

BAB 4

HASIL DAN DISKUSI

4.1 Spesifikasi Sistem

Proses pengembangan dan eksperimen dalam penelitian ini didukung oleh kombinasi perangkat keras dan perangkat lunak dengan spesifikasi sebagai berikut:

4.1.1 Perangkat Keras

1. Processor (CPU): AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx
2. Graphics Card (GPU): NVIDIA GeForce GTX 1650 (untuk *pre-processing*) dan T4-GPU (via Google Colab untuk pelatihan)
3. RAM: 16 GB
4. Penyimpanan (SSD): 512 GB

4.1.2 Perangkat Lunak

1. Sistem Operasi: Windows 11
2. Lingkungan Pengembangan: Visual Studio Code untuk tahap *pre-processing* data, dan Google Colaboratory untuk tahap pelatihan (*fine-tuning*) model.
3. Pustaka Utama: Pengembangan model sangat bergantung pada ekosistem Python dengan beberapa pustaka kunci, antara lain:
 - (a) pandas: Untuk manipulasi dan analisis data.
 - (b) numpy: Untuk operasi numerik.
 - (c) transformers: Pustaka utama dari Hugging Face yang menyediakan model IndoBERT, *tokenizer*, dan *framework* pelatihan (Trainer, TrainingArguments).
 - (d) torch: Sebagai *backend deep learning* yang digunakan oleh pustaka transformers.
 - (e) scikit-learn: Untuk pembagian data dan kalkulasi metrik evaluasi (misalnya *accuracy_score*, *confusion_matrix*).
 - (f) matplotlib dan seaborn: Untuk visualisasi data dan hasil evaluasi.

4.2 Deskripsi Dataset dan Preprocessing

Dalam pengembangan model deteksi berita hoaks bahasa Indonesia, dibutuhkan dataset yang berkualitas untuk proses pelatihan dan evaluasi model. Bagian ini menjelaskan tentang dataset yang digunakan dalam proses pelatihan, sumber yang digunakan untuk memperoleh dataset, dan metode yang digunakan dalam melakukan *pre-processing* dataset agar meningkatkan kualitas data yang digunakan oleh model.

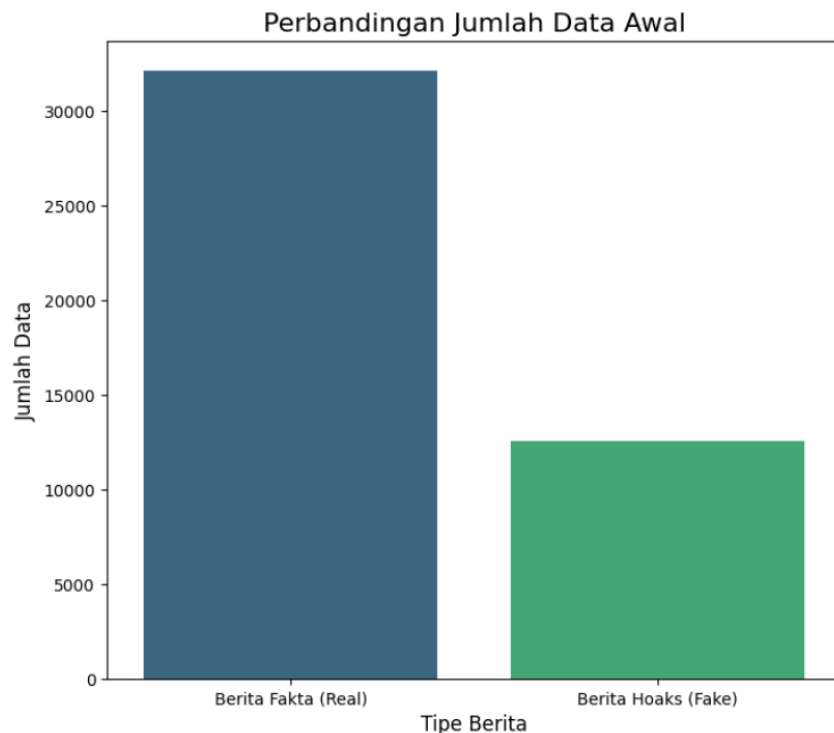
4.2.1 Dataset

Dataset yang digunakan dalam penelitian ini merupakan gabungan dari dua sumber utama untuk menciptakan korpus yang seimbang antara berita fakta dan hoaks.

1. Data Berita Fakta: Data ini bersumber dari dataset publik "Indonesian News Dataset" yang tersedia di platform Kaggle. Dataset mentah ini berisi 32.127 artikel berita dari berbagai media terkemuka di Indonesia.
2. Data Berita Hoaks: Data ini diperoleh melalui metode *web scraping* dari situs TurnBackHoax.id. Proses ini dirancang khusus untuk mengumpulkan artikel klarifikasi hoaks. Implementasi teknis dan kode sumber lengkap untuk skrip web scraping ini disajikan pada Lampiran 3.

Sebelum dilakukan balancing, terdapat disparitas jumlah yang signifikan antara kedua kelas, seperti yang ditunjukkan pada Gambar 4.1. Ketidakseimbangan ini menjadi justifikasi utama untuk melakukan tahap balancing pada proses *pre-processing* selanjutnya.

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A



Gambar 4.1. Perbandingan Data Awal

4.2.2 Proses dan Hasil Preprocessing

Untuk memastikan kualitas dan konsistensi data, serangkaian langkah *pre-processing* dilakukan secara sistematis. Proses ini tidak hanya membersihkan noise, tetapi juga mentransformasi data ke dalam format yang seragam dan optimal untuk dianalisis oleh model IndoBERT.

A Pemeriksaan dan Validasi Data Awal

Tahap paling awal dalam *pre-processing* adalah pemeriksaan dan validasi data. Langkah ini penting untuk memastikan kualitas akan dataset yang telah dikumpulkan. Pemeriksaan ini berfokus pada dua aspek utama: identifikasi nilai kosong (NaN) dan deteksi entri data yang duplikat. Keberadaan data yang tidak valid seperti ini dapat memengaruhi proses pelatihan, menyebabkan bias pada model, atau bahkan menimbulkan error komputasi. Proses pemeriksaan dan validasi dataset dapat dilihat pada Kode 4.1.

```
1 import re
2 import pandas as pd
```

```

3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from dateutil import parser
6 from sklearn.model_selection import train_test_split
7 from collections import Counter
8
9 # Load the dataset
10 df_fake = pd.read_csv('Dataset_Indo/Dataset_Fake/FakeNews_dataset.
    csv')
11 df_real = pd.read_csv('Dataset_Indo/Dataset_Real/RealNews_dataset.
    csv')
12
13 print("Fake News NaN Value: ",df_fake.isnull().sum())
14 print("Fake News Dup Value: ",df_fake.duplicated().sum())
15 print("Real News NaN value: ",df_real.isnull().sum())
16 print("Real News Dup value: ",df_real.duplicated().sum())
17 df_fake.dropna(inplace=True)
18 df_fake.drop_duplicates(inplace=True)
19 df_real.dropna(inplace=True)
20 df_real.drop_duplicates(inplace=True)
21
22 print("\n--- Rata-rata (Mean) Panjang Kata ---")
23 df_real_count = df_real['content'].apply(lambda x: len(str(x).
    split()))
24 df_fake_count = df_fake['FullText'].apply(lambda x: len(str(x).
    split()))
25
26 print(f"Rata-rata panjang kata untuk Berita Fakta: {df_real_count.
    mean()} kata")
27 print(f"Rata-rata panjang kata untuk Berita Hoaks: {df_fake_count.
    mean()} kata")

```

Kode 4.1: Pemeriksaan dan Pembersihan Data Tidak Valid

Hasil menunjukkan bahwa tidak ditemukan adanya nilai kosong maupun data duplikat pada kedua dataset. Hal ini mengindikasikan bahwa tidak ada baris data yang perlu dihapus dan dapat dilanjutkan ke tahap berikutnya. Dilakukan juga pengecekan untuk panjang rata-rata kata dalam berita fakta dan berita hoaks, hasilnya dapat dilihat pada Tabel 4.1.

