

BAB 4

HASIL DAN DISKUSI

Bab Hasil dan Diskusi menyajikan implementasi algoritma *hybrid* BERT dan XGBoost dalam mengklasifikasikan tweet yang mengandung ujaran kebencian pada platform X. Hasil penelitian disampaikan secara sistematis, dimulai dari penggunaan dataset, proses *preprocessing* data, pembagian data menjadi data latih dan data uji, pelatihan model *hybrid*, dan evaluasi performa model.

4.1 Spesifikasi Sistem

Penelitian ini dilaksanakan dengan menggunakan perangkat keras dan perangkat lunak yang mendukung proses eksperimen dan pengolahan data. Spesifikasi sistem digunakan untuk memastikan bahwa proses komputasi, khususnya pada tahap pelatihan model BERT dan XGBoost yang membutuhkan daya pemrosesan tinggi, dapat berjalan secara optimal. Adapun spesifikasi sistem yang digunakan adalah sebagai berikut:

1. Hardware

- Processor: AMD Ryzen 5 5600X (6 Core, 12 Thread)
- RAM: 16 GB DDR4 3200 MHz
- GPU: NVIDIA GeForce RTX 3060Ti (8 GB GDDR6)
- SSD 1: Samsung SSD 980 500 GB M.2 NVMe
- SSD 2: Pioneer 120 GB Sata 3
- Hard Disk 1: Western Digital Caviar Blue 1 TB SATA 3

2. Software

- Sistem Operasi: Windows 11 Pro
- Anaconda 3
- Jupyter Notebook
- Opera GX

Pemilihan spesifikasi tersebut disesuaikan dengan kebutuhan pemrosesan teks dan pelatihan model *transformers* yang relatif kompleks. Penggunaan GPU

bertujuan untuk mempercepat proses pelatihan model BERT, yang secara signifikan lebih lambat jika hanya menggunakan CPU.

4.2 Deskripsi Dataset

Penelitian ini menggunakan dataset berupa kumpulan tweet berbahasa Indonesia yang telah dilabeli sebagai *hate speech* dan *non hate speech*. Setiap class memiliki value antara 0 dan 1, value 1 mengindikasikan bahwa kolom tersebut positif terindikasi dan value 0 mengindikasikan bahwa kolom negatif terindikasi. Dataset diperoleh dari hasil kurasi dan *preprocessing* data publik yang dikumpulkan dari media sosial Twitter menggunakan Twitter API [44]. Dataset ini berisi total 13.169 tweet yang dianotasi dengan pendekatan *multilabel*, memungkinkan identifikasi lebih dari satu kategori dalam satu tweet. Setiap tweet diberi label berdasarkan dua kategori utama: "HS" (*hate speech*) dan "Abusive" (bahasa kasar). Selain itu, terdapat sub-kategori spesifik untuk *hate speech*, yaitu:

Tabel 4.1. Contoh lima data pertama

No	Tweet	HS	Abusive	HSI	HSG	HSRI	HSRc	HSP	HSGn	HSO	HSW	HSM	HSS
0	- disaat semua cowok berusaha melacak perhatian...	1	1	1	0	0	0	0	0	1	1	0	0
1	RT USER: USER siapa yang telat ngasih tau elu?...	0	1	0	0	0	0	0	0	0	0	0	0
2	41. Kadang aku berfikir, kenapa aku tetap percaya...	0	0	0	0	0	0	0	0	0	0	0	0
3	USER USER AKU ITU AKU KU TAU MATAMU SIPIT...	0	0	0	0	0	0	0	0	0	0	0	0
4	USER USER Kaum cebong kapir udah keliatan dong...	1	1	0	1	1	0	0	0	0	0	1	0

- HS_Individual (ditujukan kepada individu)
- HS_Group (ditujukan kepada kelompok)
- HS_Religion (berkaitan dengan agama/kepercayaan)
- HS_Race (berkaitan dengan ras/etnis)
- HS_Physical (berkaitan dengan fisik/disabilitas)
- HS_Gender (berkaitan dengan gender/orientasi seksual)
- HS_Other (berkaitan dengan fitnah/penistaan lainnya)
- HS_Weak (*hate speech* lemah)

- HS_Moderate (*hate speech* sedang)
- HS_Strong (*hate speech* kuat)

Dataset ini memiliki representativitas yang kuat dalam mendukung pembangunan dan evaluasi sistem deteksi ujaran kebencian pada media sosial berbahasa Indonesia.

4.3 Implementasi Sistem

Bagian ini menjelaskan proses implementasi sistem klasifikasi *hate speech* yang telah dirancang, mulai dari import libraries, *preprocessing* data, hingga validasi dan pengujian model menggunakan *metrics* evaluasi. Implementasi dilakukan menggunakan Python dengan bantuan model algoritma BERT dan XGBoost. Tahapan ini bertujuan untuk merealisasikan rancangan sistem secara teknis dan menguji performanya terhadap data komentar yang telah dikumpulkan.

4.3.1 Import Libraries

Memulai proses implementasi, langkah pertama yang dilakukan adalah mengimpor berbagai pustaka (*library*) yang dibutuhkan. Pustaka-pustaka ini mencakup modul untuk pemrosesan data, pembelajaran mesin, serta visualisasi terlampir pada Kode 4.1.

```

1 import pandas as pd
2 import torch
3 import re
4 import emoji
5 import string
6 import numpy as np
7 import torch
8 import time
9 import optuna
10 import matplotlib.pyplot as plt
11 import torch.nn as nn
12 import torch.nn.functional as F
13 import xgboost as xgb
14 import json
15 import joblib
16 import seaborn as sns
17 from xgboost import XGBClassifier

```

```

18 from xgboost.callback import TrainingCallback
19 from torch.utils.data import Dataset, DataLoader
20 from transformers import AutoTokenizer, AutoConfig,
    AutoModelForSequenceClassification
21 from transformers import BertTokenizer,
    BertForSequenceClassification, AdamW, get_scheduler, BertModel
22 from sklearn.preprocessing import StandardScaler
23 from sklearn.metrics import classification_report, accuracy_score,
    f1_score, precision_score, recall_score, roc_auc_score,
    confusion_matrix, ConfusionMatrixDisplay, log_loss
24 from sklearn.model_selection import train_test_split, GridSearchCV
25 from collections import Counter
26 from tqdm.auto import tqdm

```

Kode 4.1: Import libraries

Pustaka-pustaka tersebut menyediakan fungsi-fungsi penting yang akan digunakan dalam seluruh tahapan penelitian, mulai dari pembersihan data, pembentukan dataset, pelatihan model, evaluasi performa, hingga visualisasi hasil. Dengan melakukan impor diawal, seluruh modul yang diperlukan telah tersedia dan dapat dipanggil sesuai kebutuhan pada bagian-bagian implementasi berikutnya.

4.3.2 Preprocessing Data

Proses *preprocessing* merupakan tahap krusial dalam pemodelan teks karena secara langsung memengaruhi kualitas data yang digunakan untuk pelatihan model. Pada tahap ini, data mentah dibersihkan dari elemen-elemen yang tidak relevan. Selain itu, dilakukan juga standarisasi teks untuk memastikan konsistensi format, yang penting dalam pemrosesan bahasa alami.

A Menampilkan Dataset

Dalam proses analisis data, langkah awal yang dilakukan adalah pemeriksaan awal (*initial inspection*) untuk memahami struktur serta karakteristik data secara menyeluruh.

```

1 # Show top of dataset
2 df = pd.read_csv('./multilabel-dataset/re_dataset.csv', encoding='
    latin1')
3 df.head()

```

Kode 4.2: Pemeriksaan dataset

Tahapan ini bertujuan untuk memuat dataset awal dan menampilkan struktur umum dari dataset, meliputi jumlah entri, jumlah kolom, serta distribusi awal label yang menjadi fokus analisis. Selanjutnya, dilakukan proses penghapusan kolom-kolom yang tidak relevan terhadap tugas klasifikasi *hate speech*, guna menyederhanakan struktur data dan meningkatkan efisiensi analisis. Potongan Kode 4.3 merupakan fungsi untuk melakukan menghilangkan kolom yang tidak digunakan.

```
1 # Remove columns yang tidak diperlukan
2 df.drop(columns = ['Abusive',
3                    'HS_Individual',
4                    'HS_Group',
5                    'HS_Religion',
6                    'HS_Race',
7                    'HS_Physical',
8                    'HS_Gender',
9                    'HS_Other',
10                   'HS_Weak',
11                   'HS_Moderate',
12                   'HS_Strong'], inplace=True)
```

