BAB 4
HASIL PENELITIAN

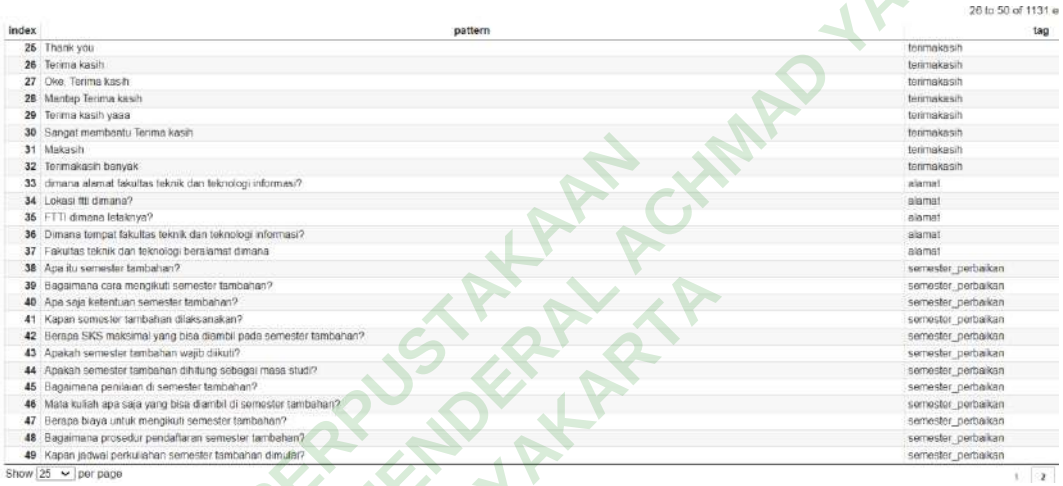
4.1 Ringkasan Hasil Penelitian

Penelitian ini mengembangkan chatbot layanan informasi mahasiswa menggunakan algoritma LSTM. Data dikumpulkan dari situs web Fakultas Teknik dan Teknologi Informasi UNJAYA dan diolah menjadi 1437 pertanyaan dan jawaban. Tahapan penelitian meliputi penghapusan tanda baca, tokenisasi, stemming, encoding dengan Word2Vec, dan penyeimbangan data dengan RandomOverSampler. Data kemudian dibagi menjadi data latih dan data uji. Model LSTM dibentuk dan dikompilasi menggunakan *optimizer* Adam. Hasil evaluasi menunjukkan akurasi 97.76%, presisi 98.34%, dan recall 97.76%. Model di-deploy dalam aplikasi web menggunakan Streamlit, dan pengujian Black Box memastikan aplikasi berfungsi dengan baik. Penelitian ini menunjukkan bahwa chatbot berbasis LSTM efektif dalam menyediakan informasi akademik secara real-time dan dapat meningkatkan kepuasan mahasiswa terhadap layanan informasi kampus.

4.2 Hasil Pengumpulan Data

Data yang didapat merupakan informasi mengenai berbagai layanan mahasiswa seperti panduan MBKM, pembuatan transkrip nilai, pengajuan surat keterangan, prosedur pembayaran SPP, informasi mengenai tugas akhir, serta layanan-layanan lainnya yang diperlukan mahasiswa. Data ini penting bagi mahasiswa karena berisi panduan dan prosedur yang membantu mereka dalam mengakses layanan-layanan tersebut. Data diperoleh dari Fakultas Teknik dan Teknologi Informasi UNJAYA yang diambil dari situs web https://ftti.unjaya.ac.id/ dalam bentuk file pdf dan dokumen lainnya. Setelah data layanan mahasiswa didapatkan, langkah selanjutnya adalah membuat data pertanyaan dan data jawaban. Pertanyaan dibuat berkaitan dengan data layanan seperti prosedur pengajuan surat keterangan, tahapan pembayaran SPP, panduan MBKM, serta prosedur pembuatan transkrip nilai. Setelah mendapatkan data dan melakukan pemahaman terhadap data, tahap berikutnya adalah persiapan data. Data yang

dijadikan acuan untuk membuat jawaban dan data pertanyaan selanjutnya dibuat menjadi satu file JSON yang terdiri dari intent, tag, *pattern*, dan response seperti pada gambar yang kemudian akan diproses dengan Python. Jumlah data yang sudah dibuat adalah 1437 data yang terdiri dari data tag sebagai label, *patterns* sebagai pertanyaan, dan responses sebagai jawaban. Data tersebut dibuat dalam format JSON kemudian dikonversi menjadi *dataframe* Pandas. Berikut merupakan data JSON yang sudah dikonversi menjadi *dataframe* menjadi sebuah list *patterns* dan tag yang terlihat pada Gambar 4.1.



26 to 50 of 1131 e

index	pattern	tag
25	Thank you	terimakasih
26	Terima kasih	terimakasih
27	Oke. Terima kasih	terimakasih
28	Mantap Terima kasih	terimakasih
29	Terima kasih yaaa	terimakasih
30	Sangat membantu Terima kasih	terimakasih
31	Makasih	terimakasih
32	Terimakasih banyak	terimakasih
33	dimana alamat fakultas teknik dan teknologi informasi?	alamat
34	Lokasi fti dimana?	alamat
35	FTTI dimana letaknya?	alamat
36	Dimana tempat fakultas teknik dan teknologi informasi?	alamat
37	Fakultas teknik dan teknologi bersamat dimana	alamat
38	Apa itu semester tambahan?	semester_perbaikan
39	Bagaimana cara mengikuti semester tambahan?	semester_perbaikan
40	Apa saja ketentuan semester tambahan?	semester_perbaikan
41	Kapan semester tambahan dilaksanakan?	semester_perbaikan
42	Berapa SKS maksimal yang bisa diambil pada semester tambahan?	semester_perbaikan
43	Apakah semester tambahan wajib diikuti?	semester_perbaikan
44	Apakah semester tambahan dihitung sebagai masa studi?	semester_perbaikan
45	Bagaimana penilaian di semester tambahan?	semester_perbaikan
46	Mata kuliah apa saja yang bisa diambil di semester tambahan?	semester_perbaikan
47	Berapa biaya untuk mengikuti semester tambahan?	semester_perbaikan
48	Bagaimana prosedur pendaftaran semester tambahan?	semester_perbaikan
49	Kapan jadwal perkuliahan semester tambahan dimulai?	semester_perbaikan

Show 25 per page

Gambar 4.1.*Dataframe*

Pada Gambar 4.1 merupakan gambar tabel *dataframe* yang dihasilkan dari dataset dengan format JSON.

