BAB 3
METODOLOGI PENELITIAN

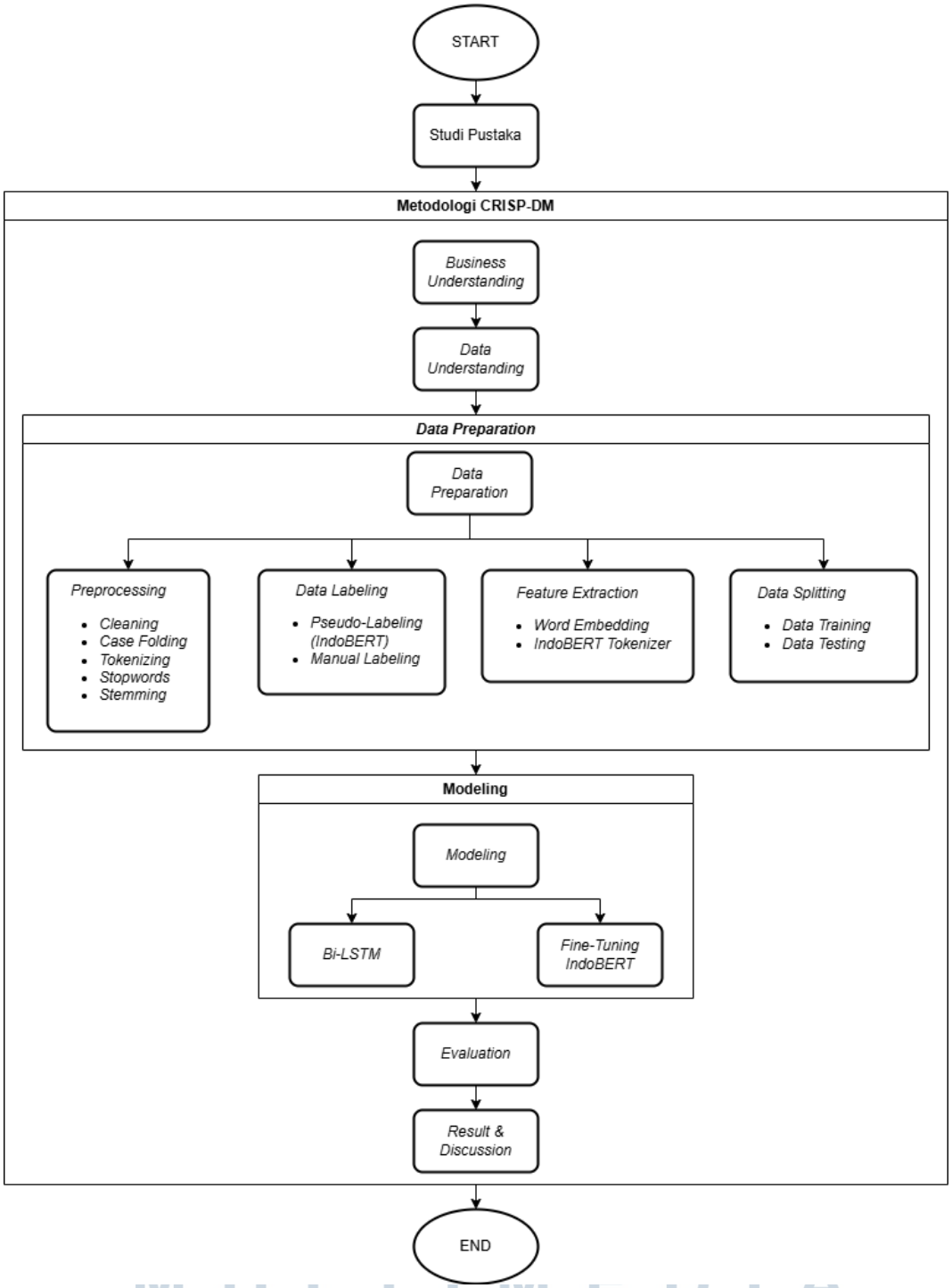
Bab ini menguraikan secara rinci dan sistematis seluruh rancangan dan langkah-langkah teknis yang dilakukan dalam penelitian. Penyusunan metodologi yang terstruktur sangat krusial untuk memastikan bahwa penelitian dapat direplikasi, divalidasi, dan hasilnya dapat dipertanggungjawabkan secara ilmiah [45]. Oleh karena itu, penelitian ini mengadopsi kerangka kerja *Cross-Industry Standard Process for Data Mining* (CRISP-DM) sebagai panduan utama dalam pelaksanaan proyek analisis data.

Metodologi CRISP-DM dipilih karena merupakan standar industri terbuka yang paling banyak diadopsi untuk proyek *data science* dan terbukti fleksibel untuk diterapkan pada berbagai domain, termasuk analisis data tidak terstruktur seperti teks [39, 41]. Berbeda dengan model proses lain seperti KDD (*Knowledge Discovery in Databases*) yang lebih berfokus pada tahap teknis, CRISP-DM mencakup seluruh siklus hidup proyek, mulai dari pemahaman masalah bisnis hingga evaluasi hasil, yang sangat selaras dengan tujuan penelitian ini [40].

