Pembangunan Aplikasi Identifikasi Kesalahan Ketik Dokumen Berbahasa Indonesia Menggunakan Algoritma *Jaro-Winkler Distance*

(Development of Typographical Error Identification Application in Indonesian Language Using Jaro-Winkler Distance Algorithm)

Grelly Lucia Yovellia Londo[1], Yohanes Sigit Purnomo W.P.[2], Martinus Maslim[3]

[1,2,3]*Program Studi Teknik Informatika, Universitas Atma Jaya Yogyakarta, Indonesia*

[1]grellylondo@gmail.com
[2]sigit.purnomo@uajy.ac.id
[3]martinus.maslim@uajy.ac.id

Abstrak - Teks adalah salah satu media yang digunakan manusia untuk berkomunikasi dan berinteraksi setiap hari, terutama di bidang pendidikan, misalnya dalam penulisan laporan tugas akhir. Hal yang paling umum dalam menulis teks adalah kesalahan ketik. Berdasarkan masalah ini, diperlukan suatu aplikasi untuk membantu mengidentifikasi kesalahan ketik dalam dokumen Bahasa Indonesia. Aplikasi yang dikembangkan menggunakan Laravel versi 5.8 untuk tampilan web dan Python versi 3 untuk memproses dataset dan membuat model serta layanan webnya. Model dibangun menggunakan modul NLTK dan implementasi algoritma Jaro-Winkler Distance menggunakan modul pylibjaro. Dataset menggunakan dataset sumber terbuka dalam bentuk daftar kata-kata dari Kamus