Kode 4.3: Potongan kode *remove* kolom

Tabel 4.2. Hasil *remove* kolom

	Tweet	HS
0	- disaat semua cowok berusaha melacak perhatian...	1
1	RT USER: USER siapa yang telat ngasih tau elu?...	0
2	41. Kadang aku berfikir, kenapa aku tetap percaya...	0
3	USER USER AKU ITU AKU KU TAU MATAMU SIPIT...	0
4	USER USER Kaum cebong kapir udah keliatan dong...	1

Pada Tabel 4.2 disajikan perubahan dataset setelah kolom yang tidak dibutuhkan dihapus, tahap berikutnya adalah melakukan pemeriksaan terhadap nilai-nilai kosong (*null values*). Langkah ini penting karena keberadaan nilai kosong dapat menyebabkan *error* pada saat pelatihan model. Selain itu, duplikasi data juga dihapus untuk memastikan bahwa setiap entri bersifat unik dan tidak memberikan bias terhadap hasil model. Potongan Kode 4.4 merupakan kode untuk *check null* dan menghilangkan duplikasi pada kolom Tweet.

```

1 # Check is null value
2 df.isnull().sum()
3
4 # Remove duplicates
5 df = df.drop_duplicates(subset=['Tweet'])

```

Kode 4.4: Cek *null* dan hapus duplikasi

B Cleaning Text

Tahap selanjutnya, dilakukan proses pembersihan teks (*text cleaning*) pada kolom Tweet. Proses ini mencakup serangkaian transformasi untuk meningkatkan kualitas data teks sebelum digunakan dalam pelatihan model. Beberapa langkah yang diterapkan meliputi: konversi seluruh karakter ke huruf kecil, penghapusan tautan (URL), tagar (*hashtag*), karakter *unicode*, simbol, kata spesifik "USER", karakter non-ASCII serta spasi berlebih (*whitespace*) yang dihilangkan untuk mengurangi *noise* pada teks.

Tahap pembersihan ini juga mencakup normalisasi kata slang atau alay menggunakan kamus eksternal, sehingga kata-kata tidak baku dapat dikonversi ke bentuk formal yang lebih representatif. Potongan kode yang menunjukkan implementasi berbagai fungsi dalam proses *preprocess_text* ditampilkan pada Kode 4.5.

```

1 # to lower
2 def to_lower(text):
3     return text.lower()
4
5 # remove URLs
6 def remove_URL(text):
7     text = re.sub(r'https?:\/\/\S+|www\.\S+', '', text)
8     return text
9
10 # remove mention and hashtag
11 def remove_hashtag(text):
12     text = re.sub(r'@\w+', '', text)
13     text = re.sub(r'#\w+', '', text)
14     return text.strip()
15
16 # remove words like "\x9f\x80\xA0"
17 def normalize_unicode(text):
18     return re.sub(r'(\x[da-fA-F]{2})+', '', text)

```

```

19
20 # remove symbols
21 def remove_punctuation(text):
22     text = re.sub(f"[{re.escape(string.punctuation)}]", " ", text)
23     # Hilangkan tanda baca standar
24     text = re.sub(r"[^\w\s]", " ", text) # Hilangkan simbol non
    -alfanumerik selain spasi
25     text = re.sub(r"\s+", " ", text).strip() # Hapus spasi
    berlebih
26     return text
27
28 # remove repetitive character
29 def remove_repetitive_char(text):
30     return re.sub(r'(\1{2,})', r'\1', text)
31
32 # remove repetitive words
33 def remove_repetitive_words(text):
34     text = re.sub(r'\b(\w+)(\1)+\b', r'\1', text)
35     return text
36
37 # remove whitespace (spasi)
38 def remove_whitespace(text):
39     # Perbaiki escape sequence literal
40     text = re.sub(r"/t", "\t", text) # Ganti "/t" dengan tab asli
41
42     # Gabungkan whitespace yang berlebihan secara selektif
43     text = re.sub(r" +", " ", text) # Hapus spasi berlebih
44     text = re.sub(r"\n+", "\n", text) # Hapus newline berlebih
45     text = re.sub(r"\r+", "\r", text) # Hapus carriage return
    berlebih
46     text = re.sub(r"\t+", "\t", text) # Hapus tab berlebih
47     return text.strip(" ")
48
49 # normalize alay words using dictionary
50 def normalize_alay(text, alay_dict=alay_dict):
51     return ' '.join([alay_dict.get(word, word) for word in text.
    split()])
52
53 # remove user Words
54 def remove_user_words(text):
55     return re.sub(r'\buser\b', '', text)
56
57 # remove retweet

```

```

57 def remove_retweet(text):
58     return re.sub(r'^\s*RT\s*[:\-\ ]?\s*', '', text)
59
60 # remove non-ascii
61 def remove_non_ascii(text):
62     return re.sub(r'^\x00-\x7f]', r'', text)
63
64 # remove emojis dan emojis classic
65 def clean_emojis_and_noise(text):
66     text = ''.join(char for char in text if char not in emoji.
        EMOJI_DATA) # Hapus emoji dengan libraries
67     text = re.sub(r'[:;=8][\-\o*\']?[\]\]\(dDpPxX/\|]\]', '', text)
        # Hapus emotikon klasik seperti :) :( :-D
68     text = re.sub(r'[ ]', '', text) # Hapus tanda
        kutip atau kurung aneh
69     return text

```

Kode 4.5: Fungsi-fungsi *cleaning text*

Setelah mendefinisikan berbagai fungsi untuk pembersihan teks, seluruh fungsi tersebut dikemas ke dalam satu fungsi utama bernama *preprocess_text* berguna mempermudah proses penerapan secara menyeluruh. Fungsi *preprocess_text* terlampir pada Kode 4.6, kode ini kemudian digunakan untuk membersihkan data tweet, dan hasil pembersihannya disimpan dalam kolom baru bernama *clean_text*. Selanjutnya, data yang telah melalui tahap praproses ini disimpan ke dalam file CSV baru untuk digunakan pada tahap analisis berikutnya.

```

1 # Preprocess function
2 def preprocess_text(text, alay_dict):
3     text = to_lower(text)
4     text = remove_URL(text)
5     text = remove_retweet(text)
6     text = remove_hastag(text)
7     text = normalize_unicode(text)
8     text = remove_non_ascii(text)
9     text = remove_repetitive_char(text)
10    text = remove_whitespace(text)
11    text = remove_punctuation(text)
12    text = remove_user_words(text)
13    text = normalize_alay(text, alay_dict)
14    return text
15
16 # Cleaning Tweet dan save in clean_text column

```



```

17 df['clean_text'] = df['Tweet'].apply(lambda x: preprocess_text(x,
    alay_dict))
18
19 # Save to csv
20 df.to_csv("cleaned_dataset.csv", index=False)
21 print("Cleaning selesai! Dataset disimpan sebagai cleaned_dataset.
    csv")

```

Kode 4.6: Fungsi *preprocessing*

Tabel 4.3. Data sebelum dan sesudah *cleaning*

No	Tweet	HS	clean_text
0	- disaat semua cowok berusaha melacak perhatian...	1	di saat semua cowok berusaha melacak perhatian...
1	RT USER: USER siapa yang telat ngasih tau elu?...	0	rt siapa yang telat memberi tau kamu edan sara...
2	41. Kadang aku berfikir, kenapa aku tetap percaya...	0	41 kadang aku berpikir kenapa aku tetap percaya...
3	USER USER AKU ITU AKU KU TAU MATAMU SIPIT...	0	aku itu aku dan ku tau matamu sipit tapi dilihat...
4	USER USER Kaum cebong kapir udah keliatan dong...	1	kaum cebong kafir sudah kelihatan dongoknya dan...

Pada Tabel 4.3 menggambarkan hasil transformasi teks sebelum dan sesudah dilakukan proses pembersihan. Perbandingan ini menunjukkan bahwa fungsi praproses yang diterapkan berhasil mengurangi berbagai elemen dan *noise* tidak relevan yang dapat mengganggu kinerja model. Pembersihan ini bertujuan untuk menyederhanakan struktur teks sekaligus mempertahankan konteks semantik utama, sehingga model pembelajaran mesin dapat lebih efektif dalam mengenali pola-pola ujaran kebencian.

