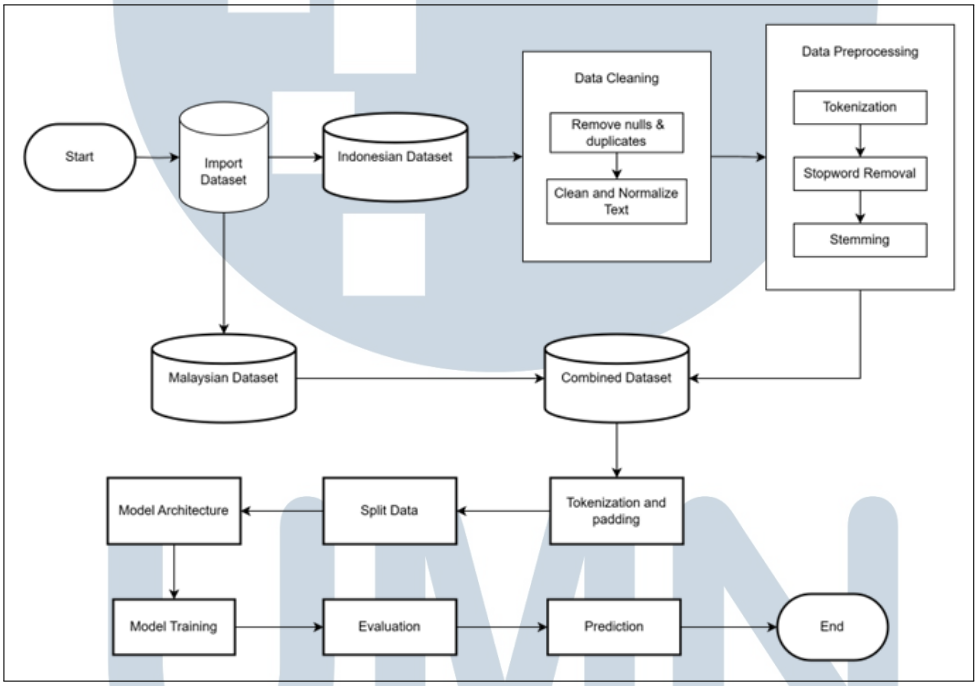
BAB 3
METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan klasifikasi teks biner untuk mendeteksi berita hoaks dalam dua korpus berbahasa: Melayu dan Indonesia. Secara garis besar, tahapan penelitian ini mencakup: pengumpulan data, pembersihan data, pra-pemrosesan teks, tokenisasi dan *padding*, pelatihan model menggunakan arsitektur BiLSTM, serta evaluasi performa model terhadap data uji. Gambar 3.1 menyajikan Arsitektur metode penelitian yang digunakan.



Gambar 3.1. Arsitektur metode penelitian

3.1 Pengumpulan Data

Penelitian ini menggunakan dua jenis korpus utama berdasarkan bahasa, yaitu korpus berbahasa Melayu dan korpus berbahasa Indonesia. Seluruh dataset yang digunakan bersifat publik dan diperoleh dari repositori daring yang dapat diakses secara bebas. Setiap dataset dipilih berdasarkan relevansi topik, kualitas anotasi, serta representasi data hoaks dan fakta secara seimbang untuk mendukung proses pelatihan model klasifikasi.

3.1.1 Dataset Malaysia

Dataset berbahasa Melayu diperoleh dari repositori GitHub publik dengan nama `AsyrafAzlan/malay-fake-news-classification`. Dataset ini telah melalui tahap pra-pemrosesan oleh penyusun dataset sebelum digunakan dalam penelitian ini. Meskipun proses pra-pemrosesan awal seperti stemming dan penghapusan *stopwords* tidak dijelaskan secara rinci, format data yang sudah bersih memungkinkan integrasi yang cepat ke dalam korpus gabungan. Untuk menjaga konsistensi dalam representasi input model, seluruh data tetap melewati tahap tokenisasi dan *padding*.