Tabel 4.1. Perbandingan Rata-rata Panjang Kata per Kategori Berita

| Label Berita | Rata-rata Panjang Kata |
|--------------|------------------------|
| Berita Fakta | 320.37 kata |
| Berita Hoaks | 273.41 kata |

B Unifikasi Format Kolom Dataset

Setelah data dari kedua sumber berhasil dimuat, tahap *pre-processing* dilanjutkan dengan unifikasi atau penyeragaman format kolom. Langkah ini krusial karena kedua dataset awal memiliki struktur, nama kolom, dan jumlah kolom yang sangat berbeda. Penyeragaman format bertujuan untuk memastikan kedua dataset memiliki struktur yang identik. Hasil dari implementasian dapat dilihat pada Kode 4.2.

```

1 # Deleting unnecessary columns in news dataset
2 delete_columns_news = ["id", "image", "url", "embedding", "
    created_at", "updated_at", "summary"]
3 columns = ['Title', 'Timestamp', 'FullText', 'source', 'label']
4
5 df_real.drop(delete_columns_news, axis=1, inplace=True)
6
7 # Renaming columns
8 df_real = (
9     df_real
10     .rename(columns={
11         'title': 'Title',
12         'date': 'Timestamp',
13         'content': 'FullText'
14     })
15 )
16 df_real['label'] = 0
17
18 df_fake['source'] = 'turnbackhoax'
19 df_fake['label'] = 1
20
21 # Ensuring columns are in the same order
22 df_real = df_real[columns]
23 df_fake = df_fake[columns]
24
25 # Verification step
26 df_real.info()
```

```
27 df_fake.info()
```

Kode 4.2: Skrip untuk Unifikasi Struktur Dataset

Seperti yang terlihat pada Kode 4.2, setelah proses dieksekusi, kedua dataset—berita fakta dan hoaks—kini memiliki lima kolom yang sama persis: Title, Timestamp, FullText, source, dan label. Kemudian dilakukan pengecekan untuk memastikan kedua dataset telah sama dan selaras. Proses dapat dilihat pada Tabel 4.2 dan hasil pada Tabel 4.3.

Tabel 4.2. Format Tabel sebelum Unifikasi

| Dataset Berita Fakta | Dataset Berita Hoaks |
|----------------------|----------------------|
| id | Title |
| source | Timestamp |
| title | FullText |
| image | |
| url | |
| content | |
| date | |
| embedding | |
| created_at | |
| updated_at | |
| summary | |

Tabel 4.3. Contoh Format Tabel Setelah Unifikasi untuk Dataset Fakta dan Hoaks

| Title | Timestamp | FullText | source | label |
|----------------|----------------|----------------|--------|-------|
| Depo Plumpu... | 2023-03-04 ... | TEMPO.CO, J... | tempo | 0 |
| Jokowi Peri... | 2023-03-04 ... | TEMPO.CO, J... | tempo | 0 |
| HNW Menduku... | 2023-03-04 ... | INFO NASION... | tempo | 0 |
| Tim Dokkes ... | 2023-03-04 ... | TEMPO.CO, J... | tempo | 0 |
| Bamsoet Aja... | 2023-03-04 ... | INFO NASION... | tempo | 0 |

C Pembersihan Sumber Berita dari Isi Artikel Berita Fakta

Salah satu tantangan dalam *pre-processing* adalah memastikan model belajar dari konten semantik teks, bukan dari pola-pola yang secara tidak sengaja membocorkan informasi label. Pada analisis awal dataset berita fakta, ditemukan bahwa banyak artikel diawali dengan tag sumber berita yang konsisten, seperti “TEMPO.CO, Jakarta -” atau “INFO NASIONAL -”. Untuk mencegah bias ini, sebuah fungsi pembersihan spesifik dirancang untuk menghilangkan semua teks dari awal kalimat hingga tanda hubung (-) pertama. Implementasi teknisnya, yang menggunakan ekspresi reguler (regex), disajikan pada Kode 4.3.

```
1 print("Before removing leading tags:")
2 df_real['FullText'].head(10)
3
4 def remove_leading_tag(text):
5     """
6     Removes the initial web tag (e.g. 'TEMPO.CO, Jakarta -') from
7     a news FullText string,
8     returning only the content after the first dash.
9     """
10    # Convert to string in case of non-str inputs
11    text = str(text)
12    # Use regex to strip everything up to and including the first
13    # dash and following spaces
14    cleaned = re.sub(r'^.*?\s-', '', text)
15    return cleaned
16
17 df_real['FullText'] = df_real['FullText'].apply(remove_leading_tag)
18
19 print("After removing leading tags:")
20 print(df_real['FullText'].head(10))
```

Kode 4.3: Function untuk Penghapusan Tag Sumber Berita

Hasil dari eksekusi skrip pembersihan ini dapat dilihat secara jelas pada Tabel 4.4, yang membandingkan contoh teks sebelum dan sesudah tag sumber dihilangkan.

Tabel 4.4. Perbandingan Contoh Teks Sebelum dan Sesudah Penghapusan Tag Sumber

| Artikel Berita Fakta Sebelum Menghilangkan Tag Sumber | Artikel Berita Fakta Sesudah Menghilangkan Tag Sumber |
|---|---|
| TEMPO.CO, Jakarta - Anggota Komisi VII DPR RI ... | Anggota Komisi VII DPR RI Rofik Hananto menyay... |
| TEMPO.CO, Jakarta - Presiden Joko Widodo atau ... | Presiden Joko Widodo atau Jokowi memerintahkan... |
| INFO NASIONAL - Wakil Ketua MPR RI Dr. H. M. H... | Wakil Ketua MPR RI Dr. H. M. Hidayat Nur Wahid... |

D Ekstraksi Konten dan Pembersihan Spesifik pada Dataset Hoaks

Dataset berita hoaks yang diperoleh dari proses *web scraping* memiliki struktur teks yang mentah dan tidak seragam. Kolom FullText dari TurnBackHoax.id sering kali berisi metadata tambahan seperti [KATEGORI], [SUMBER], dan [REFERENSI] yang disisipkan di antara konten naratif. Keberadaan metadata ini dapat menjadi *noise* atau bahkan sinyal bocoran (data leakage) yang mengganggu proses *fine-tuning* model.

Oleh karena itu, diperlukan serangkaian langkah *pre-processing* khusus untuk dataset ini. Tujuannya adalah untuk mengekstraksi inti dari konten artikel yaitu bagian [Narasi] dan [PENJELASAN] dan membuang semua metadata lainnya. Proses ini dilakukan melalui serangkaian fungsi pembersihan dan ekstraksi menggunakan ekspresi reguler (regex), seperti yang disajikan pada Kode 4.4.

```

1 # Function to clean text for Turnback Hoax dataset
2 def clean_text(text):
3     # Ensure it's a string
4     text = str(text)
5     # Remove URLs
6     text = re.sub(r'http\S+', '', text)
7     # Remove HTML tags
8     text = re.sub(r'<.*?>', '', text)
9     # Remove @ # = symbols
10    text = re.sub(r'[@#=\[\]]', '', text)
11    # Remove single and double quotation marks (including
    typographic variants)
12    text = re.sub(r'["\']', '', text)
13    # Collapse multiple spaces into one
14    text = re.sub(r'\s+', ' ', text).strip()

```



```

15     return text
16
17 def clean_dataframe_text(df):
18     for col in df.select_dtypes(include='object').columns:
19         df[col] = df[col].map(clean_text)
20     return df
21
22 # Apply to dataframes
23 df_fake = clean_dataframe_text(df_fake)
24
25 def remove_before_narasi(text):
26     # Match 'narasi' (any case), optionally followed by a colon
27     # and whitespace
28     match = re.search(r'\b(narasi)\b:?s*', text, re.IGNORECASE)
29     if match:
30         return text[match.end():].lstrip()
31     return text
32
33 df_fake['FullText'] = df_fake['FullText'].apply(
34     remove_before_narasi)
35
36 def remove_penjelasan(text):
37     # Matches only uppercase/lowercase first-letter versions, but
38     # not all-lowercase "penjelasan"
39     return re.sub(r'\b(PENJELASAN|Penjelasan)\b:?s*', '', text)
40
41 df_fake['FullText'] = df_fake['FullText'].apply(remove_penjelasan)
42
43 def remove_referensi(text):
44     # Remove all occurrences of the word 'referensi' (case-
45     # insensitive), with optional colon and trailing spaces
46     return re.sub(r'\b(referensi)\b:?s*', '', text, flags=re.
47         IGNORECASE)
48
49 df_fake['FullText'] = df_fake['FullText'].apply(remove_referensi)

```

Kode 4.4: Pipeline Fungsi untuk Pembersihan dan Ekstraksi Teks Hoaks

Proses ekstraksi pada Kode 4.4 sengaja dilakukan secara bertahap. Pertama, teks dibersihkan dari karakter umum yang mengganggu. Kemudian, konten diekstraksi dengan mengambil semua teks setelah tag NARASI. Selanjutnya, tag PENJELASAN yang biasanya muncul di tengah dihapus. Terakhir, seluruh bagian REFERENSI di akhir artikel dihilangkan. Pendekatan berurutan ini memastikan

ekstraksi konten berjalan akurat tanpa menghapus bagian teks yang penting secara berlebihan. Hasil transformasi ini dapat dilihat pada Tabel 4.5.