C Deteksi dan Koreksi Mislabeled

Setelah proses pembersihan teks dari berbagai *noise* yang dapat mengganggu kinerja model deteksi, tahap selanjutnya adalah melakukan deteksi ulang dan koreksi terhadap label dengan memanfaatkan list kata kasar (*abusive*) yang terbesar dalam menentukan kategori *hate speech*. Proses pembaruan label ini

dilakukan dengan memuat kamus eksternal serta file CSV yang telah melalui tahap *preprocessing*. Selanjutnya, dilakukan proses filterisasi untuk mengidentifikasi apakah dalam teks terdapat indikasi ujaran kebencian sesuai dengan entri dalam kamus tersebut. Potongan Kode 4.7 menunjukkan implementasi proses filterisasi untuk mengidentifikasi label yang berpotensi termasuk dalam kategori *hate speech*.

```

1 # Load clean dataset & show
2 df_2 = pd.read_csv('cleaned_dataset.csv')
3 df_2 = df_2.drop(columns=['Tweet'])
4 df_2.tail()
5
6 # Check is null value
7 df_2 = df_2.dropna(subset=['clean_text'])
8 df_2.isnull().sum()
9
10 # Tambahkan Abusive -> Hate-Speech
11 # Load kamus kata kasar dari eksternal dictionary
12 with open('./kata_kasar.txt', 'r', encoding='utf-8') as f:
13     daftar_kasar = [baris.strip().lower() for baris in f if baris.
14                     strip()]
15
16 # Cek distribusi label 0 yang mengandung kata kasar
17 def mengandung_kata_kasar(teks, daftar_kasar):
18     return any(kata in teks.lower().split() for kata in
19               daftar_kasar)
20
21 # Filter data
22 potensi_mislabeled = df_2[(df_2['HS'] == 0) & (df_2['clean_text'].
23         apply(lambda x: mengandung_kata_kasar(str(x), daftar_kasar)))]
24
25 # Simpan untuk audit manual
26 potensi_mislabeled.to_csv('potensi_mislabeled.csv', index=False)
27 print(f"Jumlah potensi mislabeling ditemukan: {len(
28       potensi_mislabeled)}")

```

Kode 4.7: Deteksi *mislabeled*

Tabel 4.4. Ringkasan jumlah data dan hasil audit label

Tahap	Jumlah Data
Total data awal	13.014
Potensi mislabeling ditemukan	1.030

Setelah mengidentifikasi sejumlah data yang berpotensi mengalami *mislabeling*, terdapat 1030 data yang mengandung ujaran kebencian. Sejumlah data yang terindikasi sebagai *hate speech* disimpan ke dalam file CSV dengan 50 sample untuk dilakukan proses audit lebih lanjut. Setelah hasil audit manual menunjukkan bahwa alasan pelabelan ulang valid dikarenakan terindikasi ujaran kata kasar di dalamnya dan hal tersebut sesuai dengan kriteria *hate speech*, maka semua data tersebut diperbarui dan dimasukkan ke dalam kategori *hate speech* pada kolom label yang bersangkutan. Potongan Kode 4.8 menyajikan penyimpanan potensial hasil deteksi ke dalam CSV sebanyak 50 sample, lalu mengupdate kolom dari kategori 'HS' 0 menjadi 1.

```

1 # Sample untuk audit data potensial mislabel
2 sample = potensi_mislabel.sample(50, random_state=42)
3 sample.to_csv('audit_sample.csv', index=False)
4
5 # Update label HS 0 menjadi 1 untuk data yang mengandung kata
  kasar
6 df_2.loc[potensi_mislabel.index, 'HS'] = 1
7
8 # Simpan DataFrame yang sudah diperbarui jika perlu
9 df_2.to_csv('df_2_terupdate.csv', index=False)
10 print("Label telah diperbarui untuk data yang mengandung kata
    kasar.")

```

Kode 4.8: *Update label*

Tabel 4.5. Distribusi label *hate speech* sebelum dan sesudah *relabeling*

Label HS	Sebelum Relabeling	Sesudah Relabeling
0 (Bukan Hate Speech)	7.516	6.486
1 (Hate Speech)	5.498	6.528

Berdasarkan hasil *relabeling* yang ditampilkan pada Tabel 4.6, dapat diamati adanya pergeseran distribusi label yang signifikan, khususnya pada peningkatan jumlah data yang dikategorikan sebagai *Hate Speech*. Proses ini bertujuan untuk mengkategorikan tweet *abusive* atau mengandung kata kekerasan menjadi kategori ujaran kebencian (*hate speech*).

4.3.3 Perancangan Model

Pada tahap ini, dilakukan proses integrasi antara representasi fitur dari model BERT dengan algoritma klasifikasi XGBoost. Model BERT digunakan untuk menghasilkan representasi vektor dari setiap teks melalui proses vektor *embedding*, yang kemudian diekstraksi dari lapisan akhir BERT. Vektor-vektor ini selanjutnya digunakan sebagai input bagi model XGBoost untuk melakukan klasifikasi terhadap label *hate speech*.

A Split Dataset

Dalam memastikan bahwa model memperoleh distribusi data yang seimbang selama pelatihan dan evaluasi, dilakukan proses pembagian dataset menggunakan teknik *stratified split*. Teknik ini menjaga proporsi label dalam subset data tetap konsisten dengan distribusi asli dataset. Dataset dibagi menjadi tiga bagian, yaitu data pelatihan (70%), data validasi (15%), dan data uji (15%). Pembagian ini dilakukan dalam dua tahap, di mana tahap pertama memisahkan data pelatihan dari data sementara (30%), dan tahap kedua membagi data sementara secara merata menjadi data validasi dan data pengujian. Potongan Kode 4.9 menunjukkan implementasi proses pembagian data tersebut.

```
1 # Stratified split (tetap menjaga distribusi label dengan seimbang
  )
2 # Split 70% train dan 30% temp validation
3 train_idx, temp_idx = train_test_split(
4     list(range(len(df_2))), test_size=0.3, random_state=42,
5     stratify=df_2['HS']
6 )
7 # Ambil label dari temp untuk stratified split kedua
8 temp_labels = [df_2['HS'].iloc[i] for i in temp_idx]
9
10 # Split temp menjadi validation dan test (masing-masing 15%)
11 val_idx, test_idx = train_test_split(
12     temp_idx, test_size=0.5, random_state=42, stratify=temp_labels
13 ) # 30% . 0.5 = 15%
14
15 # Check jumlah data
16 print("Jumlah data:")
17 print("Train      :", len(train_idx))
18 print("Validation:", len(val_idx))
```

```
19 print("Test      :", len(test_idx))
```

Kode 4.9: Split dataset

Tabel 4.6. Distribusi dataset

Data	Jumlah Data
Train	9.116
Validation	1.953
Test	1.954

Pada Kode 4.10 ditampilkan proses menampilkan distribusi dataset hasil split. Setelah dilakukan proses pembagian data menggunakan metode *stratified split*, diperoleh distribusi data sebesar 9.116 data untuk pelatihan (*train*), 1.953 data untuk validasi, dan 1.954 data untuk pengujian (*test*). Metode ini digunakan untuk memastikan proporsi label yang seimbang pada setiap subset data, sehingga mengurangi potensi bias selama proses pelatihan dan evaluasi model. Langkah selanjutnya adalah melakukan verifikasi terhadap hasil pembagian tersebut. Verifikasi ini mencakup pemeriksaan distribusi label pada masing-masing subset, yaitu data pelatihan, validasi, dan pengujian, guna memastikan bahwa distribusi kelas tetap seimbang. Selain itu, dilakukan analisis terhadap rata-rata panjang teks pada ketiga subset untuk mengetahui sejauh mana kompleksitas input teks tersebar secara merata di seluruh bagian data.

```
1 # Count distribusi label train / val / test
2 print("Train:", Counter(train_df['HS']))
3 print("Validation:", Counter(val_df['HS']))
4 print("Test:", Counter(test_df['HS']))
5
6 # check avg panjang text train / val / test
7 train_df['length'] = train_df['clean_text'].apply(lambda x: len(x.
    split()))
8 val_df['length'] = val_df['clean_text'].apply(lambda x: len(x.
    split()))
9 test_df['length'] = test_df['clean_text'].apply(lambda x: len(x.
    split()))
10
11 print("Train avg len:", train_df['length'].mean())
12 print("Val avg len:", val_df['length'].mean())
13 print("Test avg len:", test_df['length'].mean())
```

Kode 4.10: Menampilkan distribusi dataset setelah split

Hasil dari proses pembagian data ini menunjukkan bahwa distribusi label pada masing-masing subset tetap terjaga secara proporsional. Pemeriksaan terhadap jumlah data dan distribusi kelas pada data pelatihan, validasi, dan pengujian memastikan bahwa tidak terdapat ketimpangan yang signifikan yang dapat memengaruhi performa model. Selain itu, rata-rata panjang teks yang relatif serupa pada ketiga subset menunjukkan bahwa kompleksitas input teks tersebar secara merata.

