LAPORAN TUGAS AKHIR
KECERDASAN BUATAN
PENERAPAN NLP DALAM ASISTEN VIRTUAL MANAJEMEN PENJADWALAN KEGIATAN “MANAJEROBOT”

Disusun oleh
Kelompok 5

Dzikri Maulana (2210631170117)
Malik Syafi’i (2210631170029)

PROGRAM STUDI INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS SINGAPERBANGSA KARAWANG
2024

DAFTAR ISI

BAB I

PENDAHULUAN

A. LATAR BELAKANG

Penerapan teknologi Natural Language Processing (NLP) dalam berbagai bidang telah membawa revolusi signifikan, khususnya dalam pengembangan asisten digital yang cerdas dan adaptif. Salah satu aplikasi penting dari NLP adalah dalam manajemen penjadwalan kegiatan, di mana efisiensi dan akurasi sangat dibutuhkan.

Makalah ini berjudul "Penerapan NLP dalam Asisten Manajemen Penjadwalan Kegiatan “ManajeRobot" bertujuan untuk mengkaji bagaimana NLP dapat digunakan untuk mengembangkan asisten virtual yang mampu mengelola jadwal dengan efektif, membantu pengguna dalam mengatur waktu, dan memberikan rekomendasi yang tepat berdasarkan preferensi dan pola kegiatan pengguna. Melalui studi ini, diharapkan dapat diperoleh wawasan mendalam mengenai implementasi teknologi NLP dalam asisten manajemen penjadwalan, serta dampaknya terhadap produktivitas dan efisiensi kerja.

ManajeRobot menggunakan NLP untuk memahami bahasa alami pengguna, menerima permintaan penjadwalan, dan menghasilkan tanggapan yang informatif dan bermanfaat. Asisten ini dirancang untuk membantu individu dan organisasi dalam berbagai aspek manajemen kegiatan.

B. TUJUAN DAN MANFAAT

1). TUJUAN

1. Memudahkan Pengelolaan Jadwal:

- Pengguna dapat dengan mudah membuat, mengubah, dan menghapus acara dalam kalender mereka menggunakan perintah bahasa alami.
- Manajerobot dapat membantu menjadwalkan rapat dan pertemuan secara optimal, mempertimbangkan ketersediaan semua pihak dan menemukan waktu yang tepat untuk semua orang.
- Sistem ini dapat secara otomatis mengirimkan pengingat dan pemberitahuan acara, memastikan pengguna tidak melewatkan janji penting dan selalu tepat waktu.

2. Meningkatkan Efisiensi dan Produktivitas:

- Dengan sistem penjadwalan yang efisien, pengguna dapat fokus pada pekerjaan dan kegiatan utama mereka tanpa terhambat oleh kerepotan mengatur waktu.
- Manajerobot membantu meminimalisir waktu yang terbuang sia-sia karena penjadwalan yang tidak efektif, sehingga meningkatkan produktivitas dan kinerja.
- Pengguna dapat lebih fokus pada detail dan kualitas pekerjaan karena sistem telah membantu mereka mengelola waktunya secara efektif.

2). MANFAAT

1. Peningkatan Efisiensi dan Efektivitas Penjadwalan:

- Pengguna dapat membuat, mengubah, dan menghapus acara dengan mudah dan cepat menggunakan perintah bahasa alami, menghemat waktu dan tenaga.
- Manajerobot dapat membantu menjadwalkan rapat dan pertemuan secara optimal, mempertimbangkan ketersediaan semua pihak dan menemukan waktu yang tepat untuk semua orang.
- Sistem ini dapat secara otomatis mengirimkan pengingat dan pemberitahuan acara, memastikan pengguna tidak melewatkan janji penting dan selalu tepat waktu.

2. Peningkatan Produktivitas dan Kinerja:

- Dengan sistem penjadwalan yang efisien, pengguna dapat fokus pada pekerjaan dan kegiatan utama mereka tanpa terhambat oleh kerepotan mengatur waktu.
- Manajerobot membantu meminimalisir waktu yang terbuang sia-sia karena penjadwalan yang tidak efektif, sehingga meningkatkan produktivitas dan kinerja.
- Pengguna dapat lebih fokus pada detail dan kualitas pekerjaan karena sistem telah membantu mereka mengelola waktunya secara efektif.

3. Peningkatan Komunikasi dan Kolaborasi:

- Manajerobot dapat membantu menjadwalkan rapat dan pertemuan yang efektif, sehingga meningkatkan komunikasi dan kolaborasi antar anggota tim.
- Sistem ini dapat membantu menjembatani perbedaan waktu dan zona waktu, sehingga memudahkan komunikasi dan kolaborasi global.
- Pengguna dapat dengan mudah berbagi informasi dan dokumen terkait acara dengan anggota tim lain melalui Manajerobot.

4. Peningkatan Keteraturan dan Organisasi:

- Sistem penjadwalan yang terorganisir dengan Manajerobot membantu pengguna mengatur waktu dan kegiatan mereka dengan lebih rapi dan terstruktur.
- Hal ini dapat membantu mengurangi stres dan kecemasan yang terkait dengan penjadwalan yang tidak teratur.
- Pengguna dapat lebih fokus pada prioritas utama mereka dan mencapai tujuan dengan lebih efektif.

5. Peningkatan Kualitas Hidup:

- Manajerobot dapat membantu pengguna menyeimbangkan pekerjaan dan kehidupan pribadi dengan menjadwalkan waktu untuk relaksasi dan hiburan.
- Sistem ini dapat membantu pengguna mencapai tujuan hidup mereka dengan lebih mudah dengan menyediakan alat untuk mengelola waktu secara efektif.
- Pengguna dapat menikmati hidup yang lebih seimbang dan bahagia dengan bantuan Manajerobot.

BAB II

KERANGKA TEORITIS

A. PEMROSESAN BAHASA ALAMI (NLP)

Pemrosesan bahasa alami (Natural Language Processing/NLP) adalah cabang kecerdasan buatan yang memungkinkan komputer untuk memahami, menafsirkan, dan memanipulasi bahasa manusia. NLP melibatkan analisis teks dan bahasa, serta pemahaman konteks, semantik, sintaksis, dan tata bahasa. Dengan menggunakan teknik seperti mesin pencari, pengenalan suara, dan pemrosesan bahasa alami berbasis aturan, NLP membantu pengguna dalam menghasilkan dan memahami bahasa manusia dari teks, audio, atau video. NLP menjadi sangat penting dalam pengembangan teknologi digital dan AI karena meningkatkan kemampuan mesin untuk berinteraksi dengan manusia, memproses dan menganalisis teks, serta mengembangkan aplikasi yang responsif terhadap permintaan bahasa alami.

NLP telah berkembang pesat karena meningkatnya minat dalam komunikasi manusia-ke-mesin, ketersediaan big data, komputasi canggih, dan algoritma yang disempurnakan. Manusia dapat berbicara dan menulis dalam bahasa seperti Indonesia dan Inggris, tetapi bahasa alami komputer, yang dikenal sebagai kode atau bahasa mesin, tidak dapat dipahami oleh kebanyakan orang. Di tingkat perangkat yang paling dasar, komunikasi terjadi melalui jutaan kombinasi nol dan satu yang menghasilkan tindakan logis. Saat ini, kita bisa mengatakan, "Alexa, saya suka lagu ini," dan perangkat pemutar musik akan mengecilkan volume dan menjawab, "Oke. Peringkat disimpan," dengan suara mirip manusia. Perangkat tersebut kemudian menyesuaikan algoritmanya untuk memainkan lagu itu saat Anda mendengarkan stasiun musik tersebut di lain waktu.

CARA KERJA NLP

NLP menggabungkan model linguistik komputasional, machine learning, dan deep learning untuk memproses bahasa manusia. Berikut adalah beberapa cara kerja NLP

a. Tokenization

   Proses memecah teks menjadi bagian-bagian terkecil yang disebut token. Hal ini dilakukan untuk memudahkan proses analisis dan pemrosesan data teks.

b. NER (Named Entity Recognition): Proses mengidentifikasi entitas seperti orang, tempat, atau objek dalam teks. NER menggunakan teknik seperti pos-tagging dan pattern recognition untuk mengenali entitas-entitas tersebut.

c. Sentiment Analysis: Proses menganalisis sentimen atau perasaan yang terkandung dalam teks. Hal ini berguna untuk mengukur opini atau pandangan orang terhadap suatu topik tertentu.

d. Machine Translation: Proses menerjemahkan teks dari satu bahasa ke bahasa lainnya. Teknik ini menggunakan algoritma yang mengenal pola-pola bahasa untuk menghasilkan terjemahan yang disesuaikan dengan konteks dan aturan bahasa.
e. Text Summarization: Proses membuat ringkasan dari teks yang panjang. Teknik ini menggunakan algoritma yang dapat memilih informasi penting dari teks dan menghasilkan versi yang lebih singkat.