Tabel 4.5. Perbandingan Berita Hoaks Sebelum dan Sesudah Ekstraksi

| Sebelum Ekstraksi | Sesudah Ekstraksi |
|---|--|
| <p>Hasil periksa fakta ... [KATEGORI]: Konten yang Menyesatkan. [SUMBER]: Akun Facebook ... [NARASI]: TEPAT PAGI INI, GIBRAN & JOKOWI TANDA TANGAN ... [PENJELASAN]: Akun Grauda News ... mengunggah video yang berjudul ... [REFERENSI]: https://seword.com/politik/gibran...</p> | <p>TEPAT PAGI INI, GIBRAN & JOKOWI TANDA TANGAN ... Akun Grauda News ... mengunggah video yang berjudul ...</p> |

E Pembersihan Teks Global dan Normalisasi

Setelah kedua dataset melalui tahap pembersihan spesifik, langkah *pre-processing* selanjutnya adalah pembersihan teks secara global. Tahap ini diterapkan pada keseluruhan data, baik berita fakta maupun hoaks, dengan tujuan untuk menormalisasi teks. Proses ini sangat penting karena model Transformer seperti IndoBERT sensitif terhadap variasi kecil dalam teks. Kode 4.5 menyajikan fungsi inti yang digunakan untuk melakukan pembersihan ini.

```

1 #Function for general text cleaning
2 def clean_text(text):
3     # Ensure it's a string
4     text = str(text)
5     # Remove URLs
6     text = re.sub(r'http\S+', '', text)
7     # Remove HTML tags
8     text = re.sub(r'<.*?>', '', text)
9     # Remove @ # = symbols
10    text = re.sub(r'[@#=\[\]]', '', text)
11    # Remove single and double quotation marks (including
    typographic variants)
12    text = re.sub(r'["\']', '', text)
13    # Remove Emojis
14    emoji_pattern = re.compile(

```

```

15     "["
16     "\U0001F600-\U0001F64F" # Emoticons
17     "\U0001F300-\U0001F5FF" # Symbols & pictographs
18     "\U0001F680-\U0001F6FF" # Transport & map symbols
19     "\U0001F1E0-\U0001F1FF" # Flags
20     "\U00002700-\U000027BF" # Dingbats
21     "\U0001F900-\U0001F9FF" # Supplemental Symbols and
Pictographs
22     "\U00002600-\U000026FF" # Misc symbols
23     "\U00002B00-\U00002BFF" # Arrows and shapes
24     "]" + ",
25     flags=re.UNICODE
26 )
27 text = emoji_pattern.sub(r'', text)
28 # Collapse multiple spaces into one
29 text = re.sub(r'\s+', ' ', text).strip()
30 # Convert to lowercase
31 text = text.lower()
32 return text
33
34 def clean_dataframe_text(df):
35     # Iterate only over columns with object dtype (usually text
columns)
36     for col in df.select_dtypes(include='object').columns:
37         df[col] = df[col].map(clean_text)
38     return df
39
40 # Apply to dataframes
41 df_fake = clean_dataframe_text(df_fake)
42 df_real = clean_dataframe_text(df_real)

```

Kode 4.5: Fungsi untuk Pembersihan dan Normalisasi Teks Global

Fungsi pada Kode 4.5 diterapkan pada seluruh data tekstual. Hasil dari normalisasi ini adalah dataset yang lebih bersih dan seragam, di mana format teks dari berita fakta dan hoaks kini konsisten. Tabel 4.6 menunjukkan contoh dari efek proses pembersihan ini pada sebuah kalimat.

Tabel 4.6. Perbandingan Teks Sebelum dan Sesudah Proses Pembersihan Global

| Teks Sebelum Pembersihan | Teks Sesudah Pembersihan |
|--|---|
| Foto “1 korban dlm pengawasan @Covid-19”. Menteri Budi Karya dinyatakan POSITIF!! https://turnbackhoax.i@/2020/... | foto 1 korban dlm pengawasan covid 19. menteri budi karya dinyatakan positif!! |

F Balancing dan Merging Dataset Final

Tahap terakhir dalam alur kerja *pre-processing* adalah balancing dan merging dataset. Langkah ini sangat penting karena, seperti yang ditunjukkan pada analisis awal, terdapat ketidakseimbangan kelas yang signifikan antara jumlah berita fakta (32.127) dan berita hoaks (12.542). Melatih model pada dataset yang tidak seimbang dapat menyebabkan bias, di mana model cenderung lebih akurat dalam memprediksi kelas mayoritas (berita fakta) dibandingkan kelas minoritas.

Implementasi teknisnya dapat dilihat pada Kode 4.6. Pertama, sumber berita 'JawaPos' dihapuskan karena jumlah artikelnya yang sangat sedikit (1.010) dibandingkan dengan sumber lain, yang akan menyulitkan proses sampling yang seimbang. Kemudian, dari enam sumber berita yang tersisa, diambil sampel acak sebanyak 2.100 artikel dari masing-masing sumber. Penggunaan `random_state=42` memastikan bahwa proses sampling ini dapat direproduksi.

```

1 # Remove 'jawapos' source from the real news dataset
2 df_filtered = df_real[df_real['source'] != 'jawapos']
3
4 # For each source, sample 2100 rows randomly (or fewer if not
   enough)
5 sampled_dfs = []
6
7 for source_name, group in df_filtered.groupby('source'):
8     sample_size = min(len(group), 2100)
9     sampled = group.sample(n=sample_size, random_state=42) #
   fixed seed for reproducibility
10    sampled_dfs.append(sampled)
11
12 # Combine all sampled groups into one DataFrame
13 df_sampled = pd.concat(sampled_dfs).reset_index(drop=True)
14 df_real = df_sampled
15

```

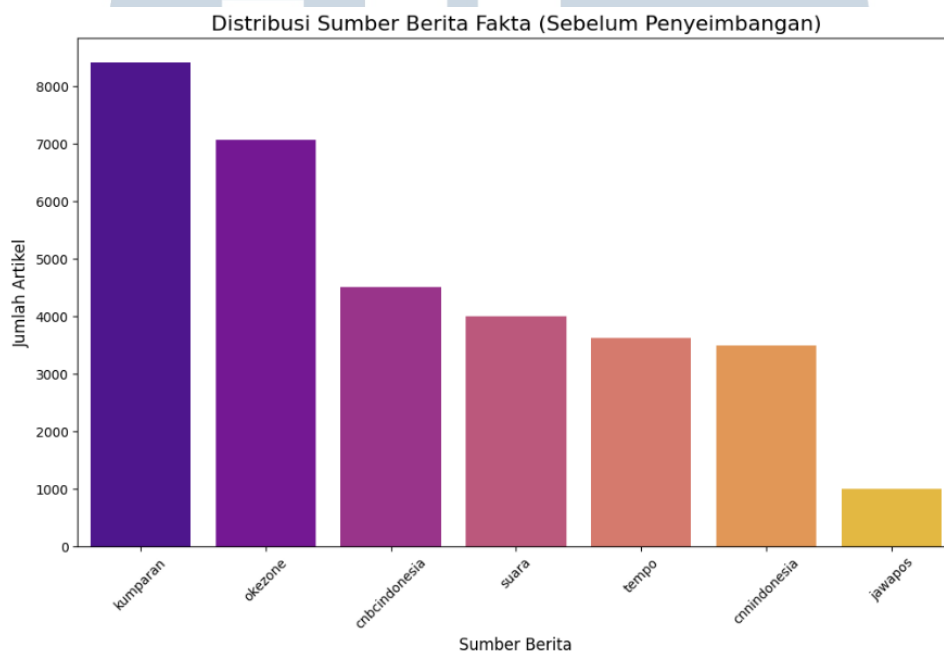
```

16 # Check result
17 print(df_real['source'].value_counts())
18 print("Total rows after sampling:", len(df_real))