4.3 Tahap Pemrosesan Data

Data pertanyaan yang telah diubah menjadi *list* kemudian dibersihkan. Proses pembersihan data terlihat pada gambar 4.2.

```
def clean_text(text):
    text = text.lower()  # Lowercasing
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    return text
```

Gambar 4.2. Kode Program Pembersihan Data

Fungsi ‘clean_text’ digunakan untuk membersihkan teks dari karakter yang tidak diinginkan agar teks lebih konsisten untuk analisis. Pertama, fungsi ini mengonversi semua huruf menjadi huruf kecil untuk menghindari perbedaan antara huruf kapital dan huruf kecil. Ekspresi reguler (regex) adalah pola yang digunakan untuk mencari, mengedit, dan memanipulasi teks berdasarkan aturan tertentu. Regex memungkinkan deteksi dan pengubahan teks secara efisien. Pada fungsi ini menggunakan regex untuk menghapus semua tanda baca dengan menyisakan hanya huruf, spasi, dan angka-angka Dengan demikian, fungsi ‘clean_text’ menghasilkan teks yang konsisten tanpa huruf kapital, tanda baca, angka, dan spasi berlebih, Hasil dari pemrosesan data dapat dilihat pada Gambar 4.3.

original_pattern	cleaned_pattern
Thank you	thank you
Terima kasih	terima kasih
Oke, Terima kasih	oke terima kasih
Mantap Terima kasih	mantap terima kasih
Terima kasih yaaa	terima kasih yaaa
Sangat membantu Terima kasih	sangat membantu terima kasih
Makasih	makasih
Terimakasih banyak	terimakasih banyak
Apa itu FTTI?	apa itu ftti
Apa kepanjangan FTTI?	apa kepanjangan ftti
Apa saja program studi di FTTI?	apa saja program studi di ftti
Kapan FTTI didirikan?	kapan ftti didirikan
Bagaimana sejarah FTTI?	bagaimana sejarah ftti
Berapa program studi di FTTI?	berapa program studi di ftti

Gambar 4.3 Hasil Pembersihan Data

Dengan pembersihan data, dapat memudahkan proses analisis teks selanjutnya seperti yang terlihat pada Gambar 4.4.

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

factory = StemmerFactory()
stemmer = factory.create_stemmer()
stop_words = set(stopwords.words('indonesian'))
```

Gambar 4.4. *Library* Pemrosesan Data Kata Dasar

Penggunaan modul 'Sastrawi' terlihat pada Gambar 4.4, di mana teknik stemming diterapkan untuk mengubah kata-kata dalam teks berbahasa Indonesia menjadi bentuk dasarnya. *Stemming* merupakan metode dalam pengolahan bahasa alami yang bertujuan untuk menemukan kata dasar dengan menghapus infleksi kata dan afiksasi. Selanjutnya, daftar *stopwords* dari pustaka NLTK (*Natural Language Toolkit*) untuk bahasa Indonesia digunakan dan disimpan dalam struktur data himpunan. *Stopwords* merupakan kategori kata-kata umum yang sering diabaikan dalam analisis teks karena kurangnya informasi yang signifikan, seperti 'dan', 'yang', dan 'di'. Proses ini bertujuan untuk menyederhanakan teks dengan menghilangkan variasi bentuk kata dan memastikan bahwa hanya kata-kata yang memiliki nilai informasi yang relevan yang dipertimbangkan dalam analisis teks.

Selanjutnya, untuk proses lanjutan seperti yang ditunjukkan pada Gambar 4.5, fungsi ‘process_text’ digunakan untuk mengolah teks lebih lanjut. Fungsi ini memecah teks menjadi token-token atau kata-kata individual menggunakan ‘word_tokenize’, yang merupakan teknik pengolahan bahasa alami untuk memperoleh unit-unit dasar teks. Setiap token kemudian dijalani melalui proses stemming menggunakan metode ‘stemmer.stem(word)’, sambil memastikan bahwa hanya token-token yang tidak termasuk dalam daftar stopwords yang dipertimbangkan. Dengan demikian, fungsi ‘process_text’ bertujuan untuk menyederhanakan kata-kata ke bentuk dasarnya dan memastikan bahwa hanya kata-kata yang memiliki nilai informasi yang relevan yang dipertimbangkan dalam analisis teks.


```
def process_text(text):
    tokens = word_tokenize(text)
    tokens = [stemmer.stem(word) for word in tokens if word not in stop_words]
    return tokens
```

Gambar 4.5. Pemrosesan Data Kata Dasar

Hasil dari keseluruhan pemrosesan data ditampilkan pada Gambar 4.6 dalam bentuk *Dataframe*. *Dataframe* ini menyajikan pola asli (*original_pattern*), pola yang telah diproses (*pattern*), dan tag yang sesuai (*tag*). Dengan demikian,

keseluruhan proses dari pembersihan teks hingga penyajian hasil akhir dalam bentuk terstruktur telah ditunjukkan secara komprehensif.

original_pattern	pattern
Thank you	thank you
Terima kasih	terima kasih
Oke, Terima kasih	oke terima kasih
Mantap Terima kasih	mantap terima kasih
Terima kasih yaaa	terima kasih yaaa
Sangat membantu Terima kasih	bantu terima kasih
Makasih	makasih
Terimakasih banyak	terimakasih
Apa itu FTTI?	ftti
Apa kepanjangan FTTI?	panjang ftti
Apa saja program studi di FTTI?	program studi ftti
Kapan FTTI didirikan?	ftti diri
Bagaimana sejarah FTTI?	sejarah ftti
Berapa program studi di FTTI?	program studi ftti

Gambar 4.6. Hasil Pemrosesan Data Kata Dasar

Setelah data dibersihkan maka akan dilakukan proses *tokenization* untuk mengubah setiap kalimat menjadi urutan bilang bulat, proses tersebut terlihat pada Gambar 4.6.

```
# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['pattern'])
vocabulary = len(tokenizer.word_index) + 1
```

Gambar 4.7. Proses *Tokenization*

Terlihat pada Gambar 4.7, proses tokenisasi mengonversi teks menjadi serangkaian token numerik yang mewakili kata-kata dalam teks tersebut. *Tokenizer* akan memetakan setiap kata unik dalam teks ke sebuah indeks numerik dan menyimpan peta kata ke indeks ini dalam *word_index*. Variabel *vocabulary* menyimpan jumlah total kata unik yang telah di-tokenisasi, ditambah satu untuk memperhitungkan indeks yang dimulai dari 1. Hasil proses tokenization dapat dilihat pada Gambar 4.8.

```
{'program': 1, 'informasi': 2, 'magang': 3, 'mahasiswa': 4,
'daftar': 5, 'unjaya': 6, 'syarat': 7, 'kuliah': 8, 'studi':
9, 'teknik': 10, 'ikut': 11, 'mbkm': 12, 'kampus': 13, 'biaya':
14, 'semester': 15, 'prodi': 16, 'mana': 17, 'krs': 18,
'fakultas': 19, 'iisma': 20, 'teknologi': 21, 'merdeka': 22,
'spp': 23, 'ajar': 24, 'kepala': 25, 'urus': 26, 'cuti': 27,
'pmmb': 28, 'prosedur': 29, 'sistem': 30, 'tukar': 31, 'baru':
32, 'industri': 33, 'prestasi': 34, 'pustaka': 35, 'ketua':
36, 'jadwal': 37, 'langkahlangkah': 38, 'informatika': 39,
'nilai': 40, 'beasiswa': 41, 'surat': 42, 'independen': 43,
'ftti': 44, 'bayar': 45, 'dosen': 46, 'mitra': 47, 'bebas':
48, 'data': 49, 'dokumen': 50, 'sertifikat': 51}
```

Gambar 4.8 Hasil Proses *Tokenization*

Pada langkah selanjutnya, seperti yang ditunjukkan pada Gambar 4.9, proses *embedding* dilakukan menggunakan model Word2Vec dari pustaka gensim. Proses *embedding* bertujuan untuk mengubah setiap kata dalam teks menjadi vektor numerik berdimensi tetap, yang merepresentasikan makna semantik kata-kata tersebut berdasarkan konteksnya dalam teks.