3.1 Kerangka dan Alur Penelitian

Alur kerja penelitian ini secara keseluruhan divisualisasikan dalam sebuah diagram alur pada Gambar 3.1. Diagram ini menggambarkan tahapan-tahapan utama, mulai dari studi pustaka, diikuti dengan fase-fase dalam metodologi CRISP-DM. Inti dari metodologi ini adalah desain eksperimen komparatif, di mana dua pendekatan pemodelan yang berbeda dievaluasi secara paralel.

Pendekatan eksperimen komparatif ini dirancang untuk dapat menarik kesimpulan yang lebih kuat dengan cara membandingkan secara langsung kinerja dari beberapa metode dalam kondisi yang terkontrol [46]. Dalam penelitian ini, perbandingan dilakukan antara arsitektur sekuensial klasik (Bi-LSTM) yang dilatih pada data bervolume besar hasil *pseudo-labeling*, dengan arsitektur Transformer modern (IndoBERT) yang di-*fine-tuning* pada data berkualitas tinggi hasil pelabelan manual. Desain ini tidak hanya membandingkan model, tetapi juga dua strategi persiapan data yang berbeda.



Gambar 3.1. Diagram Alur Penelitian

3.2 Perangkat Penelitian dan Sumber Data

Pelaksanaan penelitian ini didukung oleh serangkaian perangkat keras, perangkat lunak, dan sumber data primer yang spesifik.

3.2.1 Spesifikasi Perangkat Keras

Untuk mendukung tahap komputasi yang intensif seperti pelatihan model *deep learning*, digunakan perangkat keras (*hardware*) dengan rincian sebagai berikut:

1. **Model Laptop**: ASUS ROG GL553VD
2. **Prosesor**: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
3. **GPU**: NVIDIA GeForce GTX 1050
4. **RAM**: 16.0 GB

3.2.2 Sumber Data

Data primer dalam penelitian ini diperoleh dari platform media sosial X, yang sebelumnya dikenal sebagai Twitter. Platform *microblogging* ini dipilih karena sifatnya yang *real-time* dan kemampuannya dalam menangkap opini publik secara luas dan spontan. Karakteristik ini menjadikannya sumber data yang sangat relevan untuk menganalisis sentimen masyarakat terhadap peristiwa ekonomi yang dinamis seperti fluktuasi IHSG [13].

3.2.3 Perangkat Lunak Pendukung

Seluruh alur kerja penelitian, mulai dari pengumpulan data hingga pemodelan, diimplementasikan menggunakan serangkaian perangkat lunak (*software*) berikut:

1. **Sistem Operasi**: Microsoft Windows 10
2. **Bahasa Pemrograman**: Python 3.10 [47]
3. **Lingkungan Pengembangan**: Google Colaboratory dan Visual Studio Code.
4. **Pustaka Utama**:

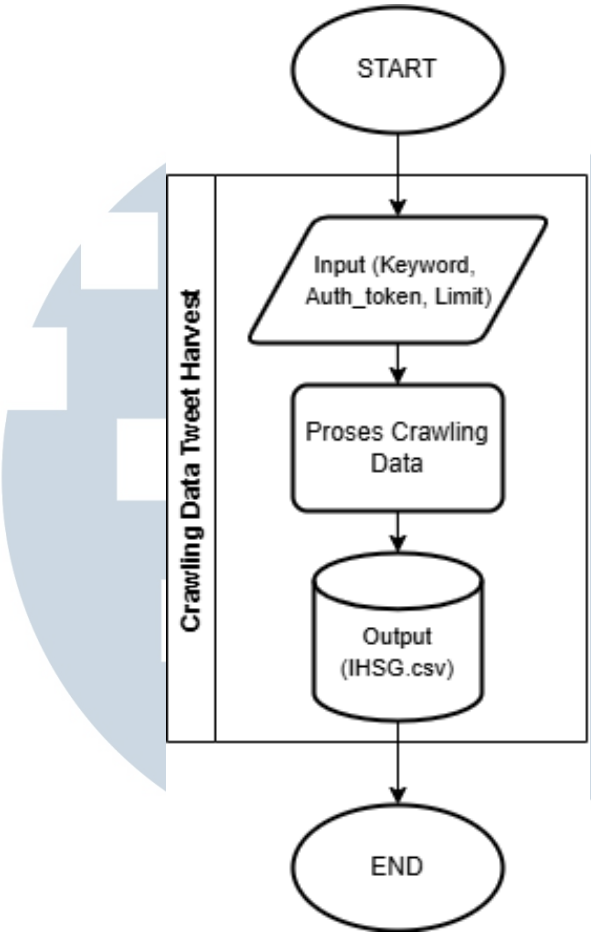
- Pengumpulan Data: `tweet-harvest`.
- Manipulasi Data: `Pandas`.
- *Preprocessing* NLP: `NLTK`, `Sastrawi`.
- Pemodelan & Evaluasi: `Scikit-learn`, `TensorFlow/Keras`, `Transformers`, `Optuna`.
- Visualisasi: `Matplotlib`, `Seaborn`, `wordcloud`.