B Feature Extraction

Pada tahap ini, proses representasi teks dilakukan dengan memanfaatkan model bahasa berbasis *transformer*, yaitu IndoBERT-Base, yang diimplementasikan melalui pustaka *Hugging Face Transformers*. Tujuan utama dari tahap ini adalah mengubah data teks yang telah dibersihkan menjadi representasi numerik berdimensi tetap, yang dapat digunakan sebagai fitur input untuk algoritma klasifikasi pada tahap selanjutnya. Model IndoBERT-Base dipilih karena dilatih secara khusus pada korpus berbahasa Indonesia dan memiliki arsitektur BERT-Base dengan dimensi vektor sebesar 768, sehingga mampu menangkap informasi semantik secara lebih mendalam dibandingkan dengan model yang berukuran lebih kecil.

Langkah pertama dalam proses ini adalah memuat *tokenizer* dan model IndoBERT-Base dari repositori `indobenchmark/indobert-base-pl`. Selanjutnya, setiap teks yang terdapat dalam dataset dikonversi menjadi token menggunakan tokenizer tersebut dan dipetakan ke dalam format tensor yang kompatibel dengan model BERT. Proses tokenisasi ini dilakukan dengan menetapkan panjang maksimum 128 token dan menerapkan *padding* serta *truncation* secara otomatis untuk menjaga konsistensi input.

Setelah teks dikonversi, representasi vektor diperoleh dengan mengambil keluaran dari lapisan terakhir model BERT, yaitu `last_hidden_state`, yang memiliki bentuk tensor berdimensi `(sequence_length, hidden_size)`. Untuk mereduksi dimensi menjadi satu vektor tetap per teks, digunakan teknik *mean pooling* hanya terhadap token yang valid (tidak dipadding), dengan memperhatikan `attention_mask`. Hasil akhir dari proses ini adalah sebuah vektor berdimensi 768 yang merepresentasikan informasi semantik dari masing-masing teks secara keseluruhan.

```
1 # Load IndoBERT
```

```

2 tokenizer = BertTokenizer.from_pretrained('indobenchmark/indobert-
  base-pl')
3 bert_model = BertModel.from_pretrained('indobenchmark/indobert-
  base-pl')
4 bert_model.eval()
5 bert_model.to(device)
6 hidden_size = bert_model.config.hidden_size
7
8 # Fungsi untuk ekstrak embedding dengan mean pooling
9 def get_bert_embedding(text):
10     inputs = tokenizer(
11         text,
12         return_tensors="pt",
13         truncation=True,
14         padding='max_length',
15         max_length=128
16     )
17     inputs = {k: v.to(device) for k, v in inputs.items()}
18
19     with torch.no_grad():
20         outputs = bert_model(**inputs)
21
22     # Ambil seluruh token embeddings dan attention mask
23     last_hidden_state = outputs.last_hidden_state.squeeze(0)
24     # shape: (seq_len, hidden_size)
25     attention_mask = inputs['attention_mask'].squeeze(0)
26     # shape: (seq_len)
27
28     # Masked mean pooling (hanya token yang bukan padding)
29     masked_embeddings = last_hidden_state * attention_mask.
30     unsqueeze(-1) # (seq_len, hidden_size)
31     sum_embeddings = masked_embeddings.sum(dim=0)
32     count_tokens = attention_mask.sum()
33     mean_embedding = sum_embeddings / count_tokens
34
35     return mean_embedding.cpu().numpy()

```

Kode 4.11: Fungsi ekstraksi *embedding* dengan *mean pooling*

Proses ekstraksi vektor *embeddings* ini kemudian diterapkan terhadap seluruh data pada set pelatihan, validasi, dan pengujian. Setiap baris teks akan dikonversi menjadi satu vektor representasi menggunakan fungsi `get_bert_embedding`. Untuk menjamin stabilitas proses dan menghindari

kegagalan saat pemrosesan teks yang tidak valid atau rusak, fungsi ini dilengkapi dengan penanganan `exception` yang akan mengembalikan vektor nol apabila proses ekstraksi gagal. Embedding yang telah diperoleh dikembalikan dalam bentuk matriks berdimensi (`jumlah_data, 768`).

```

1 # Generate embeddings dari DataFrame
2 def generate_embeddings(dataframe):
3     embeddings = []
4     for text in tqdm(dataframe['clean_text'], desc="Generating
5         embeddings"):
6         try:
7             emb = get_bert_embedding(text)
8         except Exception as e:
9             print(f"Error: {e}")
10            emb = np.zeros(hidden_size) # fallback sesuai model
11            embeddings.append(emb)
12        return np.vstack(embeddings)
13
14 # Hasilkan embedding untuk train, val, test
15 X_train = generate_embeddings(train_df)
16 X_val = generate_embeddings(val_df)
17 X_test = generate_embeddings(test_df)
18
19 # Label target
20 y_train = train_df['HS'].astype(int).values
21 y_val = val_df['HS'].astype(int).values
22 y_test = test_df['HS'].astype(int).values

```

Kode 4.12: Proses *embedding* untuk seluruh dataset

Hasil akhir dari tahap ini adalah matriks *embedding* yang berisi representasi semantik dari setiap teks dalam dataset. Matriks ini digunakan sebagai input fitur untuk model klasifikasi XGBoost pada tahap berikutnya dalam *pipeline* penelitian. Sementara itu, label target (y) telah dipersiapkan dalam bentuk vektor numerik untuk setiap subset data (*train*, *validation*, *test*). Diharapkan bahwa penggunaan representasi vektor dari IndoBERT-base ini mampu menangkap konteks dan makna teks secara lebih akurat, sehingga dapat meningkatkan performa model dalam mengklasifikasikan apakah suatu komentar mengandung ujaran kebencian (*hate speech*) atau tidak.

C Hyperparameter Tuning

Pada tahap hyperparameter tuning, model XGBoost dilatih secara otomatis menggunakan kombinasi parameter yang telah ditentukan sebelumnya melalui pendekatan *Grid Search*. Proses ini bertujuan untuk mengeksplorasi ruang parameter secara sistematis guna memperoleh pijakan awal pemilihan konfigurasi model terbaik berdasarkan metrik evaluasi *F1-score*. Penggunaan metode Grid Search dipilih karena pendekatan ini menjamin pencarian yang menyeluruh terhadap semua kombinasi parameter dalam ruang pencarian yang telah ditentukan, sehingga memberikan kontrol penuh terhadap eksplorasi awal serta cocok digunakan saat ruang parameter relatif terbatas dan dapat ditentukan secara eksplisit. Selain itu, *hypertuning* membantu meminimalkan potensi bias yang muncul dari pemilihan awal parameter secara manual. Proses inisialisasi dan pelatihan model ditunjukkan pada Kode 4.13.

```
1 # Hitung perbandingan kelas untuk imbalanced data (tidak dipakai (
    data balanced))
2 # weight = len(y_train[y_train == 0]) / len(y_train[y_train == 1])
3
4 # Grid parameter
5 param_grid = {
6     'max_depth': [4, 6, 8],
7     'learning_rate': [0.01, 0.1, 0.2],
8     'n_estimators': [100, 200],
9     'subsample': [0.8, 1.0],
10    'colsample_bytree': [0.8, 1.0]
11 }
12
13 xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss
    ')
14
15 grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid,
16                             scoring='f1', cv=3, verbose=1, n_jobs
    =-1)
17 grid_search.fit(X_train, y_train)
18
19 print("Best Parameters:", grid_search.best_params_)
20 # Best Parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1,
    'max_depth': 8, 'n_estimators': 200, 'subsample': 0.8}
```

Kode 4.13: *Hyperparameter* dengan grid search

Berdasarkan hasil pada potongan Kode 4.13 dengan menggunakan *GridSearchCV*, diperoleh kombinasi parameter terbaik sebagai berikut: *colsample_bytree*=1.0, *learning_rate*=0.1, *max_depth*=8, *n_estimators*=200, dan *subsample*=0.8. Kombinasi parameter ini kemudian digunakan untuk melatih ulang model XGBoost dengan harapan dapat meningkatkan kemampuan model dalam mendeteksi komentar bermuatan *hate speech* secara lebih akurat dan seimbang.