LANGKAH-LANGKAH IMPLEMENTASI NLP

Secara umum, langkah-langkah implementasi Natural Language Processing (NLP) meliputi:

1. Pengumpulan Data

Langkah pertama adalah mengumpulkan data teks yang akan diolah. Data dapat berasal dari berbagai sumber seperti media sosial, perusahaan, publikasi ilmiah, atau sumber lainnya.

2. Pra-pemrosesan Data

Setelah data dikumpulkan, langkah selanjutnya adalah melakukan pra-pemrosesan data, yang meliputi:

- Penghapusan Karakter Khusus, Stopword, Normalisasi Teks, dan Tokenisasi: Memecah sebuah kalimat menjadi unit kata atau frasa individual.

- Stemming dan Lemmatisasi: Menyederhanakan kata ke dalam bentuk akarnya. Misalnya, proses ini mengubah "starting" menjadi "start".

- Penghapusan Stopword: Menghapus kata yang tidak menambahkan makna signifikan ke sebuah kalimat, seperti "for" dan "with".

3. Pembuatan Korpus

Korpus adalah kumpulan teks yang memiliki karakteristik dan sifat yang sama. Sebagai contoh, korpus dapat dibuat berdasarkan topik tertentu atau bahasa yang digunakan.

4. Pengolahan Bahasa Alami

Pada tahap ini, data yang telah diproses akan diolah dalam bentuk model atau algoritma untuk memahami makna dalam teks. Beberapa contoh pengolahan bahasa alami termasuk pengelompokan kategori, analisis sentimen, dan pemodelan topik.

5. Evaluasi Model NLP

Setelah model NLP dibangun, langkah selanjutnya adalah mengevaluasi performa model. Hal ini dapat dilakukan dengan menggunakan metode evaluasi seperti precision, recall, dan F1 score. Evaluasi bertujuan untuk meningkatkan kualitas model NLP yang dibuat.

6. Deploy Model NLP

Setelah evaluasi model dilakukan, model NLP dapat di-deploy di berbagai platform, antara lain untuk meningkatkan proses bisnis, memahami keterlibatan pelanggan dan pasar, memonitor media sosial, dan lain-lain.

BAB III

HASIL DAN PEMBAHASAN

A. PENGEMBANGAN MODEL

1). Persiapan dataset

Korpus dibuat dengan mengumpulkan sejumlah besar teks yang relevan dengan domain yang sedang diteliti. Kumpulan teks tersebut dihimpun dalam file JSON dengan struktur yang sudah ditentukan, yaitu: tag, patterns, responses, dan context. Berikut ini adalah contoh kumpulan teks yang berhasil dihimpun dalam format JSON

```
    {
      "tag": "evaluasi kegiatan",
      "patterns": [
        "Bagaimana cara mengevaluasi kegiatan saya?",
        "Apa yang harus saya periksa setelah menyelesaikan tugas?",
        "Bagaimana cara mengevaluasi hasil kerja saya?",
        "Apa cara terbaik untuk mengevaluasi kegiatan?"
      ],
      "responses": [
        "Evaluasi kegiatan adalah langkah penting. Periksa apakah tujuan Anda tercapai dan identifikasi area untuk perbaikan.",
        "Setelah menyelesaikan tugas, luangkan waktu untuk meninjau apa yang berjalan dengan baik dan apa yang bisa ditingkatkan.",
        "Untuk mengevaluasi hasil kerja, bandingkan hasil aktual dengan tujuan yang telah ditetapkan dan analisis perbedaannya.",
        "Evaluasi kegiatan secara berkala membantu memastikan Anda tetap berada di jalur yang benar dan mencapai tujuan Anda."
      ]
    },
```

Dari data set ini akan dibuat model yang mampu mempelajari pola antara patterns dan tag, sehingga ketika pengguna memasukkan pernyataan, tag akan diidentifikasi untuk menghasilkan respon yang sebagai respon dari chatbox.

2). Pra-pemrosesan Data

Pra-pemrosesan data adalah langkah penting untuk memastikan bahwa data yang digunakan dalam pelatihan model adalah bersih dan dalam format yang sesuai. Langkah-langkah pra-pemrosesan data meliputi:

a. Tokenisasi

```
import nltk
nltk.download('punkt')
```

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        documents.append((w, intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Kalimat akan dipecah menjadi unit kata individual, tujuannya adalah memisahkan setiap kata dalam kalimat sehingga dapat diproses secara individual oleh algoritma pemrosesan teks. Ini membantu dalam analisis dan pemahaman struktur kalimat, memudahkan identifikasi pola, dan memungkinkan langkah-langkah pemrosesan selanjutnya seperti stemming dan lemmatization.

b. Penghapusan Stop Words

```
ignore = ['?', '!', '.', ',', ';', ':', '-', '_', '(', ')', '[', ']', '{', '}', '"', "'",
          'dan', 'atau', 'tapi', 'jika', 'sementara', 'dengan', 'tanpa', 'pada', 'di', 'ke',
          'dari', 'oleh', 'untuk', 'dalam', 'yang', 'sebuah', 'adalah', 'adalah', 'ini', 'itu',
          'saya', 'kami', 'kita', 'dia', 'mereka', 'anda', 'aku', 'kamu', 'yang', 'telah',
          'telah', 'akan', 'sedang', 'telah', 'bisa', 'dapat', 'harus']
```

Menghapus kata-kata umum yang tidak memberikan informasi penting, seperti "dan", "di", "yang", dll., dengan tujuan untuk mengurangi kebisingan dalam data teks dan fokus pada kata-kata yang lebih bermakna. Hal ini membantu meningkatkan efisiensi dan akurasi algoritma pemrosesan teks dengan menghilangkan kata-kata yang sering muncul tetapi tidak memberikan nilai signifikan dalam analisis konteks atau makna.

c. Stemming dan Lemmatization

```
# Melakukan stemming dan normalisasi data
words = [stemmer.stem(w.lower()) for w in words if w not in ignore]
words = sorted(list(set(words)))

# Menghapus class duplikat dengan 'set'
classes = sorted(list(set(classes)))
```

```
39 classes ['Identitas', 'Selamat Tinggal', 'aktivitas', 'apa kabar', 'apresiasi', 'balancing kegiatan', 'balas sapa', 'cuaca', 'delegasi tugas', 'evaluasi kegiatan', 'haha', 'hapus_jadwal', 'istirahat', 'konfirmasi', 'lelucon', 'lihat_jadwal', 'motivasi', 'opsi', 'pembicaraan menyenangkan', 'pengatur waktu', 'pengaturan tujuan', 'pengingat kegiatan', 'perubahan_jadwal', 'prioritas', 'produktivitas', 'programmer', 'proyek', 'rencana bulanan', 'rencana harian', 'rencana mingguan', 'salam', 'seru', 'tambah_jadwal', 'tanggal dan waktu', 'teka-teki', 'terima kasih', 'tidak', 'umur', 'update_jadwal']
219 unique stemmed words ['acara', 'ada', 'aktivitas', 'alarm', 'anda', 'antara', 'apa', 'apakah', 'asah', 'atas', 'atur', 'ayo', 'bagaimana', 'bagi', 'bagus', 'baik', 'banget', 'bantu', 'banyak', 'baru', 'batal', 'batas', 'belum', 'berapa', 'beri', 'besar', 'biasa', 'bicara', 'bikin', 'bisa', 'buang', 'buat', 'bukan', 'bulan', 'butuh', 'bye', 'capai', 'cara', 'cek', 'cerita', 'coba', 'cuaca', 'dadah', 'dadak', 'daftar', 'dapat', 'delegasi', 'dengan', 'dengar', 'derajat', 'desain', 'dukung', 'edit', 'efektif', 'efisien', 'evaluasi', 'fantastis', 'fleksibel', 'giat', 'gimana', 'guna', 'haha', 'hai', 'halo', 'hapus', 'harga', 'hari', 'hasil', 'hebat', 'hei', 'hi', 'hidup', 'hindar', 'hola', 'ikan', 'ikut', 'imbang', 'ingat', 'ingin', 'istirahat', 'itu', 'jadi', 'jadwal', 'jaga', 'jam', 'jangan', 'jangka', 'jumpa', 'kabar', 'kali', 'kamu', 'kapan', 'kasih', 'kata', 'kelola', 'kembang', 'keren', 'kerja', 'ketawa', 'ketemu', 'kocak', 'kondisi', 'kurang', 'lagi', 'lain', 'laku', 'lama', 'langkah', 'layan', 'lebih', 'lelah', 'lelucon', 'lihat', 'lmao', 'lol', 'luar', 'lucu', 'makasih', 'mampu', 'manajemen', 'minggu', 'motivasi', 'mulai', 'nama', 'namaste', 'nanti', 'ngakak', 'nggak', 'ngobrol', 'nikmat', 'obrol', 'oke', 'oleh', 'orang', 'otak', 'panas', 'panjang', 'pecah', 'pemberitahuan', 'penasaran', 'pergi', 'periksa', 'perlu', 'pernah', 'pertama', 'pribadi', 'prioritas', 'produktif', 'produktivitas', 'program', 'proyek', 'pukul', 'punya', 'realistis', 'rencana', 'rofl', 'rumit', 'saat', 'saja', 'sampai', 'sana', 'sangat', 'saran', 'saya', 'sedia', 'sehat', 'sekarang', 'selamat', 'selesai', 'semangat', 'semua', 'senang', 'seru', 'sesuai', 'sesuatu', 'set', 'siang', 'siapa', 'sudah', 'suhu', 'sup', 'tahu', 'tambah', 'tampil', 'tanggal', 'tantang', 'tawar', 'teka-teki', 'telah', 'tentang', 'tentu', 'tepat', 'terima', 'tertawa', 'terus', 'tetap', 'tidak', 'tim', 'timbang', 'timer', 'tinggal', 'tingkat', 'tips', 'tolong', 'tugas', 'tuju', 'tunjuk', 'ubah', 'uji', 'ulang', 'umur', 'untuk', 'update', 'usia', 'utama', 'waktu', 'wazzup', 'ya', 'yo']
```