```python
from gensim.models import Word2Vec

# Prepare the sentences for Word2Vec
sentences = [process_text(pattern) for pattern in df['pattern']]

# Training the Word2Vec model
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
embedding_dim = word2vec_model.vector_size
```

Gambar 4.9. Proses *Embedding*

Untuk mempersiapkan proses *embedding*, teks dalam kolom '*pattern*' diproses menggunakan fungsi ‘process_text’ untuk menghilangkan *stopwords* dan melakukan *stemming*, menghasilkan daftar token yang disimpan dalam variabel *sentences.* Model Word2Vec kemudian dilatih menggunakan daftar kalimat ini dengan parameter-parameter sebagai berikut:

1. ‘vector_size=100’: Parameter ini menentukan dimensi vektor *embedding* yang dihasilkan oleh model Word2Vec. Dimensi ini merepresentasikan kata-kata. Semakin tinggi dimensinya, maka semakin banyak informasi yang direpresentasikan, namun juga memerlukan lebih banyak sumber daya komputasi.

2. ‘window=5’: Parameter ini menentukan ukuran jendela konteks, yaitu jumlah kata di sekitar kata target yang akan dipertimbangkan oleh model selama pelatihan. Dalam hal ini, model akan mempertimbangkan 5 kata sebelum dan 5 kata setelah kata target sebagai konteksnya. Jendela yang lebih besar dapat menangkap konteks yang lebih luas.
3. ‘min_count=1’: Parameter ini menentukan frekuensi minimum sebuah kata untuk dimasukkan dalam pelatihan model. Dengan ‘min_count=1’ semua kata yang muncul setidaknya satu kali akan dimasukkan dalam pelatihan. Jika nilai ini lebih tinggi, kata-kata yang jarang muncul akan diabaikan, yang dapat membantu mengurangi noise namun juga dapat menyebabkan hilangnya informasi penting.
4. ‘workers=4’: Parameter ini menentukan jumlah thread paralel yang digunakan selama pelatihan model. Dengan menggunakan 4 worker, proses pelatihan akan berjalan secara paralel menggunakan 4 CPU core, yang dapat mempercepat proses pelatihan. Jumlah worker yang optimal tergantung pada jumlah core yang tersedia di mesin yang digunakan.

Setelah model dilatih dengan parameter-parameter tersebut, hasil dari proses pelatihan ini adalah vektor *embedding* yang berdimensi 100, yang disimpan dalam variabel *embedding_dim*. Berikut adalah contoh hasil proses *embedding* untuk kata ‘program’ yang dapat dilihat pada Gambar 4.10.

```
Some words and their vectors:
Word: program
Vector: [-2.98538920e-03  4.61875834e-03  3.86416493e-03  9.31920204e-03
 -6.82228152e-03 -1.50913401e-02  8.25147517e-03  1.77367292e-02
 -8.40903353e-03 -7.85959139e-03  5.27445693e-03 -6.70244358e-03
 -4.95982915e-03  9.08838399e-03 -3.43597122e-03 -5.63268736e-03
  4.54234285e-03 -1.63475145e-03 -1.01925051e-02 -1.89743079e-02
  9.18483362e-03  5.96578978e-03  5.85486973e-03 -7.37274298e-04
  4.74342797e-03 -4.11277497e-03 -3.67705734e-03  1.76084775e-03
 -1.10303769e-02 -4.00760584e-03 -4.78254026e-03  3.13978700e-04
  1.32688945e-02 -1.16857486e-02 -5.42366737e-03  4.95585985e-03
  8.97705741e-03 -7.36676389e-03 -1.62836222e-03 -1.39757628e-02
 -1.06894225e-02 -2.00823357e-04 -9.57237277e-03 -2.64995568e-03
  3.97083303e-03 -2.64953147e-03 -9.78248660e-03  7.28466874e-03
  8.99551064e-03  1.22647593e-02 -5.08234277e-03  1.04819878e-03
 -1.50425872e-03 -8.90984840e-04  7.28430273e-03 -7.22256780e-04
  5.65691944e-03 -7.09268032e-03 -8.34649336e-03  1.05243362e-02
  1.15338771e-03  2.06238893e-03 -3.79036693e-03 -9.99892130e-03
 -6.84609869e-03  6.87504094e-03  3.34784163e-05  1.26225865e-02
 -1.01347063e-02  7.05494825e-03  3.01911985e-03  1.01699578e-02
  2.68950290e-03 -8.71956907e-03  1.16300313e-02  4.48750844e-03
  6.97325543e-03  2.69253389e-04 -6.73348270e-03 -8.92885122e-03
 -8.31703306e-04  3.57710524e-03 -1.73455826e-03  1.37450062e-02
 -6.61610719e-03  7.02297490e-04  1.00913765e-02 -5.06577853e-05
  3.81369307e-03  8.87535419e-03  7.30091706e-03  3.47368978e-03
  2.33241078e-03  2.26682774e-03  1.74444299e-02  8.59314017e-03
 -6.45466149e-03 -8.92545283e-03  9.36836354e-04  3.83023545e-03]
```

Gambar 4.10 Hasil Proses *Embedding*

Pada tahap berikutnya, yang ditunjukkan pada Gambar 4.11, token-token yang telah dihasilkan oleh *tokenizer* dikonversi menjadi urutan numerik menggunakan metode *texts_to_sequences*. Setiap kalimat dalam kolom *pattern* diubah menjadi sebuah urutan indeks numerik yang mewakili kata-kata dalam kalimat tersebut. Variabel *train* menyimpan daftar dari urutan-urutan ini. Langkah berikutnya adalah menentukan panjang maksimum dari urutan token yang ada menggunakan *max(len(seq) for seq in train)*, yang menyimpan nilai panjang maksimum dalam variabel *max_sequence_length*.

```
train = tokenizer.texts_to_sequences(df['pattern'])
max_sequence_length = max(len(seq) for seq in train)
X = pad_sequences(train, maxlen=max_sequence_length, padding='post')
```

Gambar 4.11. Proses *Padding Text*

Terlihat pada Gambar 4.11. semua urutan token dipadatkan (*padded*) agar memiliki panjang yang sama dengan *max_sequence_length* menggunakan fungsi *pad_sequences*. Padding dilakukan di akhir urutan *(padding='post')* untuk memastikan bahwa semua urutan memiliki panjang yang seragam, serta hasil dari

proses padding disimpan dalam variabel X. Hasil dari proses *padding text*, dapat dilihat pada Gambar 4.12.

```
Original: Selamat pagi
Padded: [158 289   0   0   0   0   0   0   0]

Original: Selamat siang
Padded: [158 290   0   0   0   0   0   0   0]

Original: Selamat sore
Padded: [158 291   0   0   0   0   0   0   0]

Original: Selamat malam
Padded: [158 292   0   0   0   0   0   0   0]

Original: Good morning
Padded: [185 293   0   0   0   0   0   0   0]

Original: Good afternoon
Padded: [185 294   0   0   0   0   0   0   0]
```

Gambar 4.12 Hasil *Padding Text*

. Selanjutnya adalah proses *Encoding*, proses tersebut dapat dilihat pada Gambar 4.13.