```

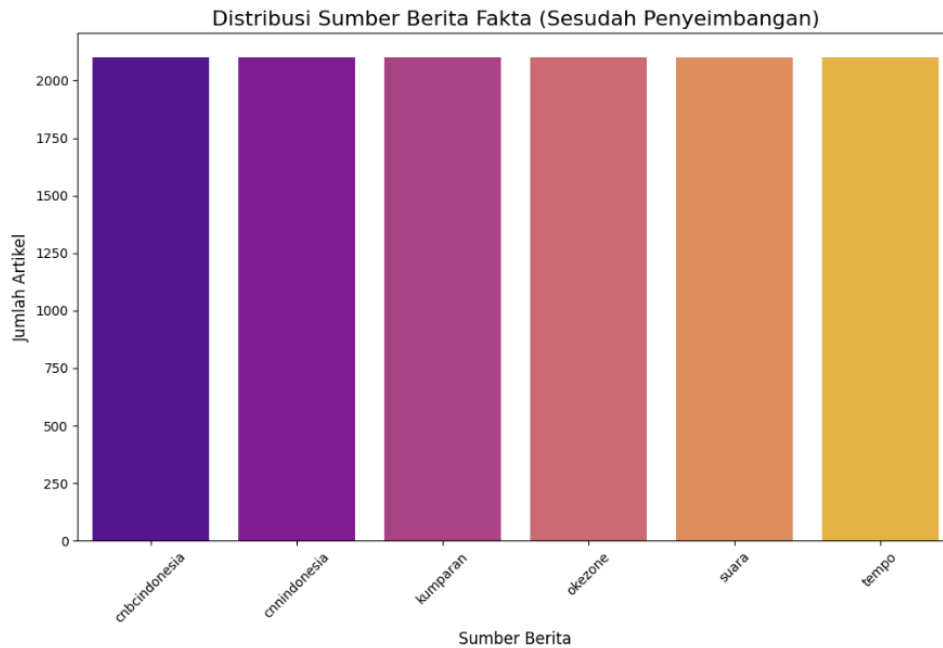
Kode 4.6: Penyamaan Value Berita Fakta

Hasil dari proses balancing ini sangat efektif, menghasilkan dataset berita fakta baru yang berjumlah 12.600 artikel. Perubahan distribusi sumber berita sebelum dan sesudah proses sampling dapat dilihat secara visual pada Gambar 4.2 dan Gambar 4.3.



Gambar 4.2. Distribusi Sumber Berita Fakta (Sebelum balancing)

UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 4.3. Distribusi Sumber Berita Fakta (Sesudah balancing)

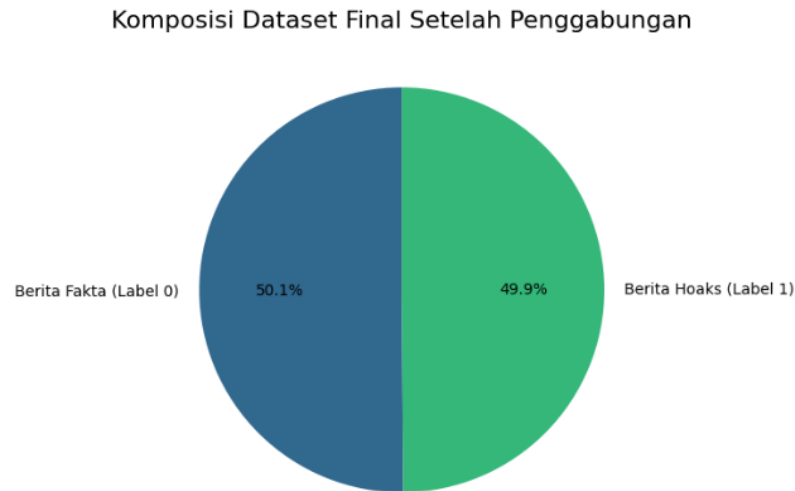
Dengan dataset berita fakta yang telah seimbang, langkah selanjutnya adalah menggabungkannya dengan dataset berita hoaks dapat dilihat pada Kode 4.7. Setelah digabungkan, keseluruhan dataset diacak (*shuffled*). Proses ini penting untuk memastikan bahwa saat pelatihan, model melihat data dalam urutan yang acak pada setiap *epoch*.

```

1 #Combine the datasets
2 df_datasets = pd.concat([df_real, df_fake], ignore_index=True)
3 # Shuffle the combined dataset
4 df_datasets = df_datasets.sample(frac=1, random_state=42).
   reset_index(drop=True)
5
6 df_datasets.info()
```

Kode 4.7: Mengabungkan Dataset

Hasilnya adalah dataset final yang siap digunakan, dengan komposisi yang sangat seimbang antara kedua kelas, seperti yang ditunjukkan pada Gambar 4.4.



Gambar 4.4. Komposisi Dataset Final Setelah merging

Pembagian data dilakukan menggunakan fungsi `train_test_split` dari `scikit-learn` dengan parameter `stratify` berdasarkan kolom label. Stratifikasi memastikan bahwa proporsi kelas fakta dan hoaks tetap terjaga secara seimbang di ketiga set data tersebut. Dalam proses pembagian dataset, dibagi menjadi 3 bagian yaitu dataset train (80%), validation(10% dari dataset train), dan test(20%). Implementasinya dapat dilihat pada Kode 4.8. Dataset kemudian disimpan menjadi dataset baru yang siap digunakan oleh model.

```
1 # Split using train_test_split with stratification on 'label'
2 # Split into train and test (e.g. 80% train, 20% test)
3 df_trainval, df_test = train_test_split(
4     df_datasets,
5     test_size=0.2,
6     random_state=42,
7     stratify=df_datasets['label']
8 )
9
10 # Split train into train and validation (e.g. 10% of original =
    12.5% of 80%)
11 df_train, df_val = train_test_split(
12     df_trainval,
13     test_size=0.125, # 12.5% of 80% = 10% of total
14     random_state=42,
15     stratify=df_trainval['label']
16 )
17
```

```

18 # Output sizes
19 print(f"Train: {len(df_train)}")
20 print(f"Validation: {len(df_val)}")
21 print(f"Test: {len(df_test)}")
22
23 # Save the datasets to CSV files
24 df_train.to_csv('Dataset_Indo/Train.csv', index=False)
25 df_val.to_csv('Dataset_Indo/Validation.csv', index=False)
26 df_test.to_csv('Dataset_Indo/Test.csv', index=False)

```

Kode 4.8: Proses Pembagian Dataset

4.3 Implementasi dan Evaluasi Model

Bagian ini menyajikan tahapan teknis dari implementasi model deteksi berita hoaks, mulai dari persiapan data hingga proses pelatihan dan evaluasi akhir. Pengimplementasian dilakukan secara sistematis untuk memastikan model dapat menghasilkan performa yang akurat dan dapat direproduksi.

4.3.1 Implementasi Model Diteksi Berita Bahasa Indonesia dengan IndoBERT

Pengimplementasian model deteksi dilakukan dengan metode *fine-tuning* pada model IndoBERT yang telah dilatih sebelumnya (*pre-trained*). Proses ini bertujuan untuk mengadaptasi pengetahuan bahasa umum dari IndoBERT agar spesifik pada tugas klasifikasi antara berita hoaks dan fakta. Alur kerja implementasi ini mencakup pemuatan dataset, tokenisasi, konfigurasi pelatihan, eksekusi *fine-tuning*, dan evaluasi akhir.