Mengubah kata-kata menjadi bentuk dasar atau akar katanya untuk mengurangi variasi kata. Stemming memotong akhiran kata untuk mencapai bentuk dasar, meskipun hasilnya mungkin bukan kata yang benar secara linguistik. Lemmatization, di sisi lain, mengubah kata menjadi bentuk dasarnya yang benar secara linguistik berdasarkan kamus, mempertimbangkan konteks dan bagian dari kata tersebut. Tujuan dari kedua teknik ini adalah untuk menyatukan variasi kata yang berbeda menjadi bentuk yang seragam, sehingga meningkatkan akurasi analisis teks dan pemahaman konteks oleh algoritma pemrosesan bahasa alami.

Normalisasi kata juga dilakukan dengan mengubah kata menjadi huruf kecil untuk mencegah kesalahpahaman model akibat kapitalisasi kata. Kata juga akan diignore jika terdapat pada list ignore agar penghapusan stopword dapat dilakukan.

Stemming dilakukan menggunakan library PySastrawi yang secara khusus menggunakan bahasa Indonesia sebagai bahasa utamanya.

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
factory = StemmerFactory()
stemmer = factory.create_stemmer()
```

Contoh stemming dengan beberapa sample kata:

```
[16]: sample_word = ['membunuh','menjadwalkan','mengatur']
      contoh = [stemmer.stem(sample.lower()) for sample in sample_word]

      contoh

[16]: ['bunuh', 'jadwal', 'atur']
```

3). Training Model

a. Pembuatan data training

```
import random
import numpy as np

training = []
output_empty = [0] * len(classes)

for doc in documents:
    pattern_words = [stemmer.stem(word.lower()) for word in doc[0]]
    bag = [1 if w in pattern_words else 0 for w in words]

    output_row = output_empty[:]
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])

random.shuffle(training)
training = np.array(training, dtype=object)

train_x = np.array([item[0] for item in training])
train_y = np.array([item[1] for item in training])
```

Data dari korpus yang telah diproses dibagi menjadi dua set utama: data training dan data testing. Data training digunakan untuk melatih model, sementara data testing digunakan untuk mengevaluasi kinerja model.

b. Pelatihan model

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import random
import json
import pickle
```

Pada langkah ini, model neural network dibangun menggunakan Sequential model dari Keras. Model ini terdiri dari beberapa lapisan, termasuk lapisan input, lapisan tersembunyi, dan lapisan output. Setiap lapisan didefinisikan dengan jumlah unit dan fungsi aktivasi yang sesuai, terdiri dari ‘relu’ dan ‘softmax’.

```
[7]: # Reset the default TensorFlow graph
     tf.compat.v1.reset_default_graph

     # Define the model
     model = keras.Sequential([
         layers.InputLayer(input_shape=(len(train_x[0]),), name='input_layer'),
         layers.Dense(10, activation='relu', name='hidden_layer1'),
         layers.Dense(10, activation='relu', name='hidden_layer2'),
         layers.Dense(len(train_y[0]), activation='softmax', name='output_layer')
     ])
```

Setelah model didefinisikan, langkah selanjutnya adalah mengompilasi model dengan menentukan optimizer, fungsi loss, dan metrik evaluasi yang akan digunakan selama pelatihan.

TensorBoard digunakan sebagai alat visualisasi yang kuat untuk memantau kinerja pelatihan dan memahami perilaku model. Proses pelatihan model dimulai dengan memanggil fungsi fit pada model. Selama proses pelatihan, model akan mempelajari pola dari data training untuk mengoptimalkan kinerjanya terhadap data yang diberikan.

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Set up TensorBoard callback
tensorboard_callback = keras.callbacks.TensorBoard(log_dir='./logs')

# Train the model
model.fit(train_x, train_y, epochs=1000, batch_size=8, callbacks=[tensorboard_callback], verbose=1)

# Save the model
model.save('model.h5')
```

Setelah pelatihan selesai, model dapat disimpan ke file untuk digunakan di masa depan tanpa perlu melatih ulang.

Penyimpanan data untuk model kemudian dilakukan untuk keperluan penggunaan model selanjutnya

```
# Save
try:
    with open("training_data", "wb") as file:
        pickle.dump({'words': words, 'classes': classes, 'train_x': train_x, 'train_y': train_y}, file)
except IOError as e:
    print(f"Error saving training data: {e}")

# Load
try:
    with open("training_data", "rb") as file:
        data = pickle.load(file)

    words = data['words']
    classes = data['classes']
    train_x = np.array(data['train_x'])
    train_y = np.array(data['train_y'])
except IOError as e:
    print(f"Error loading training data: {e}")

# Load model
try:
    model = keras.models.load_model('model.h5')
except IOError as e:
    print(f"Error loading model: {e}")
```

4). Penggunaan model

```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
    return sentence_words

def bow(sentence, words, show_details=False):
    sentence_words = clean_up_sentence(sentence)
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1
            if show_details:
                print(f"Found in bag: {w}")
    return bag

ERROR_THRESHOLD = 0.30

def classify(sentence):
    input_data = bow(sentence, words)
    results = model.predict(np.array([input_data]))[0]
    results = [[i, r] for i, r in enumerate(results) if r >= ERROR_THRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return [(classes[r[0]], r[1]) for r in results]