3.1.2 Dataset Indonesia

Terdapat dua dataset utama berbahasa Indonesia yang digunakan dalam penelitian ini, keduanya diperoleh dari platform Kaggle. Dataset pertama adalah *Indonesian Fact and Hoax Political News Dataset* [17], yang berisi artikel berita politik yang telah diklasifikasikan sebagai fakta atau hoaks. Label pada dataset ini didasarkan pada verifikasi dari Turnbackhoax, dengan sumber berita berasal dari portal seperti CNN Indonesia, Kompas, dan Tempo.

Dataset kedua adalah *Indonesia False News (Hoax) Dataset* [18], yang terdiri dari dua berkas utama, yaitu `data_latih.csv` dan `data_uji.csv`. Dataset ini mencakup ribuan artikel dari tahun 2018 hingga 2020 dengan label hoaks atau fakta, dan mencakup berbagai topik seperti politik, kesehatan publik, serta isu-isu terkait pandemi COVID-19.

Distribusi jumlah data dari masing-masing dataset dapat dilihat pada Tabel 3.1 berikut:

Tabel 3.1. Distribusi Data Dataset Hoaks Berbahasa Melayu dan Indonesia

Sumber Data	**Total Data**	**Data Fakta**	**Data Hoaks**
Dataset Melayu	37.592	25.030	12.562
Dataset Indonesia	30.343	9.575	20.768
Total	**67.935**	**34.605**	**33.330**

3.2 Data Cleaning

Proses pembersihan data difokuskan pada dataset berbahasa Indonesia yang diperoleh dalam format mentah. Tahapan awal melibatkan penghapusan data

dengan nilai kosong serta duplikat agar tidak mengganggu proses pelatihan model. Selanjutnya dilakukan normalisasi dengan mengubah seluruh huruf menjadi huruf kecil. Proses pembersihan lanjutan mencakup penggantian tanda hubung dengan spasi, penghapusan URL, simbol media sosial seperti *mention* dan *hashtag*, serta tag atau entitas HTML. Selain itu, karakter yang berulang secara berlebihan seperti "bangetttt" dinormalisasi menjadi bentuk umum seperti "banget". Tanda baca dan spasi berlebih juga dihapus untuk menghasilkan teks yang lebih bersih dan seragam. Seluruh tahapan ini diterapkan secara berurutan untuk menghasilkan kolom akhir *CleanedText*. Sementara itu, dataset berbahasa Melayu tidak melalui tahap pembersihan ulang karena telah dipra-pemrosesan sebelumnya, namun tetap diperlakukan secara konsisten pada tahap tokenisasi dan *padding*. Implementasi kode pembersihan dapat dilihat di bawah ini:

```
# Hapus baris dengan nilai NaN pada kolom 'text'
df_all.dropna(subset=['text'], inplace=True)

# Hapus duplikat setelah menghapus NaN
df_all.drop_duplicates(subset=['text'], keep='first', inplace=True
    )

# Konversi ke huruf kecil
def lowercase_text(text):
    return text.lower()

df_all['LowerCase'] = df_all['text'].apply(lowercase_text)

import re

# Fungsi pembersihan teks
def clean_text(text):
    text = text.replace('-', ' ')
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)          #
    Hapus URL
    text = re.sub(r'@\w+|#\w+', '', text)                        #
    Hapus mention dan hashtag
    text = re.sub(r'<.*?>', '', text)                            #
    Hapus tag HTML
    text = re.sub(r'&[a-z]+;', '', text)                         #
    Hapus entitas HTML
    text = re.sub(r'(.)\1{2,}', r'\1\1', text)                   #
    Normalisasi karakter berulang
```

```
    text = re.sub(r'\s+', ' ', text)                                    # Hapus spasi berlebih
    text = re.sub(r'[^\w\s]', '', text)                                 # Hapus tanda baca
    return text.strip()

# Terapkan pembersihan ke kolom 'LowerCase'
df_all['CleanedText'] = df_all['LowerCase'].apply(clean_text)
```

