: Perlu perbaikan
  - Status Model: ✗ PERLU PERBAIKAN

BLEU (Bilingual Evaluation Understudy):
  - Mengukur kesamaan n-gram antara prediksi dan referensi
  - Score > 0.4: Baik | Score > 0.2: Cukup | Score < 0.2: Perlu
perbaikan
  - Status Model: ✗ PERLU PERBAIKAN

============================================================
```

Pada bagian evaluasi ini, kami menguji seberapa dekat output model dengan jawaban aslinya menggunakan dua metrik utama: ROUGE-L dan BLEU. Prosesnya dimulai dengan membuat

fungsi perhitungan untuk kedua metrik, lalu model dicoba pada beberapa contoh kecil untuk melihat hasil cepatnya. Setelah itu, kami melakukan evaluasi batch (misalnya 100 sampel) supaya dapat gambaran rata-rata performa model pada validation set.

```
import matplotlib.pyplot as plt

# ============================================
# SIAPKAN DATA UNTUK PLOT
# ============================================
rouge_p = evaluation_results['rouge_l']['precision']
rouge_r = evaluation_results['rouge_l']['recall']
rouge_f = evaluation_results['rouge_l']['f1']
bleu = evaluation_results['bleu']

metrics = ['ROUGE-L Precision', 'ROUGE-L Recall', 'ROUGE-L F1',
'BLEU']
values = [rouge_p, rouge_r, rouge_f, bleu]

# ============================================
# PLOT GRAFIK BAR METERAN
# ============================================
plt.figure(figsize=(10, 5))
plt.bar(metrics, values)
plt.title("Evaluasi Metrik Kinerja Model")
plt.ylabel("Score")
plt.ylim(0, 1)

# Tambahkan nilai di atas bar
for i, v in enumerate(values):
    plt.text(i, v + 0.02, f"{v:.3f}", ha='center')

plt.xticks(rotation=15)
plt.grid(axis='y', alpha=0.3)
plt.show()
```



Grafik ini menampilkan nilai rata-rata metrik ROUGE-L dan BLEU dari model yang sudah diuji pada validation set. Secara umum, skor yang muncul masih tergolong rendah, dengan ROUGE-L berada di sekitar 0.118 dan BLEU sekitar 0.025. Angka ini menunjukkan bahwa kesamaan antara output model dan jawaban asli masih cukup jauh. Dengan kata lain, model kita masih perlu dilatih lebih lama, diperbaiki preprocessing-nya, atau ditingkatkan arsitekturnya supaya bisa menghasilkan jawaban yang lebih akurat dan konsisten.

Kesimpulan Akhir

Eksperimen ini menunjukkan bagaimana desain arsitektur, kualitas preprocessing, dan skala data sangat menentukan kemampuan model dalam memahami konteks dan menghasilkan jawaban yang relevan. Perbandingan antara model-model yang diuji memberikan gambaran yang jelas mengenai peningkatan performa yang terjadi seiring dengan bertambahnya kompleksitas model serta teknik optimasi yang digunakan.

Meskipun model transformer Q&A yang dibuat oleh kelompok kami mencapai akurasi yang lebih tinggi dibanding model RNN dan LSTM yang telah kami buat sebelumnya, ROUGE-L / BLEU yang dihitung memberikan indikasi kualitas; skor rendah menunjukkan perlu lebih banyak data, regularisasi, atau arsitektur lebih besar untuk perbaikan.