def response(sentence, userID='123', show_details=False):
    results = classify(sentence)
    if results:
        for intent in intents['intents']:
            if intent['tag'] == results[0][0]:
                return random.choice(intent['responses'])
    return "Maaf, saya tidak mengerti apa yang Anda katakan."
```

Kode ini merupakan bagian dari implementasi chatbot berbasis pembelajaran mesin yang menggunakan teknik stemming dan representasi Bag of Words (BoW) untuk memahami dan mengklasifikasikan input pengguna.

1. **Fungsi clean_up_sentence**:
    - Kode ini merupakan fungsi untuk membersihkan kalimat masukan. Fungsi clean_up_sentence menerima kalimat sebagai argumen, kemudian menggunakan nltk.word_tokenize untuk memecah kalimat tersebut menjadi kata-kata individual. Setiap kata kemudian diubah menjadi bentuk dasar (stemming) menggunakan pustaka stemming dan diubah menjadi huruf kecil. Fungsi ini mengembalikan daftar kata-kata yang telah di-tokenisasi dan di-stemming.
2. **Fungsi bow (Bag of Words)**:
    - Kode ini merupakan fungsi untuk mengubah kalimat menjadi representasi vektor menggunakan teknik Bag of Words (BoW). Fungsi bow menerima kalimat dan daftar kata-kata (korpus) sebagai argumen, kemudian membersihkan kalimat menggunakan fungsi clean_up_sentence.

Setelah itu, fungsi ini membuat vektor dengan panjang yang sama dengan jumlah kata dalam korpus, di mana setiap elemen diatur menjadi 1 jika kata yang bersangkutan ditemukan dalam kalimat, dan 0 jika tidak ditemukan. Fungsi ini juga memiliki opsi untuk menampilkan kata-kata yang ditemukan dalam kalimat jika parameter show_details diatur ke True.

3. **Konstanta ERROR_THRESHOLD**:
    - Kode ini mendefinisikan ambang batas kesalahan yang digunakan untuk memfilter hasil prediksi. Nilai ERROR_THRESHOLD ditetapkan sebesar 0.30, yang berarti hanya hasil prediksi dengan probabilitas di atas 0.30 yang akan dipertimbangkan sebagai hasil yang valid.
4. **Fungsi classify**:
    - Kode ini merupakan fungsi untuk mengklasifikasikan kalimat masukan ke dalam salah satu kelas yang telah dilatih. Fungsi classify menggunakan fungsi bow untuk mengubah kalimat menjadi vektor input, kemudian menggunakan model yang telah dilatih untuk memprediksi kelas dari kalimat tersebut. Hasil prediksi yang memiliki probabilitas di atas ERROR_THRESHOLD akan diurutkan dan dikembalikan sebagai pasangan kelas dan nilai probabilitasnya.
5. **Fungsi response**:
    - Kode ini merupakan fungsi untuk menghasilkan respons berdasarkan kalimat masukan. Fungsi response memanggil fungsi classify untuk menentukan kelas dari kalimat masukan, kemudian mencari respons yang sesuai dari daftar intent berdasarkan kelas tersebut. Jika kelas yang sesuai ditemukan, fungsi ini akan mengembalikan salah satu respons yang telah ditentukan secara acak; jika tidak, akan mengembalikan pesan default "Maaf, saya tidak mengerti apa yang Anda katakan."

[11]:
```
def action(order):
    if order == 'time':
        print(" sekarang ",datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    if order == 'addschedule':
        add_event()
    if order == 'showschedule':
        print_schedule()
    if order == 'updateschedule':
        update_event()
    if order == 'deleteschedule' :
        delete_event()
```

[ ]:
```
print("0 to close")
while True:
    message = input("")
    if message == "0":
        break
    result = response(message)

    if result is not None and "~" in result:
        order = (result[1:])
        action(order)
    else:
        print(result)
```

Nantinya, respond dari model akan digunakan untuk mengambil keputusan yang aksinya dilakukan oleh kode program yang akan dijelaskan selanjutnya, aksi diambil apabila respond dari chatbox mengandung karakter ‘~’ sebagai karakter yang telah diset sebagai karakter penanda untuk memberi ciri pada respond perintah.

B. PENGEMBANGAN KODE PROGRAM AKSI

```python
def load_schedule():
    file_name="schedule.xlsx"
    try:
        # Cek file excel ada?
        schedule = pd.read_excel(file_name, parse_dates=['Date'])
    except FileNotFoundError:
        #buat df jika tidak ada
        schedule = pd.DataFrame(columns=['Date', 'Start Time', 'End Time', 'Event'])
    return schedule
```

Fungsi load_schedule() melakukan pengecekan file apakah ada atau tidak, jika tidak maka akan menambahkan file baru, kemudian mengembalikan hasil pemuatan.

Memuat Jadwal (load_schedule)

1. **MULAI**
2. tetapkan nama file jadwal:
    - nama_file = "schedule.xlsx"
3. **COBA BUKA FILE EXCEL:**
    - Gunakan pd.read_excel(nama_file, parse_dates=['Date']) untuk membaca file Excel.
    - Atur parse_dates=['Date'] untuk memformat kolom "Tanggal" menjadi format tanggal.
4. **PENANGANAN KESALAHAN (FileNotFoundError):**
    - **JIKA** file tidak ditemukan:
        - Buat DataFrame kosong bernama jadwal dengan kolom:
            - "Tanggal"
            - "Waktu Mulai"
            - "Waktu Selesai"
            - "Acara"
5. **KEMBALIKAN DataFrame jadwal.**
6. **SELESAI**

```python
def save_schedule(file_name, schedule):
    schedule.to_excel(file_name, index=False)
```

Fungsi save_schedule() melakukan penyimpanan data kedalam file excel dari kegiatan yang sudah dibuat,update,atau hapus.

Menyimpan Jadwal (save_schedule)

1. **MULAI**
2. **TULIS JADWAL KE FILE EXCEL:**
    - Gunakan schedule.to_excel(nama_file, index=False) untuk menulis DataFrame jadwal ke file nama_file.xlsx.
    - index=False untuk mengecualikan kolom indeks saat menyimpan.
3. **SELESAI.**

```python
def add_event():
    schedule = load_schedule()
    print("Enter new event details:")
    event_date = input("Date (YYYY-MM-DD): ")
    start_time = input("Start Time (HH:MM): ")
    end_time = input("End Time (HH:MM): ")
    event_name = input("Event Name: ")

    event_date = pd.to_datetime(event_date).date()

    # Conflict?
    for i, row in schedule.iterrows():
        row_date = row['Date'].date() if isinstance(row['Date'], pd.Timestamp) else row['Date']
        if row_date == event_date:
            if (start_time < row['End Time'] and start_time >= row['Start Time']) or \
               (end_time > row['Start Time'] and end_time <= row['End Time']) or \
               (start_time <= row['Start Time'] and end_time >= row['End Time']):
                print(find_nearest_available_time(event_date, start_time, end_time))
                return f"Acaramu yg ini konflik dengan acara : {row['Event']} dari {row['Start Time']} ke {row['End Time']}"
```

```python
    # Menambahkan event
    new_event = pd.DataFrame([[event_date, start_time, end_time, event_name]],
                             columns=['Date', 'Start Time', 'End Time', 'Event'])
    schedule = pd.concat([schedule, new_event], ignore_index=True)

    # SOrting schedule
    schedule['Date'] = pd.to_datetime(schedule['Date'])
    schedule = schedule.sort_values(by=['Date', 'Start Time']).reset_index(drop=True)
    save_schedule("schedule.xlsx", schedule)
    print("Berhasil menambahkan jadwal")
    print_schedule()
```

Fungsi add_event() memungkinkan pengguna untuk menambahkan acara baru ke dalam daftar acara yang sudah ada. Fungsi ini dirancang untuk menghindari konflik waktu dengan acara yang telah terjadwal sebelumnya.

Fungsi: tambah_acara()

1. **MULAI**
2. **MEMUAT JADWAL:**
    - Panggil fungsi load_schedule untuk memuat jadwal yang tersimpan.

3. **INPUT DATA ACARA BARU:**
    - Tampilkan pesan "Masukkan detail acara baru:"
    - Minta pengguna memasukkan:
        - Tanggal Acara (format YYYY-MM-DD)
        - Waktu Mulai (format HH:MM)
        - Waktu Selesai (format HH:MM)
        - Nama Acara
4. **KONVERSI TANGGAL ACARA:**
    - Ubah tanggal_acara menjadi objek date menggunakan pd.to_datetime(tanggal_acara).date().
5. **CEK KONFLIK ACARA:**
    - Iterasi setiap baris dalam jadwal menggunakan schedule.iterrows().
        - Dapatkan tanggal baris saat ini (tanggal_baris) dari kolom 'Date'.
            - Periksa tipe data 'Date'. Jika pd.Timestamp, konversi ke format date.
        - **JIKA** tanggal_baris SAMA DENGAN tanggal_acara:
            - Periksa konflik waktu: (sama seperti fungsi sebelumnya)
                - Jika ada konflik, panggil fungsi find_nearest_available_time untuk mencari waktu terdekat yang tersedia.
                - Cetak pesan kesalahan dan informasi acara yang konflik.
                - **KEMBALIKAN** pesan kesalahan.
6. **TAMBAH ACARA BARU (TIDAK ADA KONFLIK):**
    - Buat DataFrame baru acara_baru dengan 1 baris dan 4 kolom:
        - Kolom: 'Date', 'Start Time', 'End Time', 'Event'
        - Nilai: [tanggal_acara, waktu_mulai, waktu_selesai, nama_acara]
    - Gabungkan acara_baru ke dalam jadwal menggunakan pd.concat dengan ignore_index=True.
7. **URUTKAN JADWAL:**
    - Konversi kolom 'Date' dalam jadwal menjadi format datetime menggunakan pd.to_datetime.
    - Urutkan jadwal berdasarkan 'Date' dan 'Start Time' menggunakan schedule.sort_values dan atur ulang indeks dengan reset_index(drop=True).
8. **SIMPAN JADWAL:**
    - Panggil fungsi save_schedule untuk menyimpan jadwal yang diperbarui ke file "schedule.xlsx".
9. **TAMPILKAN KONFIRMASI:**
    - Cetak pesan "Berhasil menambah acara".
10. **TAMPILKAN JADWAL (Opsional):**
    - Panggil fungsi print_schedule untuk menampilkan jadwal yang diperbarui (opsional).

11. **SELESAI.**