Kode 3.1: Proses pembersihan data teks pada dataset Indonesia

3.3 Pra-pemrosesan Teks

Setelah melalui proses pembersihan, data berbahasa Indonesia diproses lebih lanjut melalui beberapa tahapan pra-pemrosesan untuk menyiapkan data dalam bentuk yang sesuai untuk pelatihan model. Proses ini mencakup tokenisasi, penghapusan *stopwords*, dan stemming. Seluruh proses dilakukan secara berurutan dan hasil akhirnya digunakan bersama dengan data berbahasa Melayu yang telah disamakan strukturnya.

3.3.1 Tokenisasi

Tokenisasi dilakukan menggunakan fungsi `word_tokenize` dari pustaka NLTK untuk mengubah teks menjadi deretan kata secara terpisah. Proses ini bertujuan agar setiap kata dapat diproses secara individual dalam tahap-tahap selanjutnya seperti penghapusan *stopwords* dan stemming. Hasil tokenisasi disimpan dalam kolom baru bernama `Tokens`. Implementasi kode Tokenisasi dapat dilihat di bawah ini:

```
import nltk
nltk.download('punkt')  # Jalankan sekali untuk mengunduh tokenizer

from nltk.tokenize import word_tokenize

# Membuat kolom baru 'Tokens'
df_all['Tokens'] = df_all['CleanedText'].apply(word_tokenize)
```

Kode 3.2: Proses tokenisasi menggunakan NLTK

3.3.2 Penghapusan Stopwords

Kata-kata umum yang tidak memiliki kontribusi penting terhadap makna utama teks, seperti “dan”, “yang”, atau “di”, dihapus dari daftar token menggunakan pustaka Sastrawi. Langkah ini bertujuan untuk mengurangi noise dalam data sehingga model dapat lebih fokus pada kata-kata bermakna. Daftar kata yang dihapus berasal dari fungsi bawaan `get_stop_words()`. Implementasi kode penghapusan *stopwords* dapat dilihat di bawah ini:

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
    StopWordRemoverFactory

# Inisialisasi stopword remover
stop_factory = StopWordRemoverFactory()
stopwords = set(stop_factory.get_stop_words())

# Fungsi untuk menghapus stopwords dari token
def remove_stopwords(tokens):
    return [w for w in tokens if w not in stopwords]

# Terapkan fungsi ke kolom token
df_all['FilteredTokens'] = df_all['Tokens'].apply(remove_stopwords
    )
```

Kode 3.3: Penghapusan *stopwords* menggunakan Sastrawi

3.3.3 Stemming

Tahap akhir dalam pra-pemrosesan adalah stemming, yaitu proses mengembalikan setiap kata ke bentuk dasarnya. Sebagai contoh, kata-kata seperti ’pemilu’, ’kepemiluan’, dan ’berpemilu’ akan distandarkan menjadi ’pemilu’. Proses ini dilakukan menggunakan `StemmerFactory` dari pustaka Sastrawi dan hasilnya disimpan dalam kolom `StemmedTokens`. Implementasi kode *stemming* dapat dilihat di bawah ini:

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

# Inisialisasi stemmer
stem_factory = StemmerFactory()
stemmer = stem_factory.create_stemmer()

# Fungsi untuk stemming token
```

```python
def stem_tokens(tokens):
    return [stemmer.stem(w) for w in tokens]

# Terapkan stemming pada token yang telah difilter
df_all['StemmedTokens'] = df_all['FilteredTokens'].apply(
    stem_tokens)
```

Kode 3.4: Proses stemming menggunakan `StemmerFactory` dari Sastrawi

3.4 Split Data

Setelah proses tokenisasi dan praproses lainnya selesai dilakukan, tahap selanjutnya adalah membagi dataset menjadi tiga bagian: data latih 80%, data validasi 10%, dan data uji 10%. Pembagian ini penting untuk memastikan bahwa model dapat dilatih secara optimal serta dievaluasi dengan adil pada data yang belum pernah dilihat sebelumnya.