```
# Encode labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['tag'])
y = to_categorical(y, num_classes=len(label_encoder.classes_))
```

Gambar 4.13. Proses *Encoding*

Pada tahap ini, seperti yang ditunjukkan pada Gambar 4.13, dilakukan proses encoding terhadap label atau kategori yang terdapat dalam kolom 'tag' dari *Dataframe* df. ‘LabelEncoder’ digunakan untuk mengonversi label kategori menjadi bilangan bulat. Setiap kategori akan diberikan sebuah bilangan bulat unik berdasarkan urutan kemunculan dalam data. Selanjutnya, hasil dari ‘LabelEncoder’ diubah menjadi bentuk *one-hot encoding* menggunakan fungsi ‘to_categorical’. Proses ini mengubah bilangan bulat yang merepresentasikan label menjadi vektor biner dengan panjang yang sesuai dengan jumlah kelas yang ada seperti pada Gambar 4.14 hasil dari proses *encoding*. Tahap selanjutnya yaitu proses *oversampling* yang dapat dilihat pada Gambar 4.10

```
Encoded labels:
Original: greeting -> Encoded: 41
Original: goodbye -> Encoded: 40
Original: terimakasih -> Encoded: 148
Original: ftti -> Encoded: 38
Original: alamat -> Encoded: 7
Original: jadwal_uas -> Encoded: 45
Original: jadwal_uts -> Encoded: 46
Original: semester_perbaikan -> Encoded: 139
Original: kalender_akademik -> Encoded: 47
Original: pkm -> Encoded: 124
Original: dokumen_penilaian -> Encoded: 30
Original: template_laporan_magang -> Encoded: 147
Original: Surat_Keterangan_Aktif_Kuliah -> Encoded: 0
Original: Surat_Keterangan_lulus -> Encoded: 2
Original: Surat_Keterangan_Cuti -> Encoded: 1
Original: pengajuan_cuti -> Encoded: 100
Original: isi_form_cuti -> Encoded: 43
Original: konfirmasi_cuti -> Encoded: 65
```

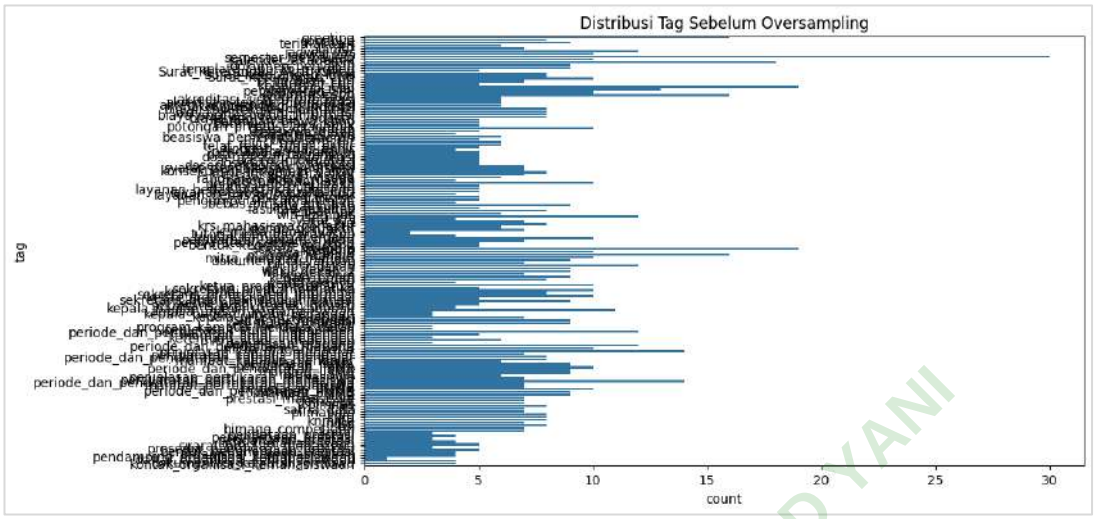
Gambar 4.14 Hasil Proses *Encoding*

```
# Oversampling
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)
```

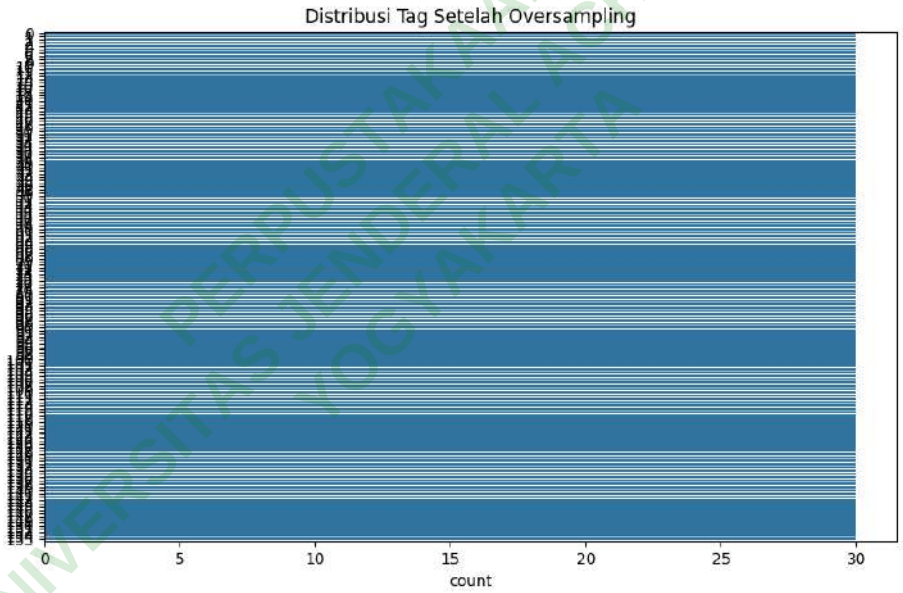
Gambar 4.15. Proses *Oversampling*

Selanjutnya, seperti yang ditunjukkan pada Gambar 4.15, dilakukan proses *oversampling* menggunakan metode RandomOverSampler. *Oversampling* adalah teknik dalam pengelolaan ketidakseimbangan kelas (*class imbalance*) di mana kelas minoritas (kelas dengan frekuensi rendah) disalin beberapa kali hingga seimbang dengan kelas mayoritas. Tujuan dari *oversampling* adalah untuk meningkatkan representasi kelas minoritas sehingga model dapat belajar dengan lebih baik membedakan kelas-kelas yang berbeda.

Metode ‘fit_resample’ dari RandomOverSampler digunakan untuk menyesuaikan (*fit*) data *input* (X dan y) dan melakukan *oversampling* pada data. Variabel X_resampled dan y_resampled menyimpan hasil dari proses *oversampling*, di mana kelas minoritas telah diperluas untuk mencapai proporsi yang setara dengan kelas mayoritas. Hasil sebelum dilakukan *oversampling* dan sesudah dilakukan o*versampling* pada kelas dapat dilihat pada Gambar 4.16 dan Gambar 4.17.



Gambar 4.16 Distribusi Kelas Sebelum *Oversampling*



Gambar 4.17 Distribusi Kelas Sesudah *Oversampling*

Selanjutnya adalah proses pembagian data seperti yang terlihat pada Gambar 4.18.

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
                                                    test_size=0.2, random_state=42)
```