```python
def print_schedule():
    schedule = load_schedule()

    # Add an index column
    schedule.reset_index(inplace=True)
    schedule.rename(columns={'index': 'No'}, inplace=True)

    # Print DataFrame in tabular format
    print(tabulate(schedule, headers='keys', tablefmt='grid', showindex=False))
```

Fungsi print_schedule() digunakan untuk menampilkan jadwal yang ada atau sudah dibuat sebelumnya dengan format tabular.

Fungsi: print_schedule()

1. **MULAI**
2. **MEMUAT JADWAL:**
    - Panggil fungsi load_schedule untuk memuat jadwal yang tersimpan.
3. **PERSIAPKAN PENAMPILAN JADWAL:**
    - Tambahkan kolom indeks sementara ke schedule menggunakan reset_index(inplace=True).
    - Ubah nama kolom indeks menjadi "No" menggunakan rename(columns={'index': 'No'}, inplace=True).
4. **TAMPILKAN JADWAL:**
    - Gunakan library tabulate (diasumsikan sudah terinstall) untuk mencetak schedule dalam format tabel.
        - Argumen:
            - schedule: DataFrame yang ingin ditampilkan.
            - headers='keys': Judul kolom menggunakan nama kolom asli.
            - tablefmt='grid': Format tabel menggunakan grid.
            - showindex=False: Sembunyikan indeks bawaan.
5. **SELESAI.**

```python
def update_event():
    schedule = load_schedule()
    print_schedule()
    print("Enter event details to update:")
    index = int(input("Select index [start from 0]: "))
    event_date = input("Date (YYYY-MM-DD): ")
    start_time = input("Start Time (HH:MM): ")
    end_time = input("End Time (HH:MM): ")
    event_name = input("Event Name: ")

    if index >= len(schedule) or index < 0:
        return False, "Invalid index"

    event_date = pd.to_datetime(event_date).date()  # Ubah string 'event_date' menjadi tipe datetime.date
    #start_time = datetime.datetime.strptime(start_time, '%H:%M').time()
    #end_time = datetime.datetime.strptime(end_time, '%H:%M').time()
```

```python
# Conflict?
    for i, row in schedule.iterrows():
        if i == index:
            continue
        row_date = row['Date'].date() if isinstance(row['Date'], pd.Timestamp) else row['Date']
        if row_date == event_date:
            if (start_time < row['End Time'] and start_time >= row['Start Time']) or \
               (end_time > row['Start Time'] and end_time <= row['End Time']) or \
               (start_time <= row['Start Time'] and end_time >= row['End Time']):
                print(find_nearest_available_time(event_date, start_time, end_time))
                return f"Acaramu yg ini konflik dengan acara : {row['Event']} dari {row['Start Time']} ke {row['End Time']}"


    # Update event
    schedule.at[index, 'Date'] = event_date
    schedule.at[index, 'Start Time'] = start_time
    schedule.at[index, 'End Time'] = end_time
    schedule.at[index, 'Event'] = event_name

    # Sort schedule by date and start time
    schedule['Date'] = pd.to_datetime(schedule['Date'])
    schedule = schedule.sort_values(by=['Date', 'Start Time']).reset_index(drop=True)
    save_schedule("schedule.xlsx", schedule)
    print("Update berhasil")
    print_schedule()
```

Fungsi update_schedule() memungkinkan pengguna untuk mengupdate jadwal, dengan memilih jadwal yang ingin diganti dengan indeks kemudian mengganti keterangan kegiatan yang ingin diubah setelah itu dilakukan pengecekan apakah bentrok lalu memfiksasi update kegiatan jika tidak terdapat bentrok.

Fungsi: update_acara(jadwal)

1. **MULAI**
2. **MEMUAT JADWAL:**
    - Panggil fungsi load_schedule untuk memuat jadwal yang tersimpan.
3. **TAMPILKAN JADWAL (Opsional):**
    - Panggil fungsi print_schedule untuk menampilkan jadwal yang tersimpan (opsional).
4. **INPUT DATA ACARA YANG DIUBAH:**
    - Tampilkan pesan "Masukkan detail acara yang ingin diubah:"

- Minta pengguna memasukkan:
  - Indeks acara (mulai dari 0)
  - Tanggal Acara (format YYYY-MM-DD)
  - Waktu Mulai (format HH:MM)
  - Waktu Selesai (format HH:MM)
  - Nama Acara

5. **VALIDASI INDEKS:**
   - **JIKA** indeks lebih besar atau sama dengan panjang jadwal ATAU indeks kurang dari 0:
     - **KEMBALIKAN** False dan pesan kesalahan "Indeks tidak valid".
6. **KONVERSI TANGGAL ACARA:**
   - Ubah tanggal_acara menjadi objek date menggunakan pd.to_datetime(tanggal_acara).date().
7. **CEK KONFLIK ACARA:**
   - Iterasi setiap baris dalam jadwal menggunakan schedule.iterrows().
     - **LEWATI** baris dengan indeks index (baris yang akan diubah).
     - Dapatkan tanggal baris saat ini (tanggal_baris) dari kolom 'Date'.
       - Periksa tipe data 'Date'. Jika pd.Timestamp, konversi ke format date.
     - **JIKA** tanggal_baris SAMA DENGAN tanggal_acara:
       - Periksa konflik waktu: (sama seperti fungsi sebelumnya)
         - Jika ada konflik, panggil fungsi find_nearest_available_time untuk mencari waktu terdekat yang tersedia.
         - Cetak pesan kesalahan dan informasi acara yang konflik.
         - **KEMBALIKAN** pesan kesalahan.
8. **PERBARUI ACARA:**
   - Perbarui kolom pada baris yang ditunjuk oleh indeks dalam jadwal:
     - schedule.at[index, 'Date'] = event_date
     - schedule.at[index, 'Start Time'] = start_time
     - schedule.at[index, 'End Time'] = end_time
     - schedule.at[index, 'Event'] = event_name
9. **URUTKAN JADWAL:**
   - Konversi kolom 'Date' dalam jadwal menjadi format datetime menggunakan pd.to_datetime.
   - Urutkan jadwal berdasarkan 'Date' dan 'Start Time' menggunakan schedule.sort_values dan atur ulang indeks dengan reset_index(drop=True).
10. **SIMPAN JADWAL:**
    - Panggil fungsi save_schedule untuk menyimpan jadwal yang diperbarui ke file "schedule.xlsx".

11. **TAMPILKAN KONFIRMASI:**
    - Cetak pesan "Update berhasil".
12. **TAMPILKAN JADWAL (Opsional):**
    - Panggil fungsi print_schedule untuk menampilkan jadwal yang diperbarui (opsional).
13. **SELESAI.**

```python
def delete_event():
    schedule = load_schedule()
    print(print_schedule())
    print("Enter event details to delete:")
    index = int(input("select index [start from 0] : "))
    if index >= len(schedule) or index < 0:
        return False, "Invalid index"
    konfirmasi = input("Anda yakin menghapus jadwal? (0/1) :")
    if konfirmasi == 0 :
        return 'gagal diproses'
    else:
        schedule = schedule.drop(index).reset_index(drop=True)
        save_schedule("schedule.xlsx", schedule)
        print ('proses hapus berhasil')
        print(print_schedule())
```

Fungsi delete_schedule() memungkinkan pengguna untuk menghapus jadwal atau kegiatan yang di inginkan.

Fungsi: delete_event()

1. **MULAI**
2. **MEMUAT JADWAL:**
    - Panggil fungsi load_schedule untuk memuat jadwal yang tersimpan.
3. **TAMPILKAN JADWAL (Opsional):**
    - Panggil fungsi print_schedule untuk menampilkan jadwal yang tersimpan (opsional).
4. **INPUT INDEKS ACARA:**
    - Tampilkan pesan "Masukkan detail acara yang ingin dihapus:"
    - Minta pengguna memasukkan indeks acara (mulai dari 0).
5. **VALIDASI INDEKS:**
    - **JIKA** indeks lebih besar atau sama dengan panjang jadwal ATAU indeks kurang dari 0:
        - **KEMBALIKAN** False dan pesan kesalahan "Indeks tidak valid".
6. **KONFIRMASI PENGHAPUSAN:**
    - Tanyakan konfirmasi penghapusan: "Anda yakin menghapus jadwal? (0/1) :"
    - **JIKA** konfirmasi adalah "0":
        - **KEMBALIKAN** pesan "Proses gagal dihapus".
7. **HAPUS ACARA:**
    - Hapus baris dengan indeks indeks dari schedule menggunakan schedule.drop(index).
    - Atur ulang indeks dengan reset_index(drop=True).
8. **SIMPAN JADWAL:**
    - Panggil fungsi save_schedule untuk menyimpan jadwal yang diperbarui ke file "schedule.xlsx".

9. **TAMPILKAN KONFIRMASI:**
    - Cetak pesan "Proses hapus berhasil".
10. **TAMPILKAN JADWAL (Opsional):**
    - Panggil fungsi print_schedule untuk menampilkan jadwal yang diperbarui (opsional).
11. **SELESAI.**