D Train Final Model

Setelah memperoleh representasi vektor dari setiap teks melalui *embedding* IndoBERT-base, tahap selanjutnya adalah pelatihan model klasifikasi menggunakan algoritma XGBoost. Potongan Kode 4.14 menjelaskan langkah awal pelatihan model XGBoost dimulai dengan mengonversi data fitur dan label ke dalam format *DMatrix*, yaitu format data internal milik pustaka XGBoost. Format ini dirancang khusus untuk mendukung komputasi paralel serta efisien dalam penggunaan memori. Kemudian, model dikonfigurasi dengan sejumlah parameter yang telah disesuaikan untuk mencapai keseimbangan antara kompleksitas model dan generalisasi terhadap data yang tidak terlihat. Selanjutnya, model dikonfigurasi dengan parameter-parameter penting seperti *max_depth*, *learning_rate*, *subsample*, dan *colsample_bytree* untuk mengendalikan kompleksitas dan mencegah *overfitting*, serta *reg_alpha* dan *reg_lambda* sebagai bentuk regularisasi tambahan guna menjaga kestabilan pelatihan dan kemampuan generalisasi model terhadap data baru.

```
1 # Load DMatrix
2 dtrain = xgb.DMatrix(X_train, label=y_train)
3 dval = xgb.DMatrix(X_val, label=y_val)
4 dtest = xgb.DMatrix(X_test, label=y_test)
5
6 # Parameter model
7 params = {
8     'objective': 'binary:logistic',
9     'max_depth': 4,
10    'learning_rate': 0.1,
11    'subsample': 0.8,
12    'colsample_bytree': 0.8,
13    'min_child_weight': 5,
14    'gamma': 0.5,
15    'reg_alpha': 0.7,
16    'reg_lambda': 2.0,
17    'eval_metric': 'logloss'
```

```
18 }
```

Kode 4.14: Inisialisasi *dmatrix* dan *best parameters*

Potongan Kode 4.15 dibuat guna memantau performa model secara lebih komprehensif selama proses pelatihan, digunakan fungsi callback khusus yang mencatat metrik evaluasi tambahan pada data validasi, yaitu akurasi, presisi, *recall*, dan *F1-score*. Metrik-metrik ini dipilih karena mampu memberikan gambaran yang lebih utuh mengenai efektivitas model, khususnya dalam konteks klasifikasi biner yang tidak selalu seimbang.

```
1 # Hasil evaluasi logloss dan metrik tambahan
2 evals_result = {}
3 history = {
4     'val_loss': [],
5     'val_acc': [],
6     'val_precision': [],
7     'val_recall': [],
8     'val_f1': [],
9
10    'train_loss': [],
11    'train_acc': [],
12    'train_precision': [],
13    'train_recall': [],
14    'train_f1': []
15 }
16
17 # Callback custom untuk logging metrik
18 class CustomMetricLogger(TrainingCallback):
19     def __init__(self, dtrain, y_train, dval, y_val, evals_result,
20                 history):
21         self.dtrain = dtrain
22         self.y_train = y_train
23         self.dval = dval
24         self.y_val = y_val
25         self.evals_result = evals_result
26         self.history = history
27
28     def after_iteration(self, model, epoch, evals_log):
29         # Prediksi untuk train
30         y_train_proba = model.predict(self.dtrain, iteration_range
31                                     =(0, epoch + 1))
32         y_train_pred = (y_train_proba > 0.5).astype(int)
```

```

32         train_acc = accuracy_score(self.y_train, y_train_pred)
33         train_prec = precision_score(self.y_train, y_train_pred,
zero_division=0)
34         train_rec = recall_score(self.y_train, y_train_pred,
zero_division=0)
35         train_f1 = f1_score(self.y_train, y_train_pred,
zero_division=0)
36
37         # Prediksi untuk val
38         y_val_proba = model.predict(self.dval, iteration_range=(0,
epoch + 1))
39         y_val_pred = (y_val_proba > 0.5).astype(int)
40
41         val_acc = accuracy_score(self.y_val, y_val_pred)
42         val_prec = precision_score(self.y_val, y_val_pred,
zero_division=0)
43         val_rec = recall_score(self.y_val, y_val_pred,
zero_division=0)
44         val_f1 = f1_score(self.y_val, y_val_pred, zero_division=0)
45
46         # Simpan log loss dari XGBoost
47         self.history['train_loss'].append(evals_log['train']['logloss'][epoch])
48         self.history['val_loss'].append(evals_log['val']['logloss'][epoch])
49
50         # Simpan metrik train
51         self.history.setdefault('train_acc', []).append(train_acc)
52         self.history.setdefault('train_precision', []).append(
train_prec)
53         self.history.setdefault('train_recall', []).append(
train_rec)
54         self.history.setdefault('train_f1', []).append(train_f1)
55
56         # Simpan metrik val
57         self.history['val_acc'].append(val_acc)
58         self.history['val_precision'].append(val_prec)
59         self.history['val_recall'].append(val_rec)
60         self.history['val_f1'].append(val_f1)
61
62         return False

```

Kode 4.15: *Callback logging metrics evaluasi*

Potongan Kode 4.16 merupakan proses pelatihan yang dilakukan dengan jumlah maksimum iterasi sebesar 2000 boosting rounds. Untuk mencegah pelatihan berlebih (*overfitting*), digunakan teknik *early stopping* dengan batas 10 iterasi. Artinya, pelatihan akan dihentikan secara otomatis jika tidak terdapat peningkatan performa pada data validasi selama 10 iterasi berturut-turut.

```
1 #Training
2 model = xgb.train(
3     params=params,
4     dtrain=dtrain,
5     num_boost_round=2000,
6     evals=[(dtrain, 'train'), (dval, 'val')],
7     early_stopping_rounds=10,
8     evals_result=evals_result,
9     callbacks=[CustomMetricLogger(dval, y_val, evals_result, history)
10         ],
11     verbose_eval=False
12 )
```

Kode 4.16: Pelatihan model xgboost

Tahap pelatihan model XGBoost merupakan komponen krusial dalam keseluruhan pipeline klasifikasi, representasi semantik hasil *embedding* dari IndoBERT-BASE dimanfaatkan secara maksimal untuk membedakan komentar yang mengandung ujaran kebencian dan non ujaran kebencian. Model ini dilatih dengan parameter-parameter yang telah disesuaikan secara optimal guna menyeimbangkan akurasi dan kemampuan generalisasi. Selama proses pelatihan iteratif, evaluasi dilakukan secara berkala dengan mencatat metrik performa seperti *log loss*, *accuracy*, *precision*, *recall*, dan *F1-score* ke dalam struktur *log evals_result* dan *history*, yang selanjutnya dapat dianalisis untuk memantau dinamika pembelajaran model.

E Metrics Pengujian Model

Setelah proses pelatihan selesai, tahap selanjutnya adalah mengevaluasi performa model terhadap data uji yang belum pernah dilihat oleh model sebelumnya. Evaluasi ini bertujuan untuk mengukur kemampuan generalisasi model dalam mengklasifikasikan komentar yang mengandung ujaran kebencian dan yang tidak. Selain menghitung metrik performa utama, dilakukan juga visualisasi menggunakan *confusion matrix* untuk memperoleh gambaran lebih jelas terkait

distribusi prediksi model.

```
1 # === TEST METRICS ===
2 y_test_proba = model.predict(dtest)
3 y_test_preds = (y_test_proba > 0.5).astype(int)
4
5 test_acc = accuracy_score(y_test, y_test_pred)
6 test_prec = precision_score(y_test, y_test_pred, zero_division=0)
7 test_rec = recall_score(y_test, y_test_pred, zero_division=0)
8 test_f1 = f1_score(y_test, y_test_pred, zero_division=0)
9
10 # Tambahkan ke history
11 history['test_acc'] = test_acc
12 history['test_precision'] = test_prec
13 history['test_recall'] = test_rec
14 history['test_f1'] = test_f1
15
16 # Print hasil
17 print("\n=== Evaluation on Test Set ===")
18 print(f"Accuracy : {test_acc:.4f}")
19 print(f"Precision: {test_prec:.4f}")
20 print(f"Recall    : {test_rec:.4f}")
21 print(f"F1 Score  : {test_f1:.4f}")
22
23 # Confusion Matrix - Test Set
24 cm = confusion_matrix(y_test, test_preds)
25 disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels
    =['Non-Hate', 'Hate'])
26
27 fig, ax = plt.subplots(figsize=(6, 6))
28 disp.plot(ax=ax, cmap='Blues', values_format='d')
29 plt.title("Confusion Matrix - Test Set")
30 plt.show()
```

Kode 4.17: Evaluasi *metrics* pada data uji

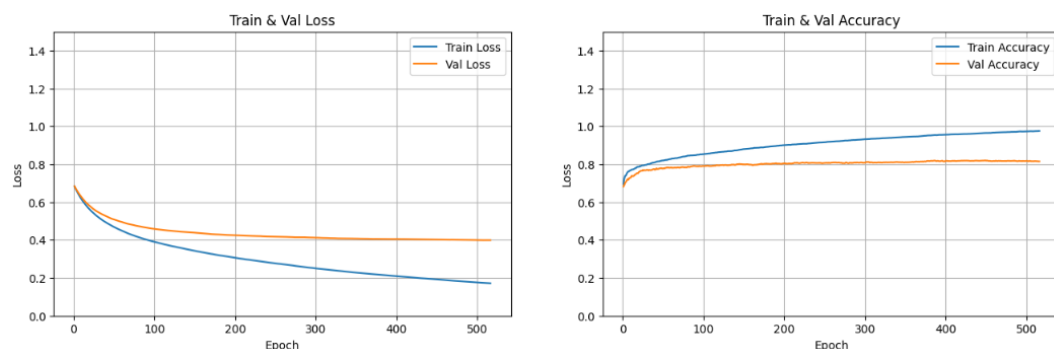
Kode 4.17 menunjukkan proses evaluasi model terhadap data uji, dengan mengukur empat metrik utama yaitu *accuracy*, *precision*, *recall*, dan *F1-score*. Prediksi dilakukan berdasarkan probabilitas yang dihasilkan oleh model XGBoost, dengan ambang batas sebesar 0.5. Seluruh nilai evaluasi disimpan ke dalam struktur *history* untuk kepentingan pelacakan dan analisis lebih lanjut.