A Pemuatan Dataset

Langkah pertama dalam pipeline implementasi adalah memuat kembali ketiga set data—latih, validasi, dan uji—yang telah dihasilkan pada tahap *pre-processing*. Setelah dimuat, konten teks (`FullText`) dan labelnya dari setiap set data diekstrak dan disimpan ke dalam variabel list yang terpisah. Struktur data dalam bentuk list ini merupakan format masukan yang paling kompatibel untuk diproses lebih lanjut oleh *tokenizer* dari pustaka `transformers`. Implementasi dari langkah ini disajikan pada Kode 4.9.


```

1 from transformers import (
2     AutoTokenizer,
3     AutoModelForSequenceClassification,
4     TrainingArguments,
5     Trainer,
6     DataCollatorWithPadding
7 )
8 import matplotlib.pyplot as plt
9 import seaborn as sns
10 import numpy as np
11 import pandas as pd
12 import torch
13 from collections import Counter
14 from torch.utils.data import Dataset
15 from sklearn.metrics import (
16     accuracy_score,
17     classification_report,
18     confusion_matrix,
19     roc_auc_score,
20     RocCurveDisplay,
21     PrecisionRecallDisplay,
22     precision_recall_fscore_support
23 )
24
25 df_test = pd.read_csv("dataset/Test.csv")
26 df_train = pd.read_csv("dataset/Train.csv")
27 df_val = pd.read_csv("dataset/Validation.csv")
28
29 #Grab texts & labels
30 train_texts = df_train["FullText"].tolist()
31 train_labels = df_train["label"].tolist()
32
33 val_texts = df_val["FullText"].tolist()
34 val_labels = df_val["label"].tolist()
35
36 test_texts = df_test["FullText"].tolist()
37 test_labels = df_test["label"].tolist()

```

Kode 4.9: Pemuatan dan Persiapan Dataset untuk Pelatihan

Setelah eksekusi Kode 4.9, data teks dan label kini telah siap untuk memasuki tahap selanjutnya, yaitu inisialisasi model, *tokenizer* model, dan tahap proses tokenisasi.

B Konfigurasi Awal Model dan Tokenizer

Setelah dataset siap, langkah selanjutnya adalah menginisialisasi model dan *tokenizer* yang akan digunakan. Penelitian ini memanfaatkan model `indolem/indobert-base-uncased`, sebuah model BERT yang telah melalui tahap *pre-training* pada korpus Bahasa Indonesia yang masif.

Proses inisialisasi dilakukan menggunakan kelas `AutoTokenizer` dan `AutoModelForSequenceClassification` dari pustaka `transformers`, seperti yang ditunjukkan pada Kode 4.10. Kelas `AutoModelForSequenceClassification` secara otomatis memuat arsitektur IndoBERT dan menambahkan sebuah "kepala" klasifikasi di atasnya. Kepala klasifikasi ini dikonfigurasi secara spesifik untuk tugas ini dengan parameter `num_labels=2`, serta pemetaan antara ID dan nama label (`id2label` dan `label2id`).

```
1 # Tokenizer and Model
2 modelname = "indolem/indobert-base-uncased"
3 tokenizer = AutoTokenizer.from_pretrained(modelname)
4
5 model = AutoModelForSequenceClassification.from_pretrained(
6     modelname,
7     num_labels=2,
8     id2label={0: "REAL", 1: "HOAX"},
9     label2id={"REAL": 0, "HOAX": 1},
10 )
11
12 # Freeze base model parameters
13 for name, param in model.base_model.named_parameters():
14     param.requires_grad = False
```

Kode 4.10: Inisialisasi Model IndoBERT dan Pembekuan Lapisan Dasar

Salah satu keputusan metode *fine-tuning* yang digunakan dalam penelitian ini adalah *feature extraction*. Seperti terlihat pada Kode 4.10, penelitian ini menggunakan pendekatan *feature extraction* dengan cara membekukan (*freeze*) seluruh parameter dari 12 lapisan dasar model IndoBERT. Dengan mengatur `param.requires_grad = False`, gradien tidak akan dihitung untuk lapisan-lapisan ini selama proses *backpropagation*. Akibatnya, bobot dari lapisan dasar yang sudah memiliki pemahaman bahasa yang kaya tidak akan diperbarui, dan proses pelatihan hanya akan mengoptimalkan bobot dari kepala klasifikasi yang baru ditambahkan.

Strategi ini dipilih secara spesifik untuk mencegah fenomena *overfitting* yang sangat cepat. Hal ini disebabkan karena model IndoBERT yang sudah sangat kuat dapat dengan mudah "menghafal" dataset penelitian yang relatif kecil dibandingkan dengan korpus data saat *pre-training*. Dengan membekukan lapisan dasar, model dipaksa untuk belajar membedakan hoaks dan fakta berdasarkan fitur linguistik yang diekstrak, bukan dengan menghafal dataset.

C Tokenisasi dan Strukturisasi Dataset untuk Pelatihan

Pada tahapan ini, dataset yang sudah dimuat sebelumnya akan melalui proses tokenisasi. Proses tokenisasi ini memastikan bahwa data yang akan dilatih dengan model sudah sesuai dengan bentuk yang diperlukan model. Proses tokenisasi ini juga menghasilkan "input_ids", "token_type_ids", dan "attention_mask". proses implementasi dapat dilihat pada kode 4.11.

C.1 Tokenisasi Teks

Langkah pertama adalah tokenisasi, yaitu proses mengonversi data teks mentah menjadi representasi numerik. *Tokenizer* yang telah dimuat sebelumnya diterapkan pada setiap set data (latih, validasi, dan uji). Proses ini, seperti yang diimplementasikan pada Kode 4.11, menghasilkan sebuah dictionary yang berisi tiga komponen krusial:

1. `input_ids`: Urutan ID numerik yang merepresentasikan setiap token (kata atau sub-kata) dalam kalimat.
2. `token_type_ids`: Indikator untuk membedakan antar kalimat, yang dalam kasus ini bernilai 0 untuk semua token karena setiap masukan adalah satu sekuens tunggal.
3. `attention_mask`: Larik biner (0 atau 1) yang memberitahu model token mana yang merupakan konten asli (nilai 1) dan mana yang merupakan *padding* (nilai 0).

Parameter `truncation=True` memastikan bahwa teks yang lebih panjang dari panjang maksimal model (512 token) akan dipotong, sedangkan padding akan ditangani pada langkah selanjutnya.

```

1 # Tokenize texts
2 train_encodings = tokenizer(
3     train_texts,
4     truncation=True,
5     max_length=512,
6 )
7
8 val_encodings = tokenizer(
9     val_texts,
10    truncation=True,
11    max_length=512,
12 )
13
14 test_encodings = tokenizer(
15     test_texts,
16     truncation=True,
17     max_length=512,
18 )
19
20 print(train_encodings.keys())
21 print(val_encodings.keys())
22 print(test_encodings.keys())

```

Kode 4.11: Proses Tokenisasi Dataset

Fungsi pada Kode 4.11 diterapkan pada seluruh data teks. Hasil dari tokenisasi ini adalah dataset yang dapat diterima dan dilatih oleh model. Tabel 4.7 menunjukkan contoh dari tokenisasi pada sebuah kalimat.

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A

Tabel 4.7. Perbandingan Teks Sebelum dan Sesudah Tokenisasi

| Teks Awal | Teks Sesudah Tokenisasi | Input ID | Attention Mask |
|---------------------------|-------------------------|----------|----------------|
| | [CLS], | 3, | 1, |
| | para, | 1766, | 1, |
| | ilmuwan, | 6671, | 1, |
| | sedang, | 2447, | 1, |
| | melakukan, | 2063, | 1, |
| para ilmuwan sedang | penelitian, | 3528, | 1, |
| melakukan penelitian hiu | hiu, | 17018, | 1, |
| pelagis untuk LSM Condrik | pela, | 4478, | 1, |
| Tenefire... | ##gis, | 31196, | 1, |
| | untuk, | 1559, | 1, |
| | lsm, | 9889, | 1, |
| | cond, | 16324, | 1, |
| | ##rik, | 3892, | 1, |
| | ten, | 2751, | 1, |
| | ##ef, | 3102, | 1, |
| | ##ire, | 11604, | 1, |

C.2 Strukturisasi dengan Dataset dan Data Collator

Agar data yang telah ditokenisasi dapat diproses secara efisien oleh Trainer Hugging Face, data tersebut perlu dibungkus dalam struktur kelas Dataset dari PyTorch. Seperti yang ditunjukkan pada Kode 4.12, sebuah kelas kustom bernama NewsDataset dibuat untuk tujuan ini. Kelas ini menyediakan antarmuka standar bagi Trainer untuk mengambil sampel data secara efisien selama pelatihan.