```python
def find_nearest_available_time(event_date, start_time, end_time):
    schedule = load_schedule()
    event_date = pd.to_datetime(event_date).date()
    duration_minutes = (datetime.datetime.strptime(end_time, '%H:%M') - datetime.datetime.strptime(start_time, '%H:%M')).total_seconds() / 60
    start_time = datetime.datetime.strptime(start_time, '%H:%M').time()
    end_time = datetime.datetime.strptime(end_time, '%H:%M').time()

    # Convert current event to datetime format
    current_event_start = datetime.datetime.combin (parameter) event_date: date
    current_event_end = datetime.datetime.combine(event_date, end_time)

    # Convert 'Start Time' and 'End Time' columns to datetime.time
    schedule['Start Time'] = schedule['Start Time'].apply(lambda x: datetime.datetime.strptime(x, '%H:%M').time())
    schedule['End Time'] = schedule['End Time'].apply(lambda x: datetime.datetime.strptime(x, '%H:%M').time())

    # Sort schedule by date and start time
    schedule['Date'] = pd.to_datetime(schedule['Date'])
    schedule = schedule.sort_values(by=['Date', 'Start Time']).reset_index(drop=True)
```

```python
# Check for conflicts
for i, row in schedule.iterrows():
    row_date = row['Date'].date()
    if row_date == event_date:
        if (current_event_start.time() < row['End Time'] and current_event_start.time() >= row['Start Time']) or \
           (current_event_end.time() > row['Start Time'] and current_event_end.time() <= row['End Time']) or \
           (current_event_start.time() <= row['Start Time'] and current_event_end.time() >= row['End Time']):
            # Conflict found, now look for next available slot
            for j in range(i, len(schedule)):
                next_start = schedule.at[j, 'End Time']
                if j == len(schedule) - 1:
                    # Check end of day availability
                    if next_start <= datetime.time(23, 59):
                        next_event_start = next_start
                        next_event_end = (datetime.datetime.combine(event_date, next_event_start) + datetime.timedelta(minutes=duration_minutes)).time()
                        if next_event_end <= datetime.time(23, 59):
                            return f"Waktu konflik dengan acara: {row['Event']} dari {row['Start Time']} hingga {row['End Time']}. Waktu terdekat yang tersedia: {next_event_start} hingga {next_event_end}"
                else:
                    next_event_start = next_start
                    next_event_end = (datetime.datetime.combine(event_date, next_event_start) + datetime.timedelta(minutes=duration_minutes)).time()
                    if next_event_end <= schedule.at[j + 1, 'Start Time']:
                        return f"Waktu konflik dengan acara: {row['Event']} dari {row['Start Time']} hingga {row['End Time']}. Waktu terdekat yang tersedia: {next_event_start} hingga {next_event_end}"

            # Check before the first event
            if i == 0:
                if current_event_end <= schedule.at[i, 'Start Time']:
                    return f"Waktu konflik dengan acara: {row['Event']} dari {row['Start Time']} hingga {row['End Time']}. Waktu terdekat yang tersedia: {start_time} hingga {end_time}"