Gambar 4.18 Proses *Train Test Split*

Pada tahap ini, seperti yang ditunjukkan pada Gambar 4.18, dilakukan proses pemisahan data menjadi data latih (*training*) dan data uji (*testing*) menggunakan fungsi 'train_test_split'. Pada proses pemisahan data ini mengambil beberapa argumen penting yaitu:

1. X_train: Data masukan yang telah dihasilkan dari proses tokenisasi dan padding digunakan untuk melatih model. Pada kode ini, X_resampled adalah data fitur yang sudah melewati proses oversampling dan padding. Data yang digunakan untuk proses ini adalah X_resampled (fitur) dan y_resampled (label) yang telah mengalami proses *oversampling* pada tahap sebelumnya.
2. y_train: Data keluaran atau label yang telah diubah menjadi angka melalui proses encoding digunakan sebagai target yang sesuai dengan data latih. y_resampled adalah label yang telah di-*oversample* dan di-*encoding*
3. test_size: Proporsi data yang digunakan sebagai data uji yaitu 20% dari total data yang akan dialokasi sebagai data uji, sementara 80% akan digunakan sebagai data latih
4. Random State: Parameter ini digunakan untuk memastikan hasil pemisahan data yang konsisten.

4.4 Tahap Pembentukan Model

Pada tahap ini, seperti yang ditunjukkan pada Gambar 4.12, dilakukan pembuatan model LSTM.

```
# Create the LSTM model
input_shape = X_train.shape[1]
i = Input(shape=(input_shape,))
x = Embedding(vocabulary, embedding_dim, weights=[embedding_matrix],
              input_length=max_sequence_length, trainable=True)(i)
x = LSTM(units=256, return_sequences=True)(x)
x = Dropout(rate=0.3)(x)
x = LSTM(units=128)(x)
x = Dropout(rate=0.3)(x)
x = Dense(units=192, activation='relu')(x)
x = Dropout(rate=0.3)(x)
output = Dense(len(label_encoder.classes_), activation="softmax")(x)
```

Gambar 4.19. Pembentukan Model LSTM

Pada Gambar 4.19, merupakan struktur model yang dipakai pada penelitian ini, pada struktur tersebut memiliki beberapa lapisan yaitu sebagai berikut:

1. *Input* Layer: *input* data dimasukkan ke dalam model menggunakan layer *Input* dengan bentuk *input*_shape=(*input*_shape,), di mana *input*_shape adalah panjang dari fitur *input* X_train
2. *Embedding* Layer: layer *Embedding* untuk mengubah representasi kata menjadi vektor numerik. Parameter yang digunakan meliputi:
   a. *vocabulary*: Jumlah kata unik dalam dataset.
   b. *embedding_dim*: Dimensi dari *embedding* vektor (dalam contoh ini, telah ditentukan sebelumnya).
   c. *weights=[embedding_matrix]*: Matriks *embedding* yang telah dipelajari sebelumnya (misalnya dari model Word2Vec).
   d. *input_length=max_sequence_length*: Panjang dari setiap urutan *input* yang sudah di*pad*.
   e. *trainable=True*: Memungkinkan pembelajaran ulang dari *embedding* di dalam model.
3. LSTM Layers: Model LSTM menggunakan dua layer LSTM berturut-turut yaitu:
   a. LSTM(units=256, return_sequences=True): LSTM pertama dengan 256 unit yang mengembalikan urutan keluaran (return_sequences=True), digunakan untuk mempertahankan urutan keluaran layer berikutnya.
   b. LSTM(units=128): LSTM kedua dengan 128 unit, yang tidak mengembalikan urutan keluaran (return_sequences default=False).
4. *Dropout Layers*: Setelah setiap layer LSTM, diterapkan *layer Dropout* dengan tingkat *dropout* sebesar 0.3 untuk mengurangi *overfitting* dengan secara acak mengabaikan sebagian unit selama pelatihan.
5. *Dense Layers*: Setelah proses LSTM dan *dropout*, diterapkan Dense dengan 192 unit dan fungsi aktivasi ReLU untuk memproses fitur yang diekstraksi dari LSTM.

6. *Output Layer*: *Layer output* terakhir menggunakan fungsi aktivasi *softmax* dengan jumlah unit sejumlah kelas yang berbeda (len(label_encoder.classes_)) yang bertujuan untuk menghasilkan probabilitas kelas-kelas yang mungkin dari *output* model.
7. *Model Compilation*: Keseluruhan model diinisialisasi menggunakan 'Model(i, *output*)' dengan i sebagai '*input'* dan '*output*' sebagai keluaran model.

```
# Train the model
history = model.fit(X_train, y_train, epochs=500, batch_size=32,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping, model_checkpoint, reduce_lr], verbose=2)
```

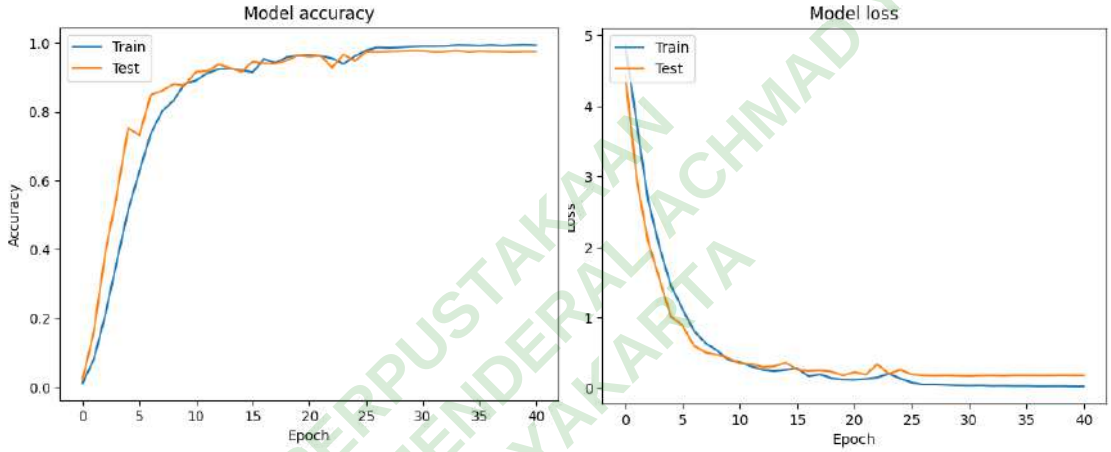
Gambar 4.20 *Training Model*

Pada Gambar 4.20 merupakah proses dari *training model*, pada tahap ini model LSTM yang telah dikompilasi akan dilatih menggunakan data latih '(X_train, y_train)'. Proses pelatihan ini melibatkan beberapa parameter penting sebagai berikut:

1. *Epochs*: Jumlah iterasi atau epoch yang digunakan untuk melatih model. Pada contoh ini, epochs=500 berarti model akan melalui dataset sebanyak 500 kali selama pelatihan.
2. *Batch Size*: Jumlah sampel data yang akan digunakan untuk menghitung gradien pada setiap iterasi. batch_size=32 berarti 32 sampel data akan digunakan pada setiap iterasi.
3. *Validation Data*: Data uji (X_test, y_test) digunakan untuk mengevaluasi kinerja model pada setiap *epoch*. Ini membantu untuk memonitor generalisasi model terhadap data yang tidak digunakan selama pelatihan.
4. *Callbacks*: *Callbacks* yang telah ditentukan sebelumnya *(early_stopping, model_checkpoint, reduce_lr)* digunakan untuk mengontrol proses pelatihan, termasuk menghentikan pelatihan lebih awal jika tidak ada peningkatan, menyimpan model terbaik, dan mengurangi learning rate jika diperlukan.