Proses pembagian data dilakukan secara bertahap menggunakan fungsi `train_test_split` dari pustaka `scikit-learn`. Parameter `stratify` digunakan agar distribusi label pada masing-masing subset tetap seimbang. Dengan demikian, model tidak akan bias terhadap kelas tertentu akibat distribusi data yang tidak merata. Berikut adalah implementasi kode pembagian data tersebut:

```python
from sklearn.model_selection import train_test_split

# 80% train, 10% val, 10% test
X_train, X_temp, y_train, y_temp = train_test_split(
    texts, labels, test_size=0.20, stratify=labels, random_state
    =42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state
    =42
)

print(f'Train : {len(X_train)}')
print(f'Val   : {len(X_val)}')
print(f'Test  : {len(X_test)}')
```

Kode 3.5: Pembagian data menjadi train, validation, dan test set

3.5 Tokenisasi dan Padding

Pada tahap ini, data teks diubah menjadi representasi numerik menggunakan `Tokenizer` dari pustaka Keras, yang dikonfigurasi dengan `num_words=30000` dan token khusus `<OOV>` untuk menangani kata-kata di luar kosakata. Tokenizer hanya di-*fit* pada data latih (`X_train`) guna menghindari kebocoran informasi. Selanjutnya, setiap data (latih, validasi, dan uji) dikonversi menjadi urutan bilangan bulat menggunakan metode `texts_to_sequences`, kemudian diseragamkan panjangnya menjadi 512 token melalui `post-padding` dan `pre-truncating`, sehingga menghasilkan input berdimensi tetap yang siap digunakan dalam pelatihan model.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Konfigurasi tokenizer
vocab_size  = 30_000
max_length  = 512
tokenizer   = Tokenizer(num_words=vocab_size + 1, oov_token='<OOV>')

# Fit hanya pada data latih
tokenizer.fit_on_texts(X_train)

# Fungsi util untuk mengubah teks     padded sequence
def to_padded_seq(text_array):
    seq = tokenizer.texts_to_sequences(text_array)
    return pad_sequences(seq, maxlen=max_length, padding='post', truncating='pre')