4.4 Uji Coba dan Evaluasi Model

Pada bagian ini menjelaskan secara sistematis tahapan pengujian dan evaluasi terhadap model yang telah dikembangkan. Proses uji coba bertujuan untuk menilai kinerja model dalam mengklasifikasikan data sesuai dengan tujuan penelitian. Evaluasi dilakukan dengan menggunakan metrik yang relevan, seperti *accuracy*, *precision*, *recall*, dan *F1-score*, guna memberikan gambaran yang objektif mengenai efektivitas model.

A Hasil Evaluasi Model BERT + XGBoost

Berdasarkan Gambar 4.1 yang merupakan hasil evaluasi model klasifikasi yang diterapkan pada data validasi, diperoleh tingkat akurasi sebesar 81,32%, yang mengindikasikan bahwa proporsi prediksi yang benar terhadap seluruh data mencapai lebih dari 80%. Dalam hal kesalahan prediksi, nilai *log loss* pada data pelatihan tercatat sebesar 0,1609, sedangkan pada data validasi sebesar 0,3889. Pengujian dengan model *hybrid* ini dijadikan pembandingan dengan dua model lainnya sebagai *baseline* model. Nilai evaluasi *metrics* performa lainnya dari model BERT + XGBoost dihasilkan seperti pada Tabel 4.7. Nilai presisi sebesar 81,35% menunjukkan bahwa dari seluruh data yang diprediksi sebagai kelas “*Hate*”, sekitar 81% benar-benar merupakan data *hate*. Sementara itu, nilai *recall* sebesar 81,43% menggambarkan bahwa dari seluruh data aktual yang termasuk ke dalam kelas “*Hate*”, sekitar 81% berhasil dikenali dengan benar oleh model. Nilai *F1-score* sebesar 81,39% mengindikasikan adanya keseimbangan yang baik antara presisi dan *recall* dalam kinerja model. Selain itu, terdapat nilai lainnya yaitu AUC-ROC sebesar 0,8961 memperlihatkan kemampuan model yang tinggi dalam membedakan antara dua kelas (*Hate* dan *NonHate*) mendekati nilai maksimum yaitu 1.

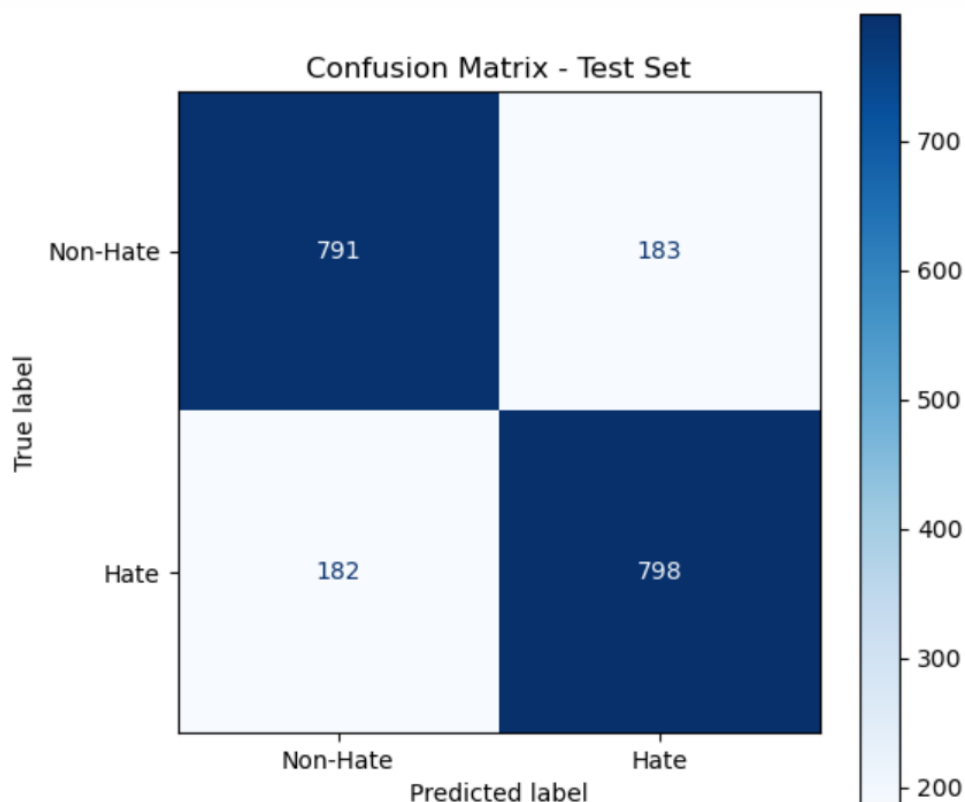


Gambar 4.1. Grafik *loss* dan akurasi *training* dan *validation* model bert+xcgboost

Tabel 4.7. Hasil evaluasi model pada data uji

Class	Precision	Recall	F1-score	Support
Non-hate	0.81	0.81	0.81	974
Hate-speech	0.81	0.81	0.81	980
Accuracy			0.81	1954

Berdasarkan Gambar 4.2 mengenai distribusi kelas dengan *confusin matrix*, terdapat 980 data untuk kelas “*Hate*” dan 974 data untuk kelas “*NonHate*”, menunjukkan distribusi kelas yang seimbang. Model berhasil melakukan prediksi yang benar terhadap 1.589 dari total 1.954 data. Analisis terhadap *confusion matrix* menunjukkan bahwa model mengklasifikasikan 791 data kelas “*NonHate*” dengan benar, namun keliru mengklasifikasikan 183 data kelas tersebut sebagai “*Hate*”. Sebaliknya, dari kelas “*Hate*”, sebanyak 798 data berhasil diklasifikasikan dengan benar, sedangkan 182 sisanya salah diklasifikasikan sebagai “*NonHate*”.



Gambar 4.2. *Confusion matrix* model bert+*xgboost*

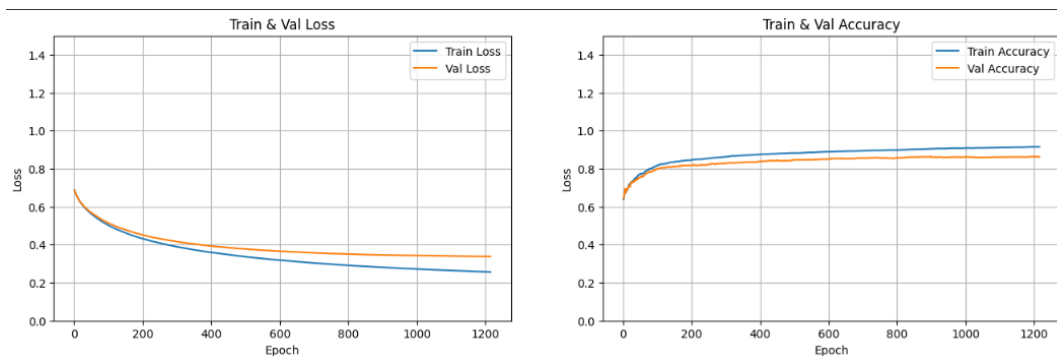
Secara keseluruhan, model menunjukkan performa yang konsisten dalam membedakan antara ujaran kebencian (*hate speech*) dan non-kebencian. Hal ini

tercermin dari distribusi prediksi yang seimbang serta nilai-nilai metrik evaluasi yang seragam antar kelas. Meskipun masih terdapat sejumlah kesalahan klasifikasi, khususnya dalam membedakan kasus-kasus ambiguitas antar kelas, hasil yang diperoleh mengindikasikan bahwa pendekatan yang digunakan mampu menangkap pola-pola relevan dalam data dengan tingkat akurasi yang baik.