# If no conflict, return original times
return "Tidak ada konflik, waktu yang tersedia: {} hingga {}".format(start_time, end_time)
```

Fungsi find_nearest_available_time() seperti namanya fungsi ini digunakan untuk mengecek dimana waktu kosong yang ada ketika terdapat bentrok antara 2 jadwal atau kegiatan yang ingin di modifikasi.

Fungsi: find_nearest_available_time(tanggal_acara, waktu_mulai, waktu_selesai)

1. **MULAI**
2. **MEMUAT JADWAL:**
    - Panggil fungsi load_schedule untuk memuat jadwal yang tersimpan.
3. **KONVERSI DATA ACARA BARU:**
    - Ubah tanggal_acara menjadi objek date menggunakan pd.to_datetime(tanggal_acara).date().
    - Hitung durasi acara dalam menit: durasi_menit = (datetime.datetime.strptime(waktu_selesai, '%H:%M') - datetime.datetime.strptime(waktu_mulai, '%H:%M')).total_seconds() / 60.

- Ubah waktu_mulai dan waktu_selesai menjadi objek time menggunakan datetime.datetime.strptime(waktu, '%H:%M').time().
- Gabungkan tanggal_acara dan waktu_mulai menjadi objek datetime untuk awal acara saat ini (current_event_start).
- Gabungkan tanggal_acara dan waktu_selesai menjadi objek datetime untuk akhir acara saat ini (current_event_end).

4. **KONVERSI KOLOM JADWAL:**
   - Ubah kolom 'Start Time' dan 'End Time' dalam schedule menjadi objek datetime.time menggunakan fungsi apply.

5. **URUTKAN JADWAL:**
   - Konversi kolom 'Date' dalam schedule menjadi format datetime menggunakan pd.to_datetime.
   - Urutkan schedule berdasarkan 'Date' dan 'Start Time' menggunakan schedule.sort_values dan atur ulang indeks dengan reset_index(drop=True).
6. **CEK KONFLIK:**
   - Iterasi setiap baris dalam schedule menggunakan schedule.iterrows().
     - Dapatkan tanggal baris saat ini (tanggal_baris) dari kolom 'Date'.
     - **JIKA** tanggal_baris SAMA DENGAN tanggal_acara:
       - Periksa konflik waktu: (sama seperti fungsi sebelumnya)
         - **JIKA** ada konflik:
           - Iterasi baris schedule mulai dari baris konflik (i) hingga akhir jadwal.
             - Dapatkan waktu akhir baris berikutnya (next_start).
             - **JIKA** di akhir hari (baris terakhir):
               - Periksa ketersediaan hingga akhir hari (23:59).
                 - **JIKA** tersedia, hitung waktu akhir acara baru (next_event_end).
                 - **JIKA** tersedia dan tidak konflik dengan jadwal berikutnya, kembalikan waktu terdekat yang tersedia.
             - **JIKA** bukan akhir hari:
               - Hitung waktu akhir acara baru (next_event_end).

        - **JIKA** tersedia dan tidak konflik dengan jadwal berikutnya, kembalikan waktu terdekat yang tersedia.
    - **JIKA** tidak ada waktu tersedia hingga akhir hari, periksa sebelum acara pertama.
        - **JIKA** awal acara baru tidak konflik dengan jadwal pertama, kembalikan waktu awal acara baru sebagai waktu terdekat yang tersedia.
7. **TIDAK ADA KONFLIK:**
    - **JIKA** tidak ada konflik, kembalikan pesan "Tidak ada konflik, waktu yang tersedia: {waktu_mulai} hingga {waktu_selesai}".
8. **SELESAI**

C. PENERAPAN MODEL PADA SISTEM AKSI MANAJEMEN KEGIATAN

Model yang telah dikembangkan terhubung ke dalam sistem manajemen jadwal untuk memfasilitasi pengambilan keputusan berdasarkan percakapan yang sedang berlangsung. Proses ini mengintegrasikan kecerdasan buatan ke dalam alur kerja sistem manajemen jadwal, memungkinkan sistem untuk memberikan respons atau aksi berdasarkan analisis terhadap masukan pengguna.

Langkah-langkah Penerapan:

1. Integrasi Model

Model yang telah dilatih, misalnya untuk pemrosesan bahasa alami atau prediksi berbasis teks, diintegrasikan ke dalam sistem manajemen jadwal. Hal ini dilakukan dengan memanfaatkan antarmuka yang memungkinkan sistem untuk berkomunikasi dengan model.

2. Pemrosesan Percakapan

Ketika pengguna berinteraksi dengan sistem melalui percakapan, teks dari percakapan tersebut dianalisis menggunakan model yang terhubung. Model akan mengenali pola atau tujuan dari percakapan berdasarkan input pengguna.

3. Pengambilan Keputusan