3.3 Tahapan Penelitian

Proses penelitian dilaksanakan melalui empat tahapan utama: pengumpulan data, persiapan data, pemodelan komparatif, dan metode evaluasi.

3.3.1 Pengumpulan Data

Penelitian ini memanfaatkan data primer yang diambil secara langsung dari platform X. Proses pengumpulan data dilakukan melalui teknik *web scraping*, dengan tujuan menghimpun opini serta sentimen publik terkait peristiwa penurunan tajam IHSG dalam rentang waktu 1 Maret 2025 hingga 30 April 2025. Proses ini diawali dengan penentuan parameter input, dieksekusi menggunakan *tool* `tweet-harvest` (v2.6.1), dan diakhiri dengan penyimpanan data dalam format `.csv`, seperti diilustrasikan pada Gambar 3.2.



Gambar 3.2. Diagram Alur Proses Pengumpulan Data

Parameter kunci yang digunakan dalam proses *crawling* adalah sebagai berikut:

- **Kata Kunci (`search_keyword`):** `'IHSG since:2025-03-01 until:2025-04-30 lang:id'`.
- **Batas Pengambilan Data (`limit`):** `5000`.
- **Token Otentikasi (`twitter_auth_token`):** Kredensial unik dari akun X peneliti.

Implementasi teknis dari proses ini ditunjukkan pada Cuplikan Kode 3.1.

```
# Definisikan parameter
filename = 'IHSG.csv'
search_keyword = 'IHSG since:2025-03-01 until:2025-04-30 lang:id'
limit = 5000
auth_token = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' # Token disensor
```

```
6
7 # Jalankan perintah crawling
8 !npx -y tweet-harvest@2.6.1 -o "{filename}" -s "{search_keyword}"
    --tab "LATEST" -l {limit} --token {auth_token}
```

Kode 3.1: Eksekusi Pengambilan Data dengan Tweet-Harvest

3.3.2 Persiapan Data

Fase ini mencakup semua aktivitas untuk membangun dataset final yang bersih dan terstruktur dari data mentah.

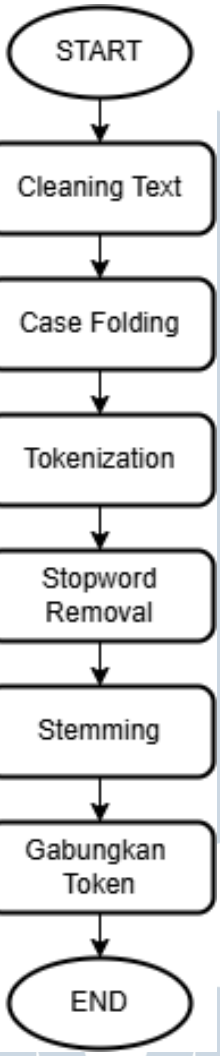
A Analisis Data Eksploratif

Sebelum pemrosesan teknis, dilakukan analisis data eksploratif menggunakan teknik visualisasi *Word Cloud*. Tujuannya adalah untuk mengidentifikasi tema dominan dalam percakapan publik dan memberikan konteks awal sebelum analisis kuantitatif.

B Preprocessing Teks

Preprocessing adalah langkah fundamental untuk membersihkan dan mentransformasi data teks mentah menjadi format yang siap untuk pemodelan. Alur proses yang diilustrasikan pada Gambar 3.3 diimplementasikan menggunakan pustaka NLTK dan Sastrawi. Proses ini mencakup lima tahapan:

1. ***Cleaning*:** Menghapus elemen-elemen yang tidak relevan seperti *username*, URL, *hashtag*, angka, dan tanda baca.
2. ***Case Folding*:** Menyeragamkan seluruh teks menjadi huruf kecil.
3. ***Tokenization*:** Memecah kalimat menjadi unit-unit kata individual (token).
4. ***Stopword Removal*:** Menghapus kata-kata umum yang tidak memiliki makna sentimen signifikan (misalnya, "yang", "di", "dan").
5. ***Stemming*:** Mengubah setiap kata ke bentuk kata dasarnya untuk menyeragamkan kosakata (misalnya, "menurun" menjadi "turun").



Gambar 3.3. Diagram Alur Proses Preprocessing Teks

Contoh transformasi teks pada setiap tahapan disajikan pada Tabel 3.1.