5. *Verbose*: Parameter *verbose=2* digunakan untuk mengontrol tampilan *output* selama pelatihan. *verbose=2* akan menampilkan satu bar untuk setiap *epoch*, sedangkan *verbose=1* akan menampilkan progress bar, dan *verbose=0* akan mematikan semua tampilan.

Pada pelatihan model mendapatkan *accuracy* sebesar 0,98 dan validasi *loss* sebesar 0,15, validasi *accuracy* sebesar 0,96 dan *loss* sebesar 0,05. Grafik hasil *accuracy* dan *loss* serta validasi *accuracy* dan validasi *loss* pada setiap *epoch* dapat dilihat pada Gambar 4.21.



Gambar 4.21. Grafik *Training* Model

Pada Gambar 4.21 merupakan hasil dari proses *training* model yang ditampilkan dalam bentuk grafik. Dari grafik akurasi, terlihat bahwa garis berwarna biru menunjukkan *accuracy* pelatihan yang baik, sedangkan garis berwarna oranye menunjukkan validasi *accuracy* yang sedikit lebih rendah. Pada grafik *loss*, garis biru menunjukkan *loss* pelatihan yang rendah, sementara garis oranye menunjukkan validasi *loss* yang sedikit lebih tinggi. Dari grafik tersebut, dapat disimpulkan bahwa proses pelatihan model menghasilkan kinerja yang baik tanpa tanda-tanda *overfitting.*

```
# 3. Menghitung accuracy, precision, dan recall
accuracy = accuracy_score(y_test_classes, y_pred)
precision = precision_score(y_test_classes, y_pred, average='weighted')
recall = recall_score(y_test_classes, y_pred, average='weighted')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

Gambar 4.22. Perhitungan *Confusion Matrix*

Pada Gambar 4.22 dilakukan perhitungan confusion matrix untuk mengevaluasi performa model LSTM yang telah dibangun, dilakukan perhitungan beberapa metrik evaluasi, yaitu *accuracy, precision,* dan *recall*. Metrik-metrik ini memberikan gambaran tentang seberapa baik model dalam melakukan klasifikasi data. Berikut penjelasan masing-masing metrik:

1. *Accuracy*: memberikan gambaran umum tentang performa model dimana ‘y_test_classes’ adalah label sebenarnya dan ‘y_pred’ adalah prediksi model.
2. *Precicion*: memahami seberapa banyak dari prediksi positif yang benar-benar positif. ‘average='weighted'’ digunakan untuk menghitung rata-rata tertimbang dari *precision* untuk setiap kelas, yang mempertimbangkan distribusi kelas yang tidak seimbang.
3. *Recall*: memahami seberapa baik model dalam menemukan semua kasus positif yang sebenarnya. ‘average='weighted'’ digunakan untuk menghitung rata-rata tertimbang dari recall untuk setiap kelas, yang mempertimbangkan distribusi kelas yang tidak seimbang.

Hasil perhitungan evaluasi matriks yang telah dilakukan disajikan dalam Tabel 4.1.

Tabel 4.1 Evaluasi Model

Accuracy	*Precision*	*Recall*
0.977	0,983	0,977

Berdasarkan evaluasi model menggunakan *confusion matrix*, didapatkan hasil yang sangat baik dengan nilai akurasi mencapai 97.76%. Nilai akurasi yang tinggi ini menunjukan bahwa model secara keseluruhan mampu mengklasifikasikan data dengan benar hampir pada semua kasus.

Selain itu, nilai presisi sebesar 98.34% menunjukan bahwa model jarang memberikan hasil positif yang salah. Dalam konteks layanan mahasiswa, ini berarti model sangat jarang memberikan informasi yang salah ketika mengidentifikasi layanan yang tersedia. Presisi yang tinggi sangat penting untuk mengurangi jumlah kesalahan positif (*false positives*), di mana model salah mengidentifikasi suatu layanan yang sebenarnya tidak ada.