# Konversi tiap split
X_train_seq = to_padded_seq(X_train)
X_val_seq   = to_padded_seq(X_val)
X_test_seq  = to_padded_seq(X_test)
```

Kode 3.6: Proses tokenisasi dan padding pada data teks

3.6 Konfigurasi Model

3.6.1 Arsitektur Model BiLSTM

Model yang digunakan dalam penelitian ini adalah *Bidirectional Long Short-Term Memory* (BiLSTM), yang dirancang untuk mempelajari konteks sekuensial dari dua arah sekaligus, yakni maju dan mundur, sehingga meningkatkan pemahaman terhadap struktur kalimat. Arsitektur model dibangun menggunakan `Sequential API` dari Keras dan terdiri atas beberapa lapisan utama: `Embedding Layer` dengan dimensi 128 untuk mengubah indeks kata menjadi vektor berdimensi tetap, diikuti oleh `SpatialDropout1D` sebagai regularisasi ringan untuk mengurangi overfitting. Selanjutnya, terdapat dua lapisan `BiLSTM` bertingkat, masing-masing dengan 128 dan 64 unit, yang dipisahkan oleh `Dropout` sebesar 0,3. Lapisan berikutnya adalah `Dense Layer` dengan 64 unit dan fungsi aktivasi ReLU, dan diakhiri oleh `Output Layer` dengan satu neuron dan aktivasi sigmoid untuk klasifikasi biner. Model dikompilasi menggunakan fungsi `binary_crossentropy` dan optimizer `Adam`, serta metrik evaluasi akurasi. Implementasi kode lengkap arsitektur BiLSTM disajikan pada Listing 3.7.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Embedding, SpatialDropout1D,
                                     Bidirectional, LSTM,
                                     Dropout, Dense)

embedding_dim = 128          # bisa dinaikkan mis. 256 jika GPU cukup
max_length    = 512          # sama seperti pada padding

model = Sequential([
    Embedding(input_dim=vocab_size + 1,          # 30  001
              output_dim=embedding_dim,
              input_shape=(max_length,)),        # (batch, 512)
    SpatialDropout1D(0.2),                       # regularisasi ringan
    Bidirectional(LSTM(128, return_sequences=True)),
    Dropout(0.30),
    Bidirectional(LSTM(64)),                     # return_sequences=False
    Dropout(0.30),
    Dense(64, activation='relu'),
    Dense(1,  activation='sigmoid')              # binary output
```

```
])

# Bangun model secara eksplisit agar tidak unbuilt
model.build(input_shape=(None, max_length))
model.summary()

# Kompilasi model
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Kode 3.7: Arsitektur Model BiLSTM untuk Klasifikasi Teks

3.6.2 Konfigurasi Pelatihan Model

Model dikompilasi menggunakan `Adam optimizer` dan fungsi kerugian `binary_crossentropy`, yang umum digunakan pada tugas klasifikasi biner. Proses pelatihan dilakukan selama maksimal 10 *epoch* dengan ukuran *batch* sebesar 128. Pada akhir setiap *epoch*, dilakukan evaluasi terhadap data validasi untuk memantau performa generalisasi model selama pelatihan. Tabel 3.2 menunjukkan konfigurasi hyperparameter yang digunakan selama proses pelatihan untuk mengoptimalkan performa model dalam mendeteksi berita hoaks secara biner.

Tabel 3.2. Konfigurasi Hyperparameter Pelatihan

Hyperparameter	**Nilai**
Optimizer	Adam
Fungsi Kerugian	Binary Crossentropy
Ukuran Batch	128
Jumlah Epoch Maksimal	10
Dropout	0.2–0.3
Aktivasi Output	Sigmoid

3.7 Evaluasi Model

Kinerja model dievaluasi menggunakan sejumlah metrik utama, seperti *accuracy*, *precision*, *recall*, dan *F1-score*, yang dihasilkan melalui *classification report* berdasarkan prediksi terhadap data uji. Metrik *accuracy* mengukur persentase keseluruhan prediksi yang benar, sedangkan *precision* menghitung

proporsi prediksi positif yang benar terhadap seluruh prediksi positif. *Recall* menilai sejauh mana model mampu mengidentifikasi seluruh data positif yang sebenarnya, dan *F1-score* merupakan rata-rata harmonik dari precision dan recall, yang sangat penting ketika data bersifat tidak seimbang.

Selain evaluasi pada data uji, proses pelatihan model juga dimonitor menggunakan nilai *validation accuracy* dan *validation loss* pada setiap *epoch*, guna mengamati kemampuan generalisasi model dan mendeteksi potensi *overfitting*.

Untuk melengkapi analisis, *confusion matrix* disusun untuk menunjukkan distribusi prediksi model terhadap kelas sebenarnya. Matriks ini memungkinkan identifikasi jumlah prediksi benar dan salah dari setiap kelas, serta memfasilitasi pemahaman terhadap jenis kesalahan yang sering dilakukan oleh model.

3.8 Pembuatan Laporan dan Dokumentasi

Pembuatan laporan dilakukan secara sistematis dengan merujuk pada struktur penulisan karya ilmiah tingkat sarjana. Setiap tahap penelitian, mulai dari pengumpulan data hingga evaluasi model, didokumentasikan secara rinci menggunakan bahasa formal dan teknis. Penulisan laporan menggunakan format LaTeX untuk memastikan konsistensi tata letak, kemudahan pengutipan referensi, serta penyusunan tabel dan gambar yang rapi dan profesional.