Tabel 3.1. Contoh Hasil Setiap Tahapan Preprocessing

Tahapan	**Contoh Teks**
Teks Asli	`@antaranewscom IHSG ditutup melemah terbatas di tengah penguatan bursa saham kawasan Asia sore ini. #IHSG`
Cleaning & Case Folding	`ihsg ditutup melemah terbatas di tengah penguatan bursa saham kawasan asia sore ini`
Tokenization	`['ihsg', 'ditutup', 'melemah', 'terbatas', 'di', 'tengah', 'penguatan', 'bursa', 'saham', 'kawasan', 'asia', 'sore', 'ini']`
Stopword Removal	`['ihsg', 'ditutup', 'melemah', 'terbatas', 'penguatan', 'bursa', 'saham', 'kawasan', 'asia']`
Stemming (Hasil Akhir)	`['ihsg', 'tutup', 'lemah', 'batas', 'kuat', 'bursa', 'saham', 'kawasan', 'asia']`

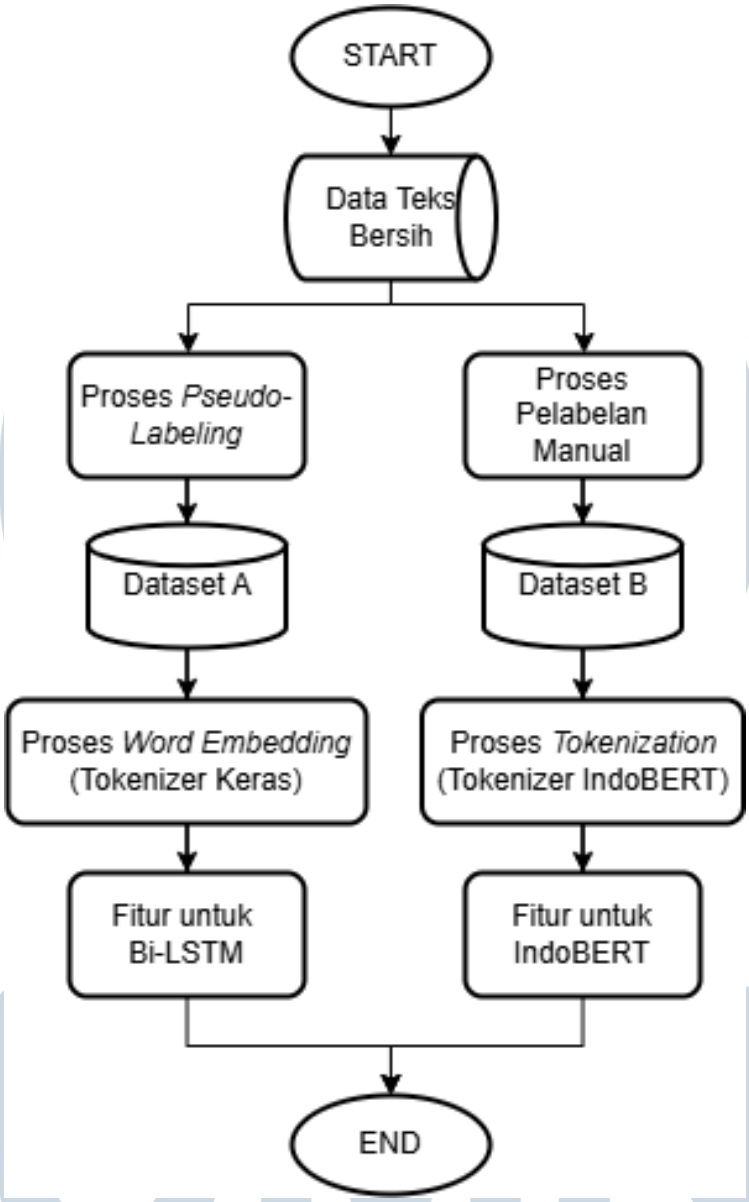
```python
import pandas as pd
import re
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from nltk.corpus import stopwords as nltk_stopwords
from nltk.tokenize import TweetTokenizer

# 1. Fungsi Cleaning Teks
def clean_tweet(text):
    text = str(text)
    text = re.sub(r'@[A-Za-z0-9_]+', '', text)  # Hapus Username
    text = re.sub(r'http\S+|www\S+', '', text)   # Hapus URL
    text = re.sub(r'#\w+', '', text)             # Hapus Hashtag
    text = re.sub(r'\d+', '', text)              # Hapus angka
    text = re.sub(r'[^\w\s]', '', text)          # Hapus tanda baca
    text = re.sub(r'\s+', ' ', text).strip()     # Hapus spasi
    berlebih
    return text

# 2. Fungsi Case Folding
def case_folding(text):
    return text.lower()
```

```
21
22 # 3. Fungsi Tokenisasi
23 tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True
       , reduce_len=True)
24 def tokenize_tweet(text):
25     return tokenizer.tokenize(text)
26
27 # 4. Fungsi Stopword Removal
28 list_stopwords_id = set(nltk_stopwords.words('indonesian'))
29 def remove_stopwords(tokens):
30     return [word for word in tokens if word not in
       list_stopwords_id]
31
32 # 5. Fungsi Stemming
33 factory = StemmerFactory()
34 stemmer = factory.create_stemmer()
35 def stem_text(tokens):
36     return [stemmer.stem(word) for word in tokens]
37
38 # Alur pemrosesan pada DataFrame
39 df['cleaned_tweet'] = df['full_text'].apply(clean_tweet)
40 df['folded_tweet'] = df['cleaned_tweet'].apply(case_folding)
41 df['tokenized_tweet'] = df['folded_tweet'].apply(tokenize_tweet)
42 df['filtered_tweet'] = df['tokenized_tweet'].apply(
       remove_stopwords)
43 df['stemmed_tweet'] = df['filtered_tweet'].apply(stem_text)
44 df['processed_text'] = df['stemmed_tweet'].apply(lambda x: ' '.
       join(x))
```