Selain itu, dataset akan memanfaatkan DataCollatorWithPadding. Penggunaan *data collator* akan menerapkan *dynamic padding* pada setiap *batch* data sehingga data yang masuk kedalam model hanya akan di-*padding* sesuai dengan panjang sekuens terpanjang di dalam *batch* tersebut.

```

1 # Wrap them in a Dataset
2 class NewsDataset(Dataset):
3     def __init__(self, encodings, labels):
4         self.encodings = encodings
5         self.labels = labels

```

```

6
7     def __len__(self):
8         return len(self.labels)
9
10    def __getitem__(self, idx):
11        item = {k: torch.tensor(v[idx])
12                for k, v in self.encodings.items()}
13        item["labels"] = torch.tensor(self.labels[idx], dtype=
14        torch.long)
15        return item
16
17    train_dataset = NewsDataset(train_encodings, train_labels)
18    val_dataset = NewsDataset(val_encodings, val_labels)
19    test_dataset = NewsDataset(test_encodings, test_labels)
20
21    # Use a data collator to pad dynamically
22    data_collator = DataCollatorWithPadding(tokenizer)

```

Kode 4.12: Strukturisasi Data dengan Kelas Dataset dan Data Collator

D Konfigurasi Metrik Evaluasi dan Argumen Pelatihan

Setelah dataset siap, tahap selanjutnya adalah mendefinisikan konfigurasi untuk proses evaluasi dan *fine-tuning*. Langkah ini melibatkan pembuatan fungsi untuk menghitung metrik performa dan pengaturan seluruh hiperparameter yang akan mengontrol jalannya proses pelatihan.

D.1 Definisi Metrik Evaluasi.

Untuk memonitor performa model secara kuantitatif selama pelatihan, sebuah fungsi kustom bernama `compute_metrics` dibuat, seperti yang terlihat pada Kode 4.13. Fungsi ini akan dipanggil oleh objek `Trainer` setiap kali evaluasi dilakukan pada *validation set*. Fungsi ini menerima keluaran mentah model dan label sebenarnya, kemudian menghitung dan mengembalikan serangkaian metrik evaluasi utama: akurasi, F1-score, *precision*, dan *recall*.

```

1 # Define a simple accuracy metric
2 def compute_metrics(eval_pred):
3     logits, labels = eval_pred
4     preds = np.argmax(logits, axis=-1)

```

```

5     precision, recall, f1, _ = precision_recall_fscore_support(
6     labels, preds, average='binary')
7     acc = accuracy_score(labels, preds)
8     return {
9         'accuracy': acc,
10        'f1': f1,
11        'precision': precision,
12        'recall': recall
13    }

```

Kode 4.13: Fungsi untuk Kalkulasi Metrik Evaluasi

D.2 Konfigurasi Hiperparameter Pelatihan.

Seluruh konfigurasi untuk proses *fine-tuning* didefinisikan dalam sebuah objek `TrainingArguments`, seperti yang disajikan pada Kode 4.14. Beberapa hiperparameter yang diatur dalam penelitian ini meliputi:

1. `num_train_epochs=4`: Model akan dilatih pada keseluruhan *training set* sebanyak empat kali.
2. `learning_rate=3e-5`: Menentukan seberapa besar pembaruan yang dilakukan pada bobot model di setiap langkah.
3. `weight_decay=0.1`: Teknik regularisasi yang membantu mencegah *overfitting*.
4. `lr_scheduler_type="linear"`: Menggunakan penjadwal yang akan mengurangi *learning rate* secara linear dari nilai awal ke 0 selama proses pelatihan.
5. `load_best_model_at_end=True`: Sebuah fitur penting yang memastikan bahwa setelah pelatihan selesai
6. `metric_for_best_model="f1"`: Menentukan bahwa "model terbaik" dipilih berdasarkan nilai F1-score tertinggi.

Terakhir, objek `Trainer` diinisialisasi dengan menyatukan semua komponen yang telah disiapkan: model, argumen pelatihan, dataset, *data collator*, dan fungsi metrik.

```

1 # Prepare training arguments
2 training_args = TrainingArguments(
3     output_dir="indobert-fake-news",
4     per_device_eval_batch_size=32,
5     per_device_train_batch_size=32,
6     num_train_epochs=4,
7     learning_rate=3e-5,
8     lr_scheduler_type="linear",
9     eval_strategy="epoch", # run evaluation at end of each epoch
10    save_strategy="epoch", # checkpoint at each epoch
11    weight_decay=0.1,
12    load_best_model_at_end=True,
13    metric_for_best_model="f1", # Evaluates models on F1,
14    greater_is_better=True,
15    logging_steps=100,
16    fp16=True,
17 )
18
19 # Instantiate the Trainer
20 trainer = Trainer(
21     model=model,
22     args=training_args,
23     train_dataset=train_dataset,
24     eval_dataset=val_dataset,
25     tokenizer=tokenizer,
26     data_collator=data_collator,
27     compute_metrics=compute_metrics,
28 )

```

Kode 4.14: Konfigurasi Training

E Eksekusi Pelatihan dan Evaluasi Akhir Model

Dengan seluruh komponen—dataset, model, tokenizer, dan konfigurasi—telah siap, tahap akhir dari implementasi adalah mengeksekusi proses *fine-tuning* dan melakukan evaluasi final pada data uji.

E.1 Pelaksanaan Proses Fine-Tuning

Proses pelatihan diinisiasi dengan memanggil metode `trainer.train()`. Perintah ini mengorkestrasi seluruh alur kerja pelatihan secara otomatis berdasarkan

TrainingArguments yang telah didefinisikan sebelumnya, implementasi dapat dilihat pada Kode 4.15 dan hasil dari pelatihan dapat dilihat pada Gambar 4.5. Selama proses ini, Trainer akan:

1. Melakukan iterasi pada *training set* sesuai dengan jumlah *epoch* yang ditentukan.
2. Menghitung nilai *loss* dan melakukan *backpropagation* untuk memperbarui bobot pada lapisan klasifikasi.
3. Mengevaluasi performa model pada *validation set* di setiap akhir *epoch*.
4. Menyimpan *checkpoint* model pada setiap *epoch* dan secara otomatis memuat kembali *checkpoint* dengan F1-score terbaik di akhir pelatihan.

Setelah pelatihan selesai, model dan *tokenizer* terbaik akan disimpan ke direktori keluaran untuk penggunaan di masa depan.

```
1 trainer.train()
2 trainer.save_model()
3 tokenizer.save_pretrained(training_args.output_dir)
4
5 best_ckpt = trainer.state.best_model_checkpoint
6 print("Best checkpoint path:", best_ckpt)
```

Kode 4.15: Pelatihan Model dengan Trainer

| Epoch | Training Loss | Validation Loss | Accuracy | F1 | Precision | Recall |
|-------|---------------|-----------------|----------|----------|-----------|----------|
| 1 | 0.419900 | 0.370270 | 0.912127 | 0.914967 | 0.884673 | 0.947410 |
| 2 | 0.300200 | 0.272190 | 0.947913 | 0.948036 | 0.943918 | 0.952191 |
| 3 | 0.253600 | 0.236977 | 0.954672 | 0.954509 | 0.956035 | 0.952988 |
| 4 | 0.252700 | 0.227177 | 0.955467 | 0.955272 | 0.957566 | 0.952988 |

Gambar 4.5. Proses Trainer.train()

E.2 Evaluasi Akhir pada Data Uji.