Sementara itu, *recall* sebesar 97.76% menunjukkan bahwa model mampu menemukan sebagian besar dari semua *instance* yang sebenarnya positif. Ini berarti bahwa jika ada layanan tertentu yang harus diidentifikasi oleh *chatbot*, model ini mampu mengenalinya hampir pada semua kasus. *Recall* yang tinggi penting untuk memastikan bahwa layanan – layanan yang ada tidak terlewatkan oleh model.

Secara keseluruhan, metrik evaluasi yang tinggi ini menandakan bahwa model LSTM yang dibangun tidak hanya akurat dalam melakukan klasifikasi tetapi juga memiliki keseimbangan yang baik antara mengidentifikasi layanan dengan benar dan menghindari kesalahan identifikasi. Hal ini menunjukkan potensi yang besar untuk diimplementasikan dalam sistem *chatbot* layanan mahasiswa yang efektif.

4.5 Hasil *Deployment Chabot*

Model *chatbot* yang telah dibuat, dilakukan *deployment* dalam bentuk aplikasi web menggunakan Streamlit.

4.5.1 Halaman Awal *Chatbot*

Halaman Aplikasi *web* dirancang dengan satu halaman utama yang sederhana, yang diberi judul "BoTi" yang merupakan singkatan dari Bot FTTI di bagian atas. Pada halaman utama, terdapat sebuah bidang *input* yang memungkinkan pengguna untuk memasukkan pertanyaan mereka. Di sebelah kanan bidang *input*, terdapat tombol "Send" yang berfungsi untuk mengirimkan

pertanyaan tersebut ke *chatbot*. Halaman awal *chatbot* dapat dilihat pada Gambar 4.23.

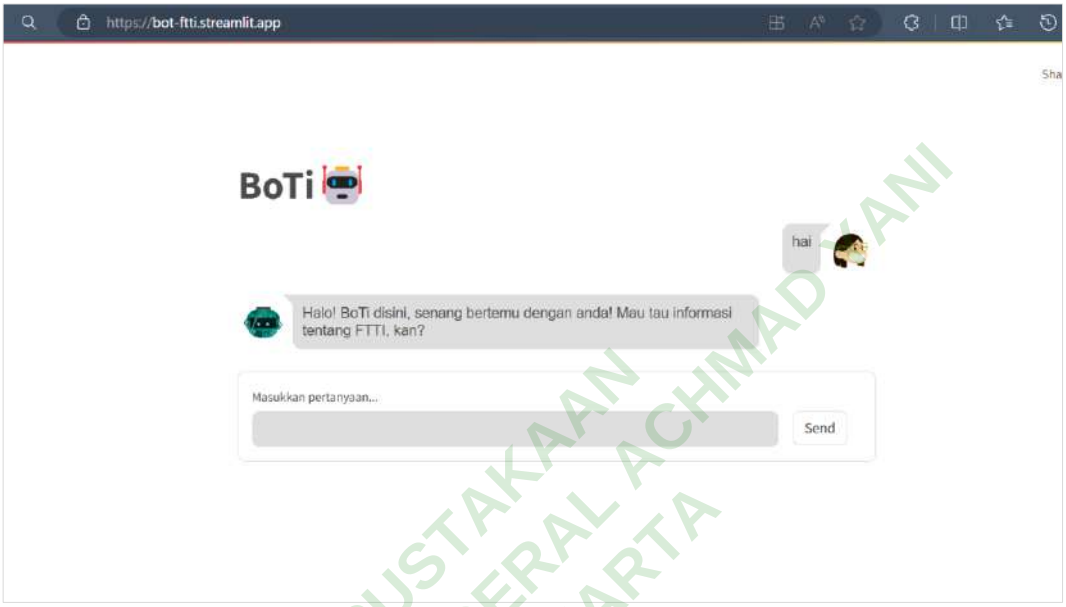
Gambar 4.23. Halaman Awal

4.5.2 Implementasi *Chatbot*

Pada Gambar 4.24 menunjukkan implementasi aplikasi yang diawali dengan menyapa sistem *chatbot* dengan memasukkan sapaan sederhana "hai" ke dalam bidang *input*. *Chatbot* kemudian merespons sapaan tersebut, dan hasil respons ini dapat dilihat pada Gambar 4.25.

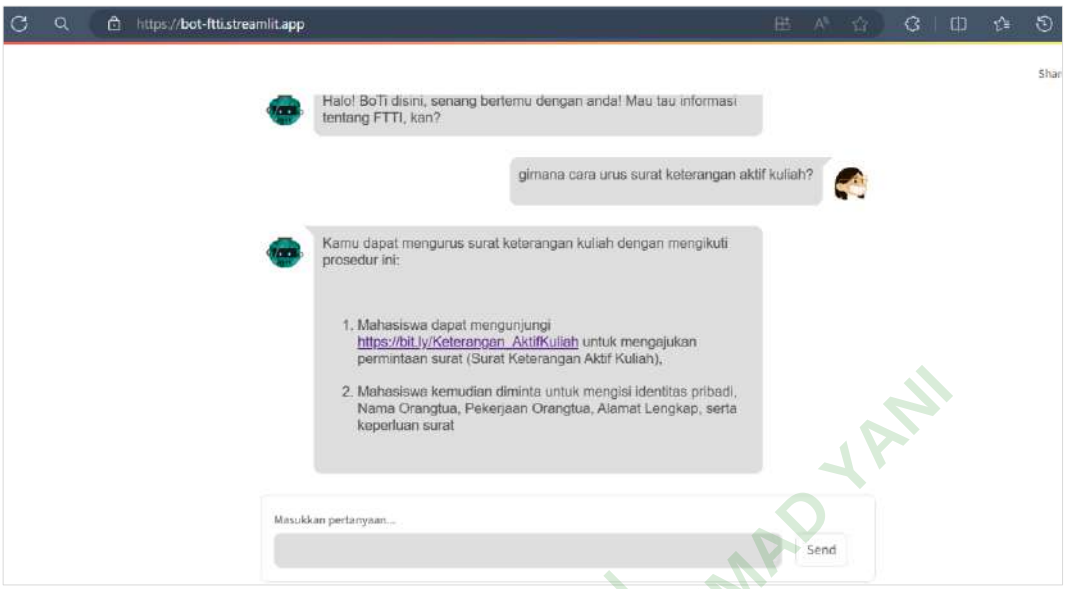
Gambar 4.24. Implementasi *Chatbot* ke-1

Pada Gambar 4.25 menunjukkan bahwa *chatbot* mampu memberikan respons yang baik dan akurat sesuai dengan sapaan "hai" yang diberikan. Selanjutnya, untuk menguji lebih lanjut mencoba memasukkan pertanyaan seputar layanan informasi lainnya.



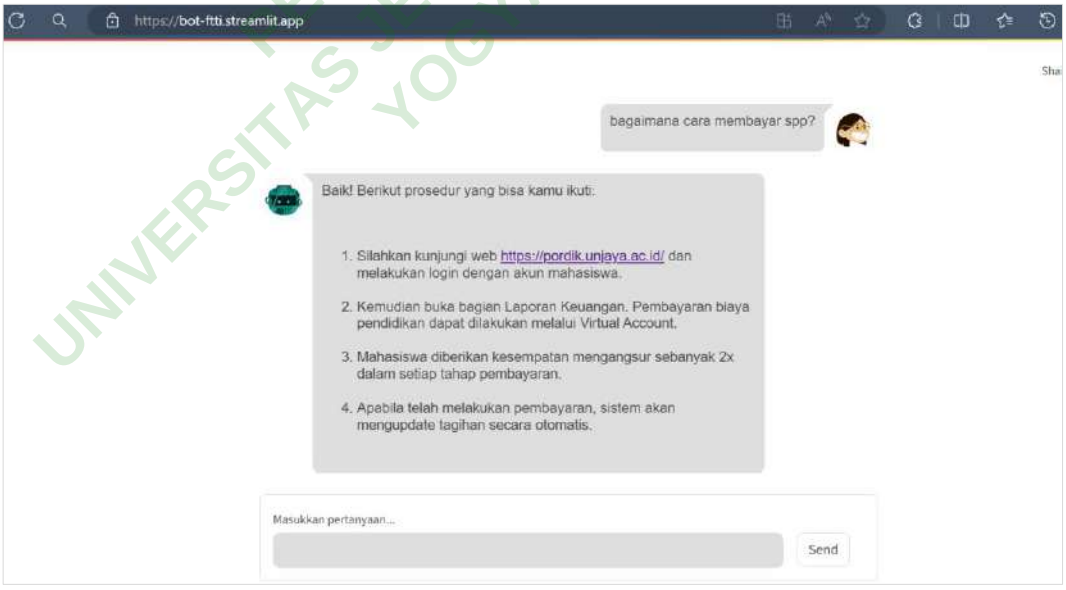
Gambar 4.25. Respon Implementasi *Chatbot* ke-1

Pada Gambar 4.26 menampilkan hasil dari pertanyaan bagaimana cara mengurus surat keterangan aktif yang diberikan. Dari respons yang diberikan oleh *chatbot*, menandakan bahwa *chatbot* bekerja dengan baik dan memberikan jawaban yang akurat. Berikutnya mencoba memberikan pertanyaan lagi seputar cara membayar SPP yang dapat dilihat pada Gambar 4.27.



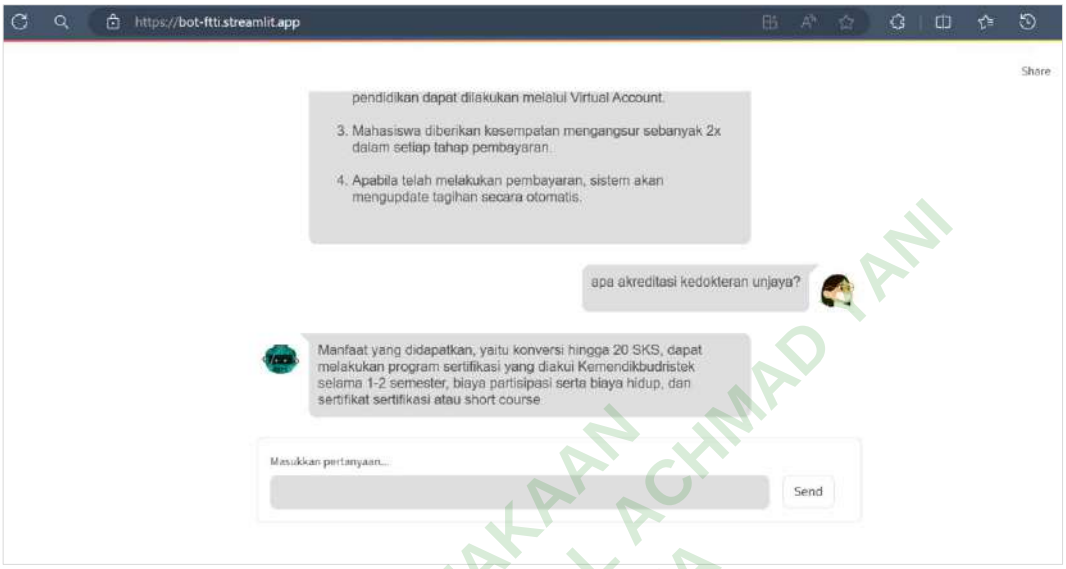
Gambar 4.26. Respon Implementasi *Chatbot* ke-2

Pada Gambar 4.27 menampilkan hasil dari pertanyaan bagaimana cara membayar SPP yang diberikan. Dari respons yang diberikan oleh *chatbot*, menandakan bahwa *chatbot* bekerja dengan baik dan memberikan jawaban yang akurat.



Gambar 4.27. Respon Implementasi *Chatbot* ke-3

Selanjutnya, mencoba memberikan pertanyaan kepada sistem *chatbot* dengan topik yang berada di luar lingkup FTTI, seperti yang ditunjukkan pada Gambar 4.28.



Gambar 4.28. Respon Implementasi *Chatbot* ke-4

Pada Gambar 4.28, terlihat *chatbot* memberikan respons berupa jawaban namun jawaban yang diberikan tidak sesuai dengan pertanyaan. Hal ini disebabkan oleh pertanyaan yang diberikan pada sistem *chatbot* tidak terkait dengan ruang lingkup FTTI.

4.6 Hasil Pengujian Sistem