Kode 3.2: Implementasi Lengkap Alur Preprocessing Teks

C Alur Persiapan Dataset Eksperimental

Setelah teks mentah berhasil dibersihkan melalui tahap *preprocessing*, data memasuki alur persiapan yang berbeda sesuai dengan skenario eksperimen yang telah dirancang. Perbedaan alur ini mencakup proses pelabelan dan ekstraksi fitur yang disesuaikan untuk masing-masing model. Gambar 3.4 memvisualisasikan dua jalur paralel dalam tahap persiapan data ini.



Gambar 3.4. Diagram Alur Proses Pelabelan dan Ekstraksi Fitur

D Skenario Dataset dan Pelabelan Sentimen

Penelitian ini menerapkan dua strategi persiapan data yang berbeda untuk menguji dua filosofi pemodelan yang kontras:

1. **Dataset A (*Pseudo-Labeled* untuk Eksperimen LSTM):** Terdiri dari ~5.000 *tweet* yang dilabeli secara otomatis menggunakan model pra-terlatih `W11wo/indonesian-roberta-base-sentiment-classifier`. Skenario ini mensimulasikan kondisi di mana volume data besar lebih diutamakan

daripada akurasi label yang sempurna.

2. **Dataset B (*Manual-Labeled* untuk Eksperimen IndoBERT):** Terdiri dari 529 *tweet* yang dilabeli secara manual oleh peneliti. Skenario ini merepresentasikan pendekatan di mana kualitas data (*ground truth*) lebih diutamakan daripada kuantitas, yang ideal untuk proses *fine-tuning*.

Untuk mengatasi potensi subjektivitas dalam proses pelabelan manual, serangkaian pedoman (*guideline*) yang jelas ditetapkan sebelum proses pelabelan dimulai. Pedoman ini berfungsi sebagai acuan untuk memastikan konsistensi dalam memberikan label pada setiap *tweet*:

1. **Sentimen Positif:** Diberikan pada *tweet* yang secara eksplisit mengandung kata-kata optimisme (misalnya "kuat", "naik", "beli", "peluang"), atau menyatakan keyakinan bahwa pasar akan pulih. Contoh: *"Waktunya serok saham blue chip nih mumpung IHSG lagi diskon."*

2. **Sentimen Negatif:** Diberikan pada *tweet* yang secara eksplisit mengandung kata-kata kekhawatiran atau kepanikan (misalnya "turun", "anjlok", "rugi", "krisis"), atau menyatakan emosi negatif terhadap kondisi pasar. Contoh: *"IHSG anjlok lagi, investor asing terus jualan."*

3. **Sentimen Netral:** Diberikan pada *tweet* yang bersifat informatif atau faktual, seperti laporan harga penutupan, kutipan berita tanpa opini, atau pertanyaan yang tidak mengandung emosi. Contoh: *"IHSG hari ini ditutup di level 6.200."*

Pendekatan terstruktur ini dilakukan untuk meningkatkan objektivitas dan reliabilitas dari Dataset B yang menjadi *gold standard* dalam penelitian ini.

E Ekstraksi Fitur dan Pembagian Data

Setelah teks bersih dan berlabel, data diubah menjadi format numerik. Untuk Model Bi-LSTM, digunakan metode *Word Embedding* dengan `Tokenizer` Keras. Untuk Model IndoBERT, digunakan *Tokenizer* bawaan dari model pra-terlatih.

Untuk mengevaluasi kinerja dan sensitivitas model, penelitian ini menerapkan dua skenario pembagian data yang berbeda:

1. **Skenario Uji Utama (80:20):** Dataset dibagi menjadi **80% data latih dan 20% data uji**. Rasio ini merupakan praktik umum dalam pemodelan *machine learning* dan menjadi dasar perbandingan utama antara model Bi-LSTM dan IndoBERT.
2. **Skenario Uji Sensitivitas (70:30):** Sebagai pengujian tambahan, dataset juga dibagi menjadi **70% data latih dan 30% data uji**. Skenario ini bertujuan untuk mengamati bagaimana kinerja model berubah ketika dihadapkan pada jumlah data latih yang sedikit lebih kecil dan data uji yang lebih besar.

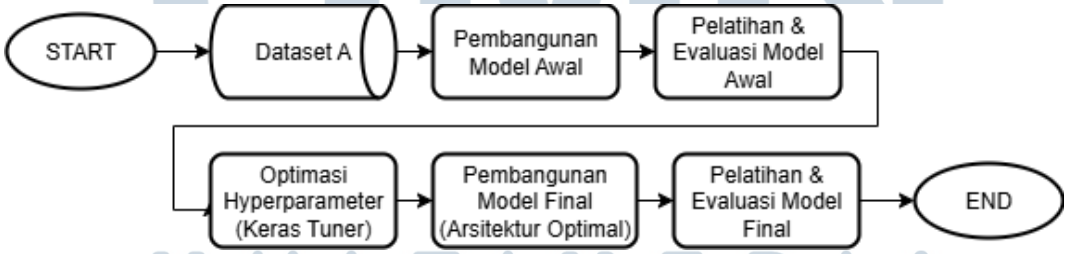
Pada kedua skenario, metode *stratify* diterapkan saat pembagian untuk memastikan proporsi setiap kelas sentimen (positif, negatif, netral) tetap sama baik di data latih maupun data uji.

3.3.3 Pemodelan Komparatif

Tahap pemodelan adalah inti dari studi komparatif ini, di mana dua arsitektur *deep learning* dibangun, dilatih, dan dioptimalkan.

A Model Bidirectional LSTM

Eksperimen pertama membangun model Bi-LSTM dari awal menggunakan **Dataset A**. Gambar 3.5 menyajikan gambaran keseluruhan dari alur kerja eksperimen untuk model Bi-LSTM, mulai dari pembangunan model awal hingga evaluasi model final setelah melalui proses optimasi.



Gambar 3.5. Diagram Alur Proses Eksperimen Model Bi-LSTM

A.1 Pembangunan dan Pelatihan Model Awal.

Sebagai *baseline*, sebuah model Bi-LSTM dibangun dengan arsitektur yang ditentukan secara manual (Cuplikan Kode 3.3). Untuk menangani

ketidakseimbangan kelas, teknik *class weight* diterapkan. Proses pelatihan dikontrol menggunakan *callbacks* seperti `EarlyStopping` dan `ModelCheckpoint`.

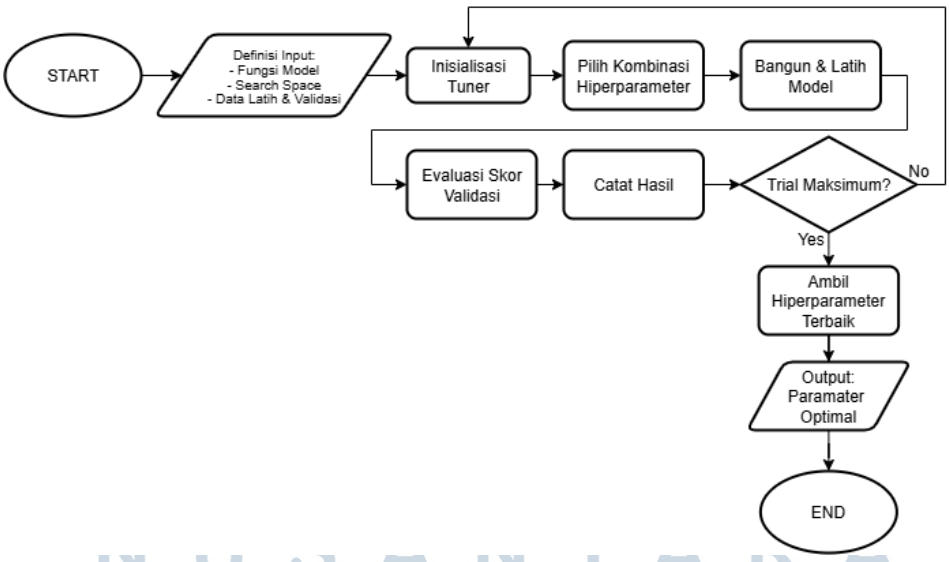
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense,
    Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping,
    ModelCheckpoint, ReduceLROnPlateau
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Definisi arsitektur model
model = Sequential([
    Embedding(vocab_size, 128, input_length=30),
    Bidirectional(LSTM(64, return_sequences=True,
    recurrent_dropout=0.3)),
    Dropout(0.4),
    Bidirectional(LSTM(32, recurrent_dropout=0.3)),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=regularizers.
    l2(0.001)),
    Dropout(0.3),
    Dense(3, activation='softmax')
])

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

# Implementasi Class Weight
y_train_labels = np.argmax(y_train, axis=1)
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_labels),
    y=y_train_labels
)
class_weight_dict = dict(enumerate(class_weights))

# Definisikan callbacks
early_stop = EarlyStopping(monitor='val_accuracy', patience=5,
    restore_best_weights=True)
```

```
checkpoint = ModelCheckpoint('best_lstm_model.h5', monitor='
    val_accuracy', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
    patience=3, min_lr=1e-5)