Berdasarkan analisis yang dilakukan oleh model, sistem manajemen jadwal dapat mengambil keputusan yang relevan. Misalnya, sistem dapat menyarankan jadwal atau tindakan yang sesuai dengan permintaan pengguna yang terdeteksi dari percakapan.

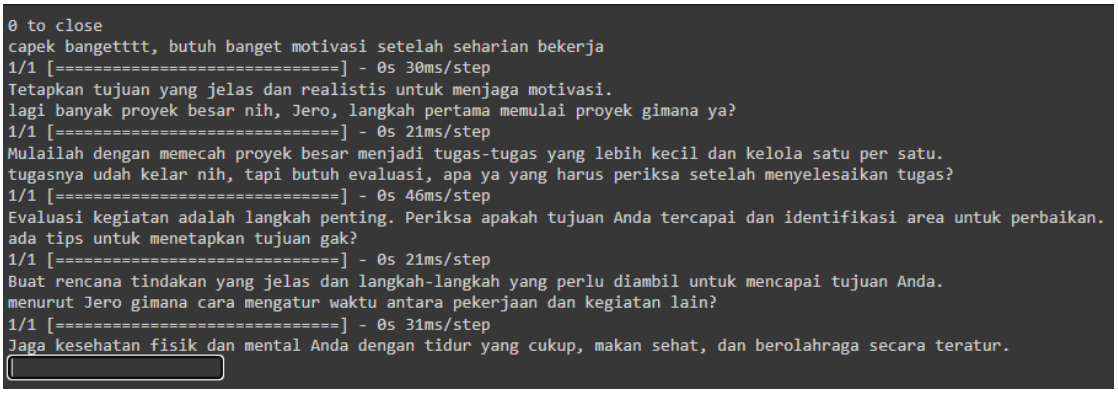
4. Respons Interaktif

Sistem memberikan respons kembali kepada pengguna berdasarkan hasil analisis model. Respons ini dapat berupa informasi terjadwal, pengingat, atau rekomendasi lain yang sesuai dengan kebutuhan pengguna.

D. PENGETESAN SISTEM

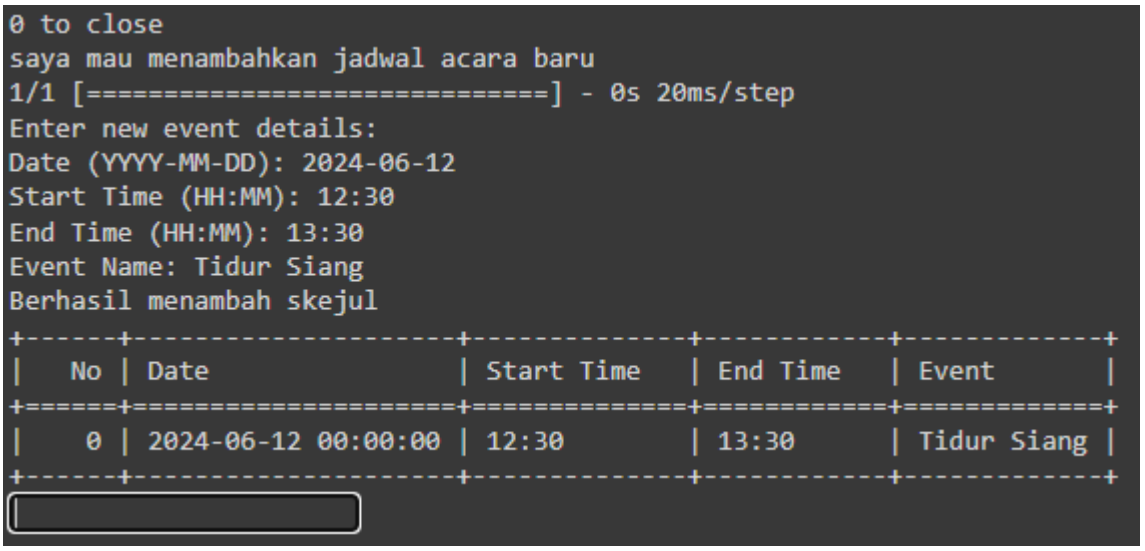
Sistem ini telah diuji menggunakan User Acceptance Testing (UAT) dengan beberapa skenario yang telah ditentukan, yaitu:

a. Mengajak ngobrol



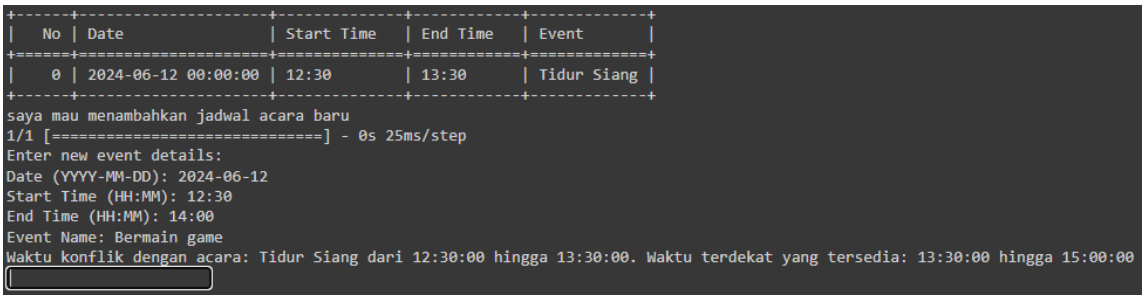
manajerobot, merespon percakapan dari user dengan baik dan interaktif

b. Menambah jadwal



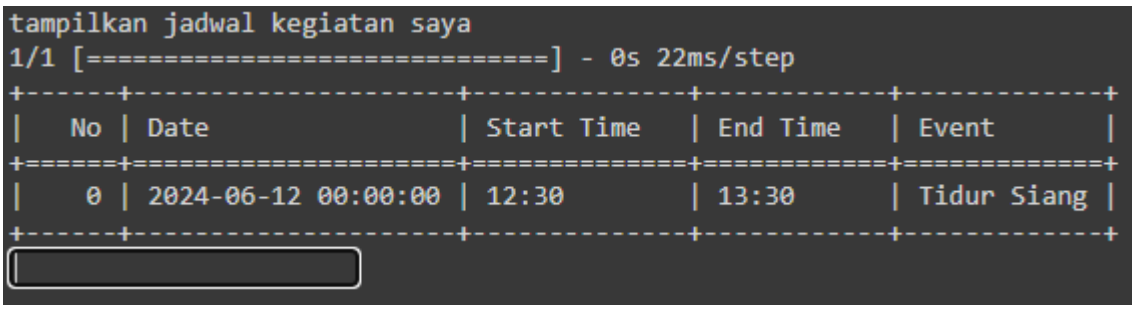
manajerobot, dapat merespon user ketika ingin menambahkan jadwal baru.

c. Menambah jadwal namun bentrok dengan kegiatan lain



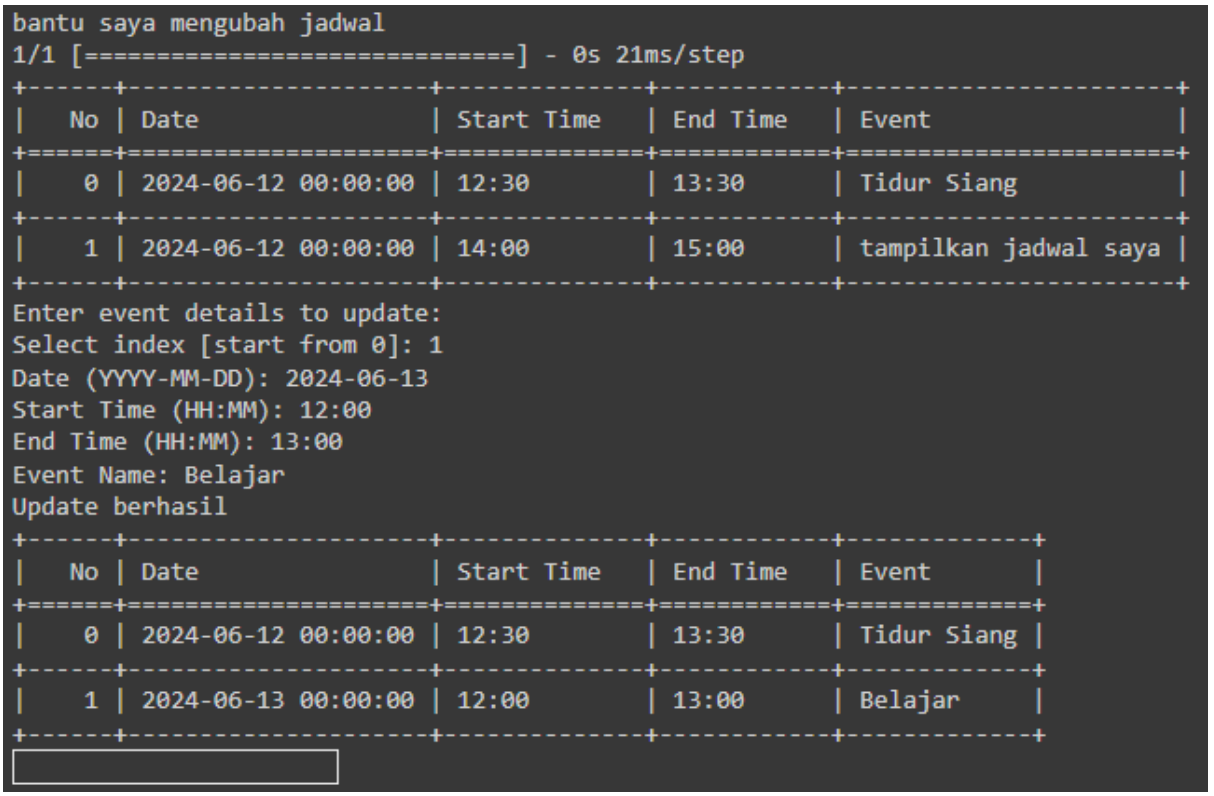
Jika user ingin menambahkan jadwal/kegiatan baru, namun memiliki waktu yang bentrok dengan jadwal yang sudah ada maka akan menampilkan saran berupa perubahan jadwal ke waktu kosong terdekat.

d. Menampilkan jadwal



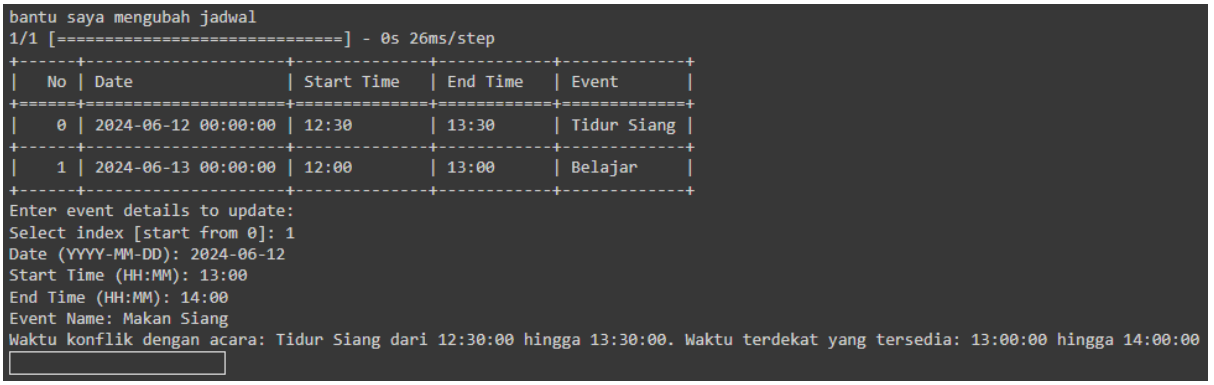
manajerobot, merespon user dengan baik ketika user meminta untuk menampikan jadwal.

e. Mengupdate jadwal



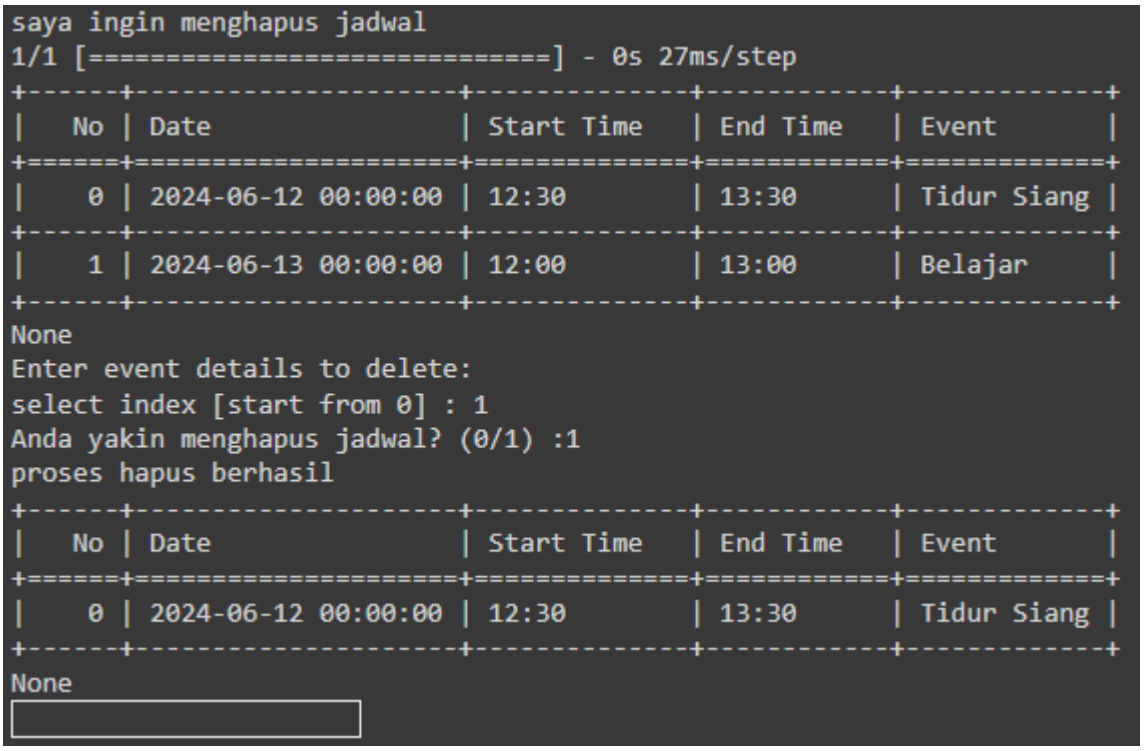
manajerobot dapat merespon user ketika diminta untuk melakukan perubahan jadwal kegiatan

f. Mengupdate jadwal namun bentrok dengan kegiatan lain



Jika user meminta manajerobot untuk mengubah jadwal namun terdapat bentrok, maka akan muncul pemberitahuan bahwa waktu yang diminta terdapat bentrok dan disarankan untuk mengisi ke waktu kosong terdekat.

g. Menghapus jadwal



manajerobot mampur merespon ketika user meminta untuk menghapus suatu jadwal, user akan diminta untuk memilih jadwal kemudian mengkonfirmasi penghapusan jadwal.

KESIMPULAN & SARAN

A. KESIMPULAN

Penelitian ini telah berhasil mengembangkan sistem asisten manajemen penjadwalan kegiatan virtual bernama "Manajerobot" yang memanfaatkan Natural Language Processing (NLP). Sistem ini mampu memahami bahasa alami pengguna, seperti perintah untuk menambahkan, memperbarui, dan menghapus jadwal, serta menjawab pertanyaan terkait jadwal. Manajerobot juga dapat memberikan saran waktu yang tersedia dan mendeteksi konflik jadwal.

Pengujian terhadap sistem menunjukkan hasil yang memuaskan. Sistem mampu memahami maksud pengguna dengan baik dan menyelesaikan tugas-tugas penjadwalan dengan efektif.

B. SARAN

Penelitian ini masih dapat dikembangkan lebih lanjut dengan beberapa saran berikut:

- **Meningkatkan kemampuan NLP:** Sistem dapat diperluas untuk memahami perintah yang lebih kompleks, seperti perintah untuk mengulang acara, mencari jadwal berdasarkan kata kunci, dan menggabungkan beberapa jadwal.
- **Integrasi dengan platform lain:** Manajerobot dapat diintegrasikan dengan platform lain, seperti kalender online, email, dan aplikasi pesan instan, untuk meningkatkan kemudahan penggunaan.
- **Pengembangan personalisasi:** Sistem dapat dipersonalisasi untuk menyesuaikan dengan kebutuhan pengguna yang berbeda, seperti preferensi waktu dan jenis kegiatan.
- **Evaluasi lebih lanjut:** Diperlukan evaluasi yang lebih luas dengan melibatkan lebih banyak pengguna dan skenario penggunaan yang beragam untuk meneliti efektivitas dan kegunaan sistem secara lebih menyeluruh.

Secara keseluruhan, Manajerobot merupakan sistem asisten manajemen penjadwalan kegiatan virtual yang menjanjikan dengan potensi untuk meningkatkan efisiensi dan efektivitas dalam mengelola jadwal kegiatan.

DAFTAR PUSTAKA

Huda, I. (2019). Implementasi Natural Language Processing (NLP) untuk Aplikasi Pencarian Lokasi.

ProjectPro.io. 2023. Membangun Chatbot Python dari Awal. https://www.projectpro.io/article/python-chatbot-project-learn-to-build-a-chatbot-from-scratch/429. Diakses tanggal 12 Juni 2024.

Malik, Karan. 2023. Chatbot Karan Malik. https://github.com/Karan-Malik/Chatbot. Diakses tanggal 12 Juni 2024.