Pada tahap pengujian sistem *chatbot* layanan informasi tugas akhir, dilakukan pengujian *usability* menggunakan metode skala likert dengan menyiapkan pertanyaan-pertanyaan yang akan dijadikan acuan untuk menilai *chatbot* oleh pengguna. Pengujian ini dilakukan oleh 16 responden mahasiswa Fakultas Teknik dan Teknologi Informasi. Sebelum responden memberikan nilai terhadap sistem, responden terlebih dahulu mencoba sistem *chatbot* tersebut. Agar penilaian terhadap sistem dapat *fair*, maka responden harus melakukan percobaan sistem dengan tahapan seperti diperlihatkan pada Tabel 4.2.

Tabel 4.2. Tahap Pengujian Responden

Tahapan	Kegiatan
1	Kunjungi tautan untuk mencoba *chatbot* BoTi (bot-ftti.streamlit.app)
2	Mengajukan pertanyaan seputar FTTI untuk melihat bagaimana *chatbot* merespons.

Kemudian responden diberikan pertanyaan dari kuesioner yang tujuannya untuk mengukur *usability* sistem *chatbot.* Pertanyaan tersebut berisi 14 pertanyaan dan nilai yang diberikan oleh responden yakni dengan skala 1 sampai 5 sesuai dengan Tabel 2.3. Pertanyaan – pertanyaan dapat dilihat pada Tabel 4.3.

Tabel 4.3 Pertanyaan Kuesioner

No	Pertanyaan	Skala				
		1	2	3	4	5
Aspek Fungsi						
1.	Seberapa mudah kamu menemukan informasi yang kamu cari menggunakan chatbot ini?					
2.	Apakah chatbot mampu memahami pertanyaan kamu dengan baik?					
3.	Seberapa cepat chatbot merespons pertanyaan kamu?					
4.	Apakah chatbot mampu memberikan informasi yang akurat?					
5.	Seberapa efektif chatbot dalam membantu kamu menyelesaikan masalah atau menjawab pertanyaan kamu?					
Aspek Tampilan						
6.	Bagaimana kamu menilai tampilan antarmuka chatbot ini?					
7.	Apakah tata letak dan desain antarmuka chatbot mudah digunakan?					
8.	Apakah teks dan elemen visual lainnya jelas dan mudah dibaca?					
9.	Seberapa puas kamu dengan pengalaman pengguna secara keseluruhan dalam berinteraksi dengan antarmuka chatbot?					

Aspek Hasil						
10.	Apakah hasil yang diberikan oleh chatbot memenuhi kebutuhan informasi kamu?					
11.	Seberapa relevan informasi yang diberikan oleh chatbot dengan pertanyaan yang kamu ajukan?					
12.	Seberapa puas kamu dengan kualitas jawaban yang diberikan oleh chatbot?					
13.	Apakah chatbot memberikan saran atau solusi yang berguna untuk pertanyaan atau masalah kamu?					
14.	Seberapa besar kemungkinan kamu akan menggunakan chatbot ini lagi di masa depan?					

Berikut merupakan skor asli hasil hitung dengan skala likert terhadap sistem chatbot. Rekapitulasi skor dapat dilihat pada Tabel 4.4.

Tabel 4.4 Rekapitulasi Keseluruhan

R	**Skor Asli**													
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
1.	3	3	4	3	2	4	3	4	4	3	3	3	2	3
2.	3	4	4	3	3	3	3	4	4	3	3	3	4	3
3.	4	4	5	4	4	3	4	4	4	4	4	3	4	4
4.	4	4	4	4	4	4	4	4	4	3	4	4	4	4
5.	4	5	4	5	4	3	3	3	4	5	5	4	4	4
6.	5	4	5	5	4	4	4	5	5	5	4	5	5	5
7.	4	4	4	4	4	4	5	5	4	5	5	4	4	4
8.	4	4	5	4	4	3	4	4	4	4	4	4	4	4
9.	5	5	5	5	5	5	4	4	4	4	4	5	5	5
10.	3	4	3	3	3	3	3	4	4	3	4	3	3	3
11.	4	4	4	4	4	4	4	4	4	4	4	4	4	4
12.	2	2	4	2	2	4	4	4	4	2	3	2	2	4
13.	4	4	4	4	4	4	4	4	4	4	4	4	4	4
14.	5	5	5	5	5	4	4	4	4	4	4	4	4	4
15.	4	5	3	3	4	4	5	5	4	4	3	4	4	4

16.	3	3	4	3	3	3	4	4	4	3	3	3	3	3
Rata-rata	3,81	4,00	4,19	3,81	3,69	3,69	3,88	4,13	4,06	3,75	3,81	3,69	3,75	3,88
Skor Rata-rata (Hasil Akhir)								**3,87**						

Keterangan:

R = Responden

P = Pertanyaan

Berdasarkan Tabel 4.4, didapatkan nilai rata-rata keseluruhan yaitu 3,87. Mengacu pada karakteristik poin penilaian yang tertera di Tabel 2.4 (Parwati et al, 2024), maka dapat disimpulkan bahwa sistem *chatbot* layanan informasi mahasiswa dikategorikan baik.