# Latih model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop, checkpoint, reduce_lr],
    class_weight=class_weight_dict
)
```

Kode 3.3: Pembangunan dan Pelatihan Model Bi-LSTM Awal

A.2 Optimasi Hiperparameter.

Untuk menemukan konfigurasi yang lebih optimal, dilakukan proses *hyperparameter tuning*. Proses ini secara otomatis menguji berbagai kombinasi arsitektur dan parameter pelatihan untuk menemukan yang terbaik. Gambar 3.6 mengilustrasikan logika iteratif dari proses pencarian ini.



Gambar 3.6. Diagram Alur Umum Proses Optimasi Hiperparameter

Proses optimasi untuk model Bi-LSTM secara spesifik dilakukan menggunakan `Keras Tuner` dengan strategi `RandomSearch`. Ruang pencarian

didefinisikan untuk menguji berbagai kombinasi arsitektur dan hiperparameter (Cuplikan Kode 3.4).

```
import keras_tuner as kt
from tensorflow import keras
from tensorflow.keras import layers

def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Embedding(
        input_dim=vocab_size,
        output_dim=hp.Choice('embedding_dim', [64, 128, 256]),
        input_length=30
    ))
    model.add(layers.Bidirectional(layers.LSTM(
        units=hp.Choice('lstm_units_1', [32, 64, 128]),
        recurrent_dropout=hp.Float('recurrent_dropout_1', 0.2,
    0.5),
        return_sequences=True
    )))
    model.add(layers.Dropout(hp.Float('dropout_1', 0.2, 0.5)))
    model.add(layers.Bidirectional(layers.LSTM(
        units=hp.Choice('lstm_units_2', [32, 64]),
        recurrent_dropout=hp.Float('recurrent_dropout_2', 0.2,
    0.5)
    )))
    model.add(layers.Dropout(hp.Float('dropout_2', 0.2, 0.5)))
    model.add(layers.Dense(
        units=hp.Choice('dense_units', [32, 64, 128]),
        activation='relu'
    ))
    model.add(layers.Dropout(hp.Float('dropout_3', 0.2, 0.5)))
    model.add(layers.Dense(3, activation='softmax'))

    model.compile(
        optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',
    [1e-3, 5e-4, 1e-4])),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

tuner = kt.RandomSearch(
    build_model,
```

```
    objective="val_accuracy",
    max_trials=10,
    executions_per_trial=1,
    directory="keras_tuner_dir",
    project_name="ihsg_sentiment"
)
tuner.search(X_train, y_train, epochs=10, validation_split=0.2)
```

Kode 3.4: Definisi Ruang Pencarian untuk Keras Tuner

A.3 Pelatihan Model Final.

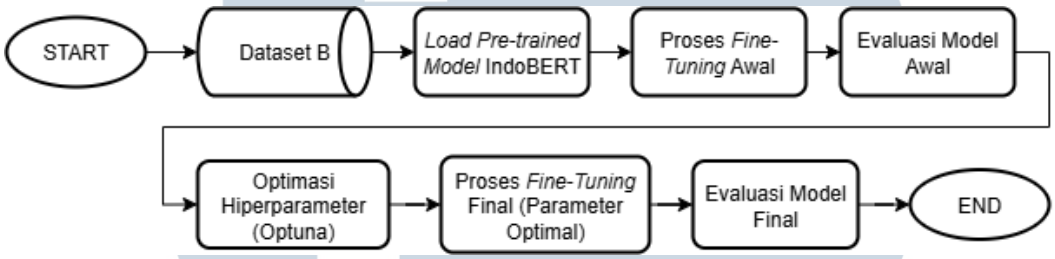
Arsitektur final model Bi-LSTM dibangun berdasarkan kombinasi hiperparameter terbaik yang ditemukan dari proses *tuning* (dirangkum pada Tabel 3.2) dan dilatih kembali pada data latih untuk mendapatkan model final yang akan dievaluasi.

Tabel 3.2. Arsitektur Final Model Bi-LSTM Hasil Optimasi

Layer (tipe)	**Parameters**
Embedding	`output_dim`: 64, `input_length`: 30
Bidirectional (LSTM)	`units`: 64, `recurrent_dropout`: 0.3
Dropout	`rate`: 0.3
Bidirectional (LSTM)	`units`: 64, `recurrent_dropout`: 0.2
Dropout	`rate`: 0.4
Dense	`units`: 128, `activation`: 'relu', `kernel_regularizer`: L2(0.001)
Dropout	`rate`: 0.3
Dense (Output)	`units`: 3, `activation`: 'softmax'
Compiler	`Optimizer`: Adam(lr=0.001), `Loss`: 'categorical_crossentropy'

B Model IndoBERT (Fine-Tuning)

Eksperimen kedua menerapkan pendekatan *transfer learning* dengan melakukan *fine-tuning* pada model pra-terlatih menggunakan **Dataset B**. Alur kerja eksperimen untuk IndoBERT, yang serupa dengan Bi-LSTM namun dengan langkah-langkah spesifik untuk model Transformer, divisualisasikan pada Gambar 3.7.



Gambar 3.7. Diagram Alur Proses Eksperimen Model IndoBERT

B.1 Persiapan Data dan Pelatihan Awal.

Data berlabel manual dikonversi ke format `Dataset` Hugging Face. Proses *fine-tuning* awal dilakukan untuk menetapkan kinerja dasar menggunakan kelas `Trainer` (Cuplikan Kode 3.5).