B Hasil Evaluasi XGBoost TF-IDF

Pada Gambar 4.3 disajikan hasil evaluasi model yang diterapkan data validasi, diperoleh tingkat akurasi sebesar 85,57%, yang memberitahukan hasil prediksi yang benar terhadap seluruh data yang melebihi 85% yang dimana lebih baik dari model sebelumnya. Dalam kesalahan prediksi, nilai *loss* dalam pelatihan tercatat pada angka 0.2628, sedangkan pada *validation loss* menyentuh angka sebesar 0,3333 mengindikasikan bahwa model belajar dan memvalidasi dengan baik. Adapun matrices performa lainnya dari model XGBoost TF-IDF menghasilkan *metrics* seperti pada Tabel 4.8.

Berdasarkan hasil tabel tersebut, model menunjukkan performa yang baik dalam membedakan antara ujaran kebencian (*hate speech*) dan non-kebencian. Untuk kelas *hate speech*, model mencatat nilai presisi sebesar 88%, yang berarti model memprediksi 88% benar pada kategori *hate-speech*. Nilai *recall* untuk kelas ini mencapai 83%, menunjukkan bahwa dari seluruh data aktual yang termasuk dalam kelas *hate speech*, 83% berhasil diidentifikasi secara tepat. *F1-score* sebesar 0,85 mencerminkan keseimbangan yang baik antara presisi dan *recall* dalam mengenali kelas tersebut. Sementara itu, untuk kelas *nonhate*, model meraih presisi sebesar 83% dan *recall* sebesar 89%, dengan *F1-score* sebesar 0,86. Hal ini mengindikasikan bahwa model memiliki sedikit keunggulan dalam mengenali data *nonhate* dibandingkan *hate speech*. Secara keseluruhan, akurasi model mencapai 86% pada data uji, yang menunjukkan bahwa sebagian besar data berhasil diklasifikasikan dengan benar. Selain itu, nilai AUC-ROC sebesar 0,9347 menunjukkan bahwa model memiliki kemampuan diskriminatif yang sangat baik dalam membedakan antara dua kelas, mendekati nilai maksimum 1.



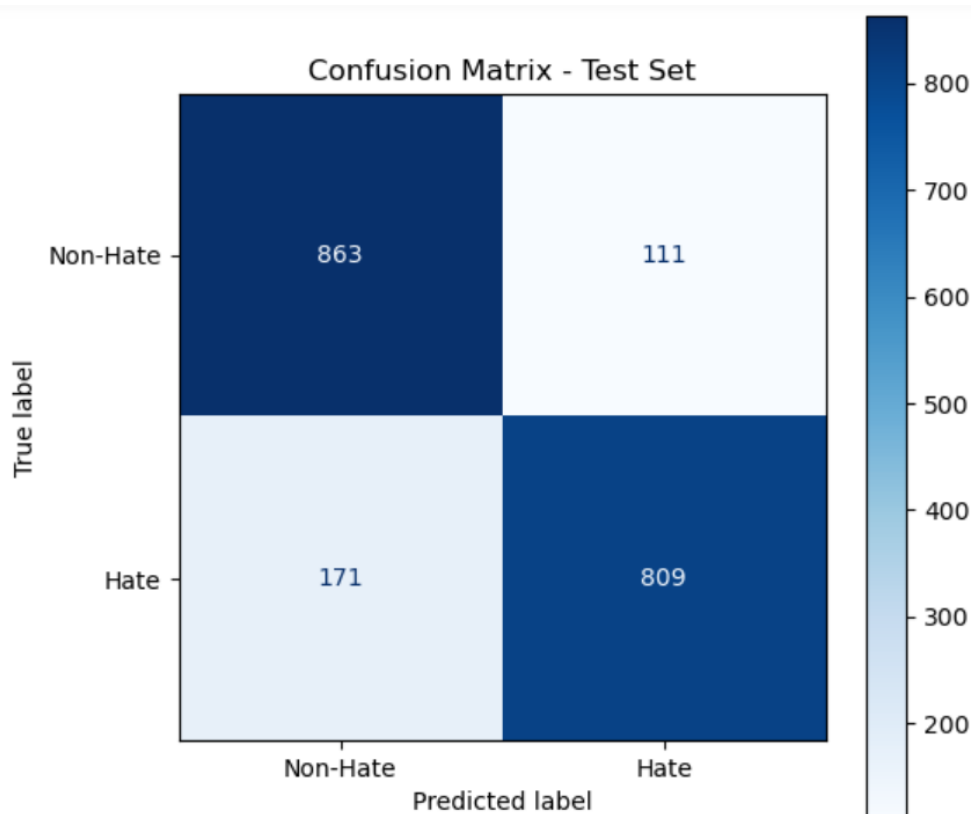
Gambar 4.3. Grafik *loss* dan akurasi *training* dan *validation* model xgboost tf-idf

Tabel 4.8. Hasil evaluasi model pada data uji

Class	Precision	Recall	F1-score	Support
Non-hate	0.83	0.89	0.86	974
Hate-speech	0.88	0.83	0.85	980
Accuracy			0.86	1954

Berdasarkan hasil evaluasi terhadap data uji yang terdiri dari 1.954 sampel, model XGBoost berhasil membuat 1.672 prediksi yang benar, yang menghasilkan tingkat akurasi sebesar 85,57%. Analisis lebih lanjut ditunjukkan melalui *confusion matrix*, di mana sebanyak 863 sampel dari kelas "NonHate" berhasil diklasifikasikan dengan benar, sementara 111 sampel lainnya keliru diklasifikasikan sebagai "Hate". Untuk kelas "Hate", sebanyak 809 sampel berhasil diidentifikasi secara tepat, sedangkan 171 sampel salah diklasifikasikan sebagai "NonHate".

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 4.4. *Confusion matrix* model xgboost tf-idf

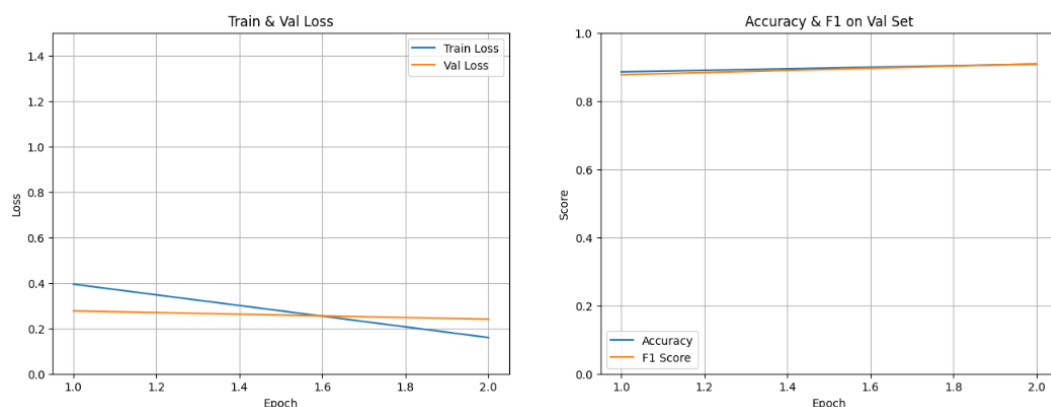
Secara keseluruhan, hasil evaluasi menunjukkan bahwa model memiliki kemampuan klasifikasi yang seimbang, dengan kinerja yang baik dalam mengenali kedua kelas, baik *hate speech* maupun *nonhate*. Jumlah prediksi yang benar yang tinggi serta distribusi kesalahan yang tidak terlalu dominan pada salah satu kelas mengindikasikan bahwa model mampu membedakan ujaran kebencian dan non-kebencian secara efektif. Kinerja ini memperkuat bahwa pendekatan XGBoost dengan representasi fitur TF-IDF memberikan performa yang andal dalam tugas klasifikasi biner. Selain itu, performa model ini menunjukkan sedikit peningkatan dibandingkan pendekatan *hybrid* sebelumnya yang menggabungkan BERT dengan XGBoost.

C Hasil Evaluasi BERT Finetuned

Gambar 4.5 menyajikan hasil evaluasi model BERT *Finetuned* selama proses pelatihan terhadap data validasi. Model menunjukkan stabilitas akurasi pada kisaran 88–89% selama dua *epoch* terakhir, yang mengindikasikan bahwa model memiliki kemampuan prediksi yang sangat baik, dengan tingkat keberhasilan

klasifikasi mendekati 90%. Selisih antara *training loss* (0,1678) dan *validation loss* (0,2806) relatif kecil, yang menunjukkan bahwa model mampu mempelajari pola data secara efektif baik pada data pelatihan maupun data validasi. Hasil ini menunjukkan bahwa model BERT *Finetuned* merupakan pendekatan terbaik dibandingkan tiga model lainnya yang telah dievaluasi sebelumnya.