Setelah model terbaik diperoleh, evaluasi final dilakukan pada *test set*—data yang belum pernah dilihat sama sekali oleh model selama proses pelatihan maupun validasi. Langkah ini memberikan pengukuran performa yang paling objektif.

Metode `trainer.evaluate()` digunakan pada *test_dataset* untuk mendapatkan metrik kuantitatif. Selanjutnya, metode `trainer.predict()`

digunakan untuk mendapatkan prediksi mentah (*logits*) yang diperlukan untuk menghasilkan laporan klasifikasi yang lebih detail dan menghitung nilai ROC-AUC.

```
1 #Evaluation
2 test_metrics = trainer.evaluate(test_dataset)
3 print("Final Test Set Metrics:", test_metrics)
4
5 preds_output = trainer.predict(test_dataset)
6 y_pred = preds_output.predictions.argmax(-1)
7 y_true = test_dataset.labels
8
9 print(classification_report(y_true, y_pred, target_names=["REAL", "
    HOAX"])))
10 cm = confusion_matrix(y_true, y_pred)
11 print("Confusion Matrix:\n", cm)
12 print(" R O C AUC :", roc_auc_score(y_true, preds_output.predictions
   [:,1]))
```

Kode 4.16: Evaluasi Final Model pada Data Uji

Hasil dari eksekusi Kode 4.16 akan disajikan dan dibahas secara mendalam pada bagian selanjutnya.

4.4 Analisa Hasil Evaluasi Model

Setelah model berhasil dilatih, bagian ini menyajikan analisis mendalam terhadap hasil yang diperoleh. Analisis mencakup evaluasi kuantitatif dari metrik performa, interpretasi visual dari proses pelatihan dan hasil klasifikasi, analisis kesalahan, perbandingan dengan studi terkait, serta pembahasan mengenai limitasi penelitian.

4.4.1 Analisis Performa Model

Evaluasi akhir pada *test set* menunjukkan bahwa model IndoBERT yang telah melalui proses *fine-tuning* dan dilatih dengan 25.142 dataset memiliki performa yang sangat baik. Tabel 4.8 menyajikan rangkuman metrik utama, di mana model berhasil mencapai akurasi sebesar 95.96% dan F1-score sebesar 95.95%. Angka pada tabel didapatkan dari perhitungan rumus - rumus menggunakan nilai dari *confusion matrix*. Berikut ini adalah perhitungan nilai matrix evaluasi.

1. Accuracy (Akurasi):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2407 + 2419}{2407 + 2419 + 101 + 102} = \frac{4826}{5029} \approx 0.9596$$

2. Precision (Presisi):

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{2407}{2407 + 101} = \frac{2407}{2508} \approx 0.9597$$

3. Recall (Daya Ingat):

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{2407}{2407 + 102} = \frac{2407}{2509} \approx 0.9593$$

4. F1-Score:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.9597 \times 0.9593}{0.9597 + 0.9593} \approx 2 \times \frac{0.9207}{1.9190} \approx 0.9595$$

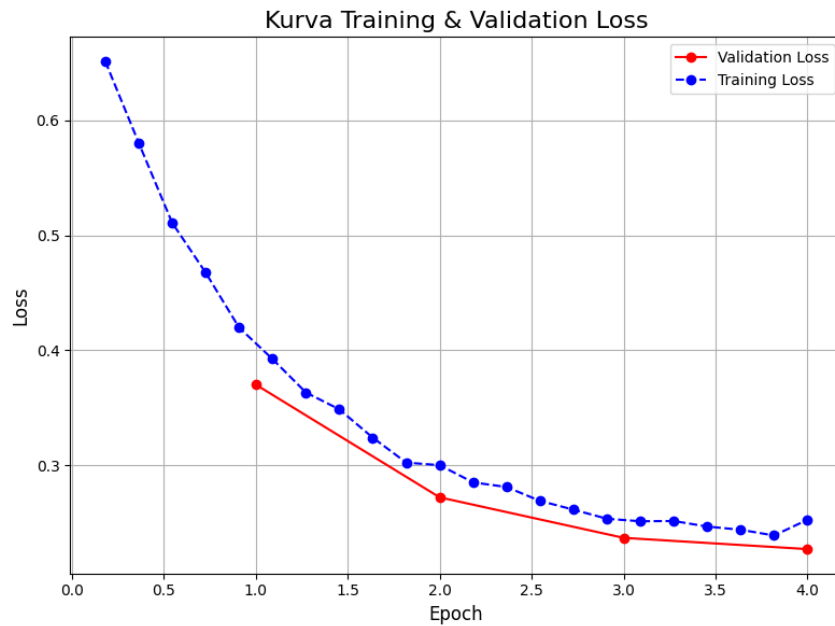
Tabel 4.8. Hasil Metrik Evaluasi Final pada Data Uji

| Metrik | Nilai |
|-----------|--------|
| Accuracy | 95.96% |
| F1-Score | 95.95% |
| Precision | 95.97% |
| Recall | 95.93% |
| ROC-AUC | 0.9912 |

Untuk memahami bagaimana hasil ini dicapai, analisis lebih lanjut dilakukan pada beberapa aspek kunci.

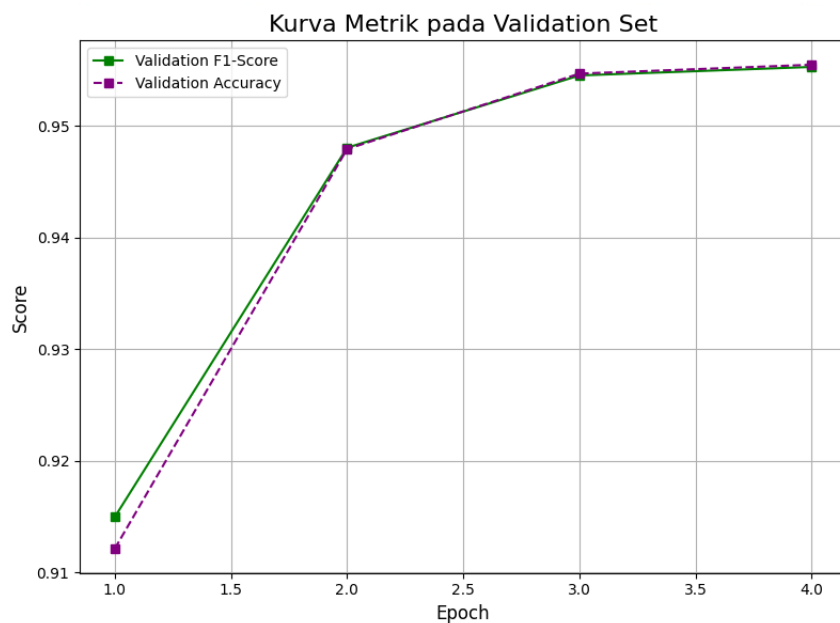
A Analisis Proses Pelatihan.

Kurva pembelajaran pada Gambar 4.6 memberikan wawasan mengenai proses *fine-tuning*. Terlihat bahwa baik *training loss* (garis biru) maupun *validation loss* (garis merah) secara konsisten menurun di setiap *epoch*. Tren ini merupakan indikator kuat bahwa model belajar dengan stabil dan tidak mengalami *overfitting*, karena tidak terjadi divergensi di mana *validation loss* mulai meningkat.