```
from transformers import AutoTokenizer,
    AutoModelForSequenceClassification, TrainingArguments, Trainer,
     DataCollatorWithPadding
from datasets import Dataset
import pandas as pd
from sklearn.model_selection import train_test_split
import evaluate
import numpy as np

# 1. Load dan Persiapkan Data
train_file = "/content/labeled_tweets_ihsg_500.csv"
df = pd.read_csv(train_file)
mapping = {'pos': 2, 'neg': 0, 'neu': 1}
df["label"] = df["sentiment_label"].map(mapping)
df = df.dropna(subset=["label"])
df["label"] = df["label"].astype(int)
train_df, test_df = train_test_split(df, test_size=0.2, stratify=
    df["label"], random_state=42)

```

```
# 2. Membuat Dataset Huggingface
train_ds = Dataset.from_pandas(train_df[["full_text", "label"]])
test_ds = Dataset.from_pandas(test_df[["full_text", "label"]])

# 3. Model & Tokenizer
model_name = "Wllwo/indonesian-roberta-base-sentiment-classifier"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=3)

# 4. Preprocess
def preprocess_function(examples):
    return tokenizer(examples["full_text"], truncation=True,
    padding=True, max_length=128)
train_ds = train_ds.map(preprocess_function, batched=True)
test_ds = test_ds.map(preprocess_function, batched=True)
train_ds = train_ds.remove_columns(["full_text"])
test_ds = test_ds.remove_columns(["full_text"])
train_ds = train_ds.rename_column("label", "labels")
test_ds = test_ds.rename_column("label", "labels")
train_ds.set_format("torch")
test_ds.set_format("torch")

# 5. Metrics
accuracy_metric = evaluate.load("accuracy")
f1_metric = evaluate.load("f1")
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    accuracy = accuracy_metric.compute(predictions=predictions,
    references=labels)['accuracy']
    f1 = f1_metric.compute(predictions=predictions, references=
    labels, average='weighted')['f1']
    return {"accuracy": accuracy, "f1": f1}

# 6. Training Arguments
training_args = TrainingArguments(
    output_dir="./results_initial",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
```

```
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
)

# 7. Trainer
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# 8. Training
trainer.train()
```

Kode 3.5: Pelatihan Model IndoBERT Awal

B.2 Optimasi Hiperparameter.

Untuk meningkatkan performa, dilakukan pencarian hiperparameter menggunakan *framework* Optuna yang terintegrasi dengan metode `hyperparameter_search` dari `Trainer` (Cuplikan Kode 3.6).

```
import optuna

def model_init():
    return AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=3)

def hp_space(trial):
    return {
        "learning_rate": trial.suggest_float("learning_rate", 1e
    -5, 5e-5, log=True),
        "num_train_epochs": trial.suggest_int("num_train_epochs",
    2, 5),
        "per_device_train_batch_size": trial.suggest_categorical("
    per_device_train_batch_size", [8, 16, 32]),
        "weight_decay": trial.suggest_float("weight_decay", 0.0,
    0.1),
```

```
    }

trainer_for_tuning = Trainer(
    model_init=model_init,
    args=TrainingArguments(output_dir="./tuning_results"),
    train_dataset=train_ds,
    eval_dataset=test_ds,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
    data_collator=data_collator,
)

best_run = trainer_for_tuning.hyperparameter_search(
    direction="maximize",
    backend="optuna",
    hp_space=hp_space,
    n_trials=5
)
```

Kode 3.6: Definisi Ruang Pencarian dan Eksekusi Hyperparameter Search

B.3 Pelatihan Model Final.

Model final dilatih kembali menggunakan kombinasi hiperparameter terbaik yang ditemukan dari proses optimasi, yaitu *learning rate* `2.83e-05`, *batch size* `8`, dan *weight decay* `0.023` selama `3` epoch.

3.3.4 Metode Evaluasi

Setelah proses pelatihan kedua model diselesaikan, evaluasi kuantitatif dilakukan dengan memanfaatkan Confusion Matrix berukuran 3x3. Berdasarkan matriks tersebut, performa masing-masing model diukur dan dibandingkan menggunakan sejumlah metrik evaluasi, yaitu Akurasi, Presisi, Recall, dan F1-Score. Di antara keempat metrik tersebut, F1-Score digunakan sebagai indikator utama mengingat adanya ketidakseimbangan distribusi kelas dalam dataset yang dianalisis.