Berdasarkan Tabel 4.9, performa model dalam membedakan ujaran kebencian (*hate speech*) dan non-kebencian tercatat sangat baik. Untuk kelas "Hate", model mencatat nilai presisi sebesar 90% dan *recall* sebesar 88%, menghasilkan *F1-score* sebesar 0,89. Nilai ini mencerminkan keseimbangan yang solid antara presisi dan *recall*, yang mengindikasikan bahwa model tidak hanya akurat dalam memprediksi *hate-speech*, tetapi juga sensitif terhadap keberadaannya dalam data. Sementara itu, untuk kelas "NonHate", model meraih presisi sebesar 89% dan *recall* sebesar 90%, dengan *F1-score* yang juga mencapai 0,89. Nilai metrik yang seimbang ini menunjukkan bahwa model bekerja konsisten dalam mengenali kedua kategori kelas. Secara keseluruhan model mencapai akurasi 89% untuk pengklasifikasian dengan benar. Selain itu, terdapat faktor penilaian AUC-ROC sebesar 0.9584 menunjukan model sangat percaya diri dalam membedakan antara dua kelas tersebut.



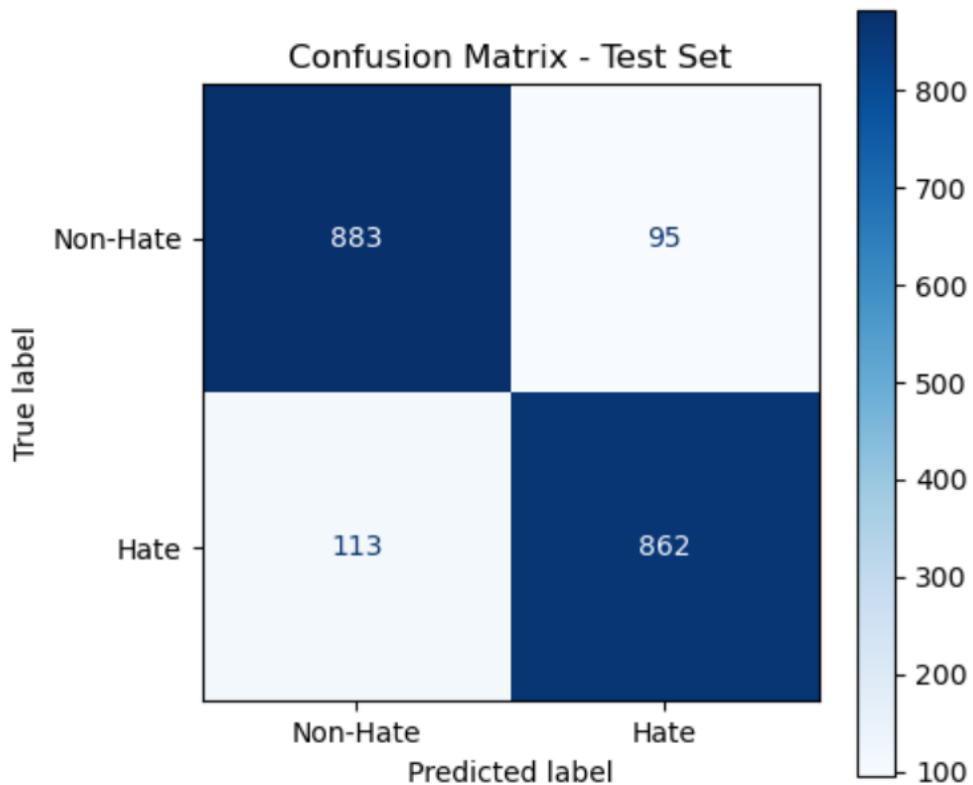
Gambar 4.5. Grafik *loss* dan akurasi *training* dan *validation* model bert finetuned

Evaluasi terhadap data uji yang terdiri dari 1.953 sampel memperkuat hasil tersebut. Model BERT *Finetuned* berhasil mengklasifikasikan sebanyak 1.737 sampel dengan benar, menghasilkan akurasi sebesar 88,99%. Dari *confusion matrix* pada Gambar 4.6, diketahui bahwa sebanyak 883 sampel kelas "NonHate" diklasifikasikan dengan tepat, sementara 95 sampel salah diklasifikasikan sebagai "Hate". Pada kelas "Hate", terdapat 862 prediksi benar dan 113 prediksi yang keliru diklasifikasikan sebagai "NonHate". Keseimbangan kesalahan ini memperlihatkan

bahwa model tidak bias terhadap salah satu kelas, dan mampu menangani distribusi kelas yang seimbang dengan baik.

Tabel 4.9. Hasil evaluasi model pada data uji

Class	Precision	Recall	F1-score	Support
Non-hate	0.89	0.90	0.89	978
Hate-speech	0.90	0.88	0.89	976
Accuracy			0.89	1954



Gambar 4.6. Confusion matrix model bert finetuned

Secara keseluruhan, model BERT *Finetuned* menunjukkan performa klasifikasi terbaik dibandingkan seluruh model yang telah diuji dalam mengenali dua kategori, yaitu *Hate speech* dan *Non Hate*. Model ini unggul dari segi akurasi, keseimbangan metrik antar kelas, serta kemampuan generalisasi yang baik tanpa menunjukkan bias terhadap salah satu kelas. Dengan demikian, model BERT *Finetuned* dapat dianggap sebagai pendekatan paling andal dan efektif dalam tugas klasifikasi ujaran kebencian pada penelitian ini.

4.4.1 Rangkuman dan Evaluasi Akhir Antar Model

Pada bagian ini disajikan rangkuman hasil evaluasi akhir dari tiga pendekatan model yang digunakan dalam tugas klasifikasi *hate speech*, yaitu kombinasi BERT + XGBoost, XGBoost dengan fitur TF-IDF, dan BERT *finetuned*. Evaluasi dilakukan berdasarkan metrik performa klasifikasi yang mencakup akurasi, presisi, *recall*, dan *F1-score*. Masing-masing metrik dihitung untuk dua kategori kelas, yaitu *Hate-Speech* dan *NonHate*, serta dihitung nilai rata-rata untuk memberikan gambaran umum kinerja tiap model. Tabel 4.10 merangkum hasil evaluasi tersebut.

Tabel 4.10. Evaluasi model untuk klasifikasi *hate-speech* dan *non-hate*

Skenario	Sentimen	Akurasi	Metrics (%)		
			Precision	Recall	F1-Score
BERT + XGBoost	Hate-Speech	81%	81%	81%	81%
	Non-Hate		81%	81%	91%
	Avg.		81%	81%	81%
XGBoost TF-IDF	Hate-Speech	85%	83%	89%	86%
	Non-Hate		88%	83%	85%
	Avg.		86%	86%	86%
BERT Finetuned	Hate-Speech	90%	89%	90%	89%
	Non-Hate		90%	89%	89%
	Avg.		90%	90%	89%

Berdasarkan Tabel 4.10, model BERT *Finetuned* menunjukkan kinerja terbaik secara keseluruhan dengan akurasi sebesar 90% dan skor evaluasi rata-rata yang seimbang pada semua metrik (*precision*, *recall*, dan *F1-score* sebesar 89–90%). Model XGBoost TF-IDF menempati posisi kedua setelah BERT *finetuned* dengan akurasi 85% dan skor metrik yang cukup kompetitif, khususnya pada kelas *Hate Speech* yang menunjukkan *recall* sebesar 89%. Sementara itu, model BERT + XGBoost menunjukkan performa paling rendah di antara ketiganya, dengan akurasi hanya 81% dan skor metrik yang relatif seragam namun lebih rendah.

Hasil ini menunjukkan bahwa penggabungan representasi vektor *embeddings* dari BERT dengan model pembelajaran klasik seperti XGBoost tidak secara otomatis menghasilkan peningkatan performa. Salah satu penyebabnya

adalah kurang optimalnya proses integrasi representasi vektor kontekstual dengan arsitektur model pembelajaran klasik yang tidak dirancang untuk memanfaatkan konteks semantik mendalam sebagaimana halnya jaringan neural transformer. Selain itu, pendekatan ini juga tidak melibatkan pelatihan ulang parameter BERT, sehingga potensi penuh dari representasi bahasa yang kontekstual tidak dapat dimanfaatkan secara maksimal.