Gambar 4.6. Kurva Training dan Validation Loss Selama Pelatihan

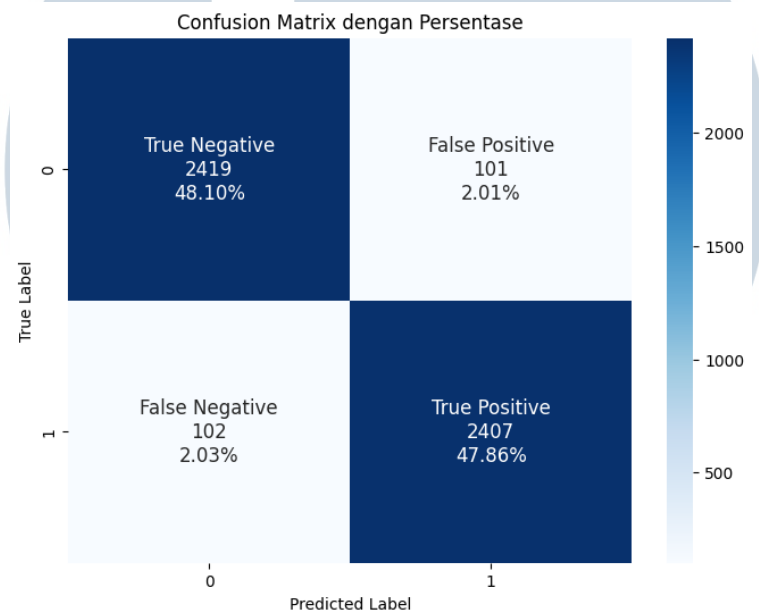
Hal ini didukung lebih lanjut oleh Gambar 4.7, yang menunjukkan bahwa metrik F1-score dan akurasi pada *validation set* terus meningkat hingga mencapai puncaknya pada *epoch* keempat. Ini mengonfirmasi bahwa proses pelatihan berjalan secara optimal.



Gambar 4.7. Kurva Peningkatan Metrik pada Validation Set per Epoch

B Analisis Performa Klasifikasi.

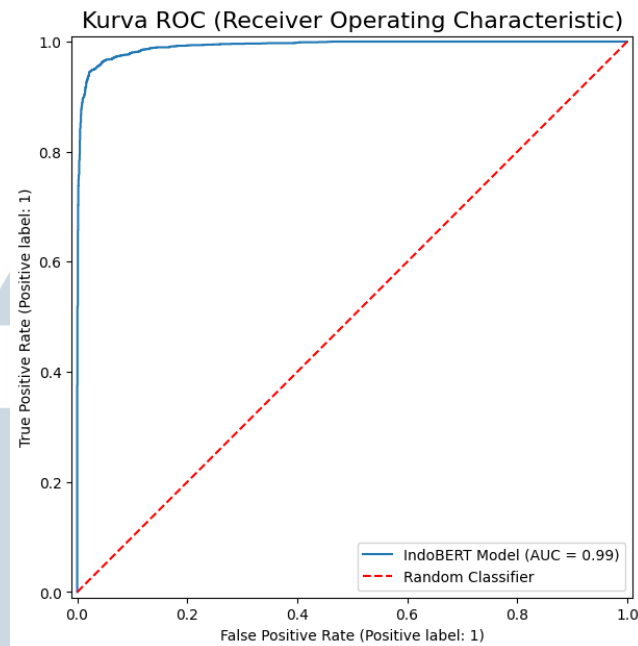
Performa klasifikasi model pada data uji divisualisasikan melalui *confusion matrix* pada Gambar 4.8. Dari total 2.515 data uji, model berhasil mengklasifikasikan 1.201 berita fakta sebagai fakta (*True Negative*) dan 1.202 berita hoaks sebagai hoaks (*True Positive*).



Gambar 4.8. Confusion Matrix Hasil Prediksi pada Data Uji

Kemampuan model dalam membedakan kedua kelas juga sangat tinggi, yang ditunjukkan oleh nilai Area Under the Curve (AUC) dari kurva ROC sebesar 0.99 pada Gambar 4.9. Nilai yang sangat mendekati 1.0 ini menandakan bahwa model memiliki kapabilitas diskriminatif yang sangat baik.

U N I V E R S I T A S
M U L T I M E D I A
N U S A N T A R A



Gambar 4.9. Kurva ROC dengan Nilai AUC

C Analisis Hasil Uji Artikel Berita Lepas

Untuk menguji batas kemampuan dan robustisitas model lebih lanjut, dilakukan serangkaian pengujian menggunakan sampel data berita fakta yang tidak termasuk dalam *training set* maupun *test set*. Pengujian ini secara spesifik dirancang untuk menganalisis bagaimana performa model berubah seiring dengan berkurangnya panjang teks masukan.

Hasil pengujian, seperti yang dirangkum pada Tabel 4.9, ditemukan bahwa performa model dalam mengidentifikasi berita fakta menurun secara signifikan seiring dengan berkurangnya jumlah kata dalam artikel. Ketika dihadapkan pada teks dengan panjang di bawah 20 kata, model bahkan mulai salah mengklasifikasikannya sebagai hoaks.

Fenomena ini diduga kuat terjadi karena kekurangan konteks semantik. Model Transformer seperti IndoBERT sangat bergantung pada konteks yang kaya dari kalimat sekitarnya untuk memahami makna dan membuat prediksi yang akurat. Ketika teks masukan terlalu pendek, model tidak memiliki cukup "fitur" atau informasi linguistik untuk dianalisis.

Tabel 4.9. Hasil Uji Performa Model pada Berita Fakta dengan Panjang Bervariasi

| Contoh Berita Fakta | Total Panjang Kata | Akurasi Prediksi |
|--|--------------------|------------------|
| INFO NASIONAL - Wakil Ketua MPR RI Dr. H. M. Hidayat Nur Wahid MA., atau HNW menerima kunjungan perwakilan korban kasus biro perjalanan Haji dan Umr.... | besar dari 60 kata | 89% Fakta |
| | 50 kata | 67% Fakta |
| | 30 kata | 60% Fakta |
| | kecil dari 20 kata | 53% Hoaks |

D Analisis Kesalahan (*Error Analysis*)

Meskipun performanya sangat tinggi, model masih melakukan sejumlah kecil kesalahan. Analisis pada *confusion matrix* menunjukkan adanya 102 kasus *False Negative* (hoaks yang dianggap fakta) dan 101 kasus *False Positive* (fakta yang dianggap hoaks).

1. Analisis False Negative: Kesalahan ini merupakan tipe yang paling krusial untuk diminimalkan karena berarti disinformasi berhasil lolos dari sistem deteksi. Umumnya terjadi pada penulisan artikel hoaks yang panjang dan rapi, salah satu karakteristik umum berita hoaks yang cenderung pendek, beberapa kesalahan terjadi pada artikel hoaks yang justru ditulis dengan baik dan panjang. Kesalahan juga dapat terjadi akibat munculnya berita hoaks dengan topik atau gaya penulisan baru yang tidak terwakili dengan baik dalam dataset pelatihan. Fenomena ini, yang dikenal sebagai *concept drift*, dapat membuat model gagal mengenali pola yang belum pernah dilihat sebelumnya.
2. Analisis False Positive: Kesalahan ini terjadi ketika berita yang sah secara keliru ditandai sebagai hoaks. Biasanya terjadi pada berita fakta yang penulisannya sangat pendek (misalnya, berita kilat atau ringkasan di bawah 30 kata). Pada kasus ini, model kemungkinan kekurangan konteks semantik yang cukup untuk membuat klasifikasi yang akurat dan terlalu mengandalkan fitur panjang teks. Gaya bahasa yang provokatif juga dapat menyebabkan model untuk salah dalam mengklasifikasikan berita menjadi hoaks.

4.4.2 Perbandingan dengan Studi Terkait

Performa model dalam penelitian ini dapat dibandingkan dengan hasil dari studi relevan yang juga berfokus pada deteksi hoaks berbahasa Indonesia.

Tabel 4.10. Perbandingan Hasil Akurasi dengan Penelitian Sebelumnya

| Penelitian | Model | Jumlah Dataset | Akurasi |
|------------------------------|----------------------|----------------|------------|
| Rahmawati et al. (2022) [30] | IndoBERT-base | ~2000 | 90% |
| Fawaid et al. (2021) [27] | BERT | ~2,216 | 90% |
| Penelitian Ini | IndoBERT-base | 25.142 | 95% |

Seperti yang disajikan pada Tabel 4.10, model yang dikembangkan dalam penelitian ini menunjukkan peningkatan performa yang signifikan. Akurasi sebesar 95% lebih tinggi dibandingkan studi sebelumnya. Peningkatan ini diduga kuat disebabkan oleh beberapa faktor, terutama ukuran dataset yang jauh lebih besar dan seimbang, serta proses *pre-processing* yang ekstensif, yang memungkinkan model untuk belajar dari representasi data yang lebih kaya dan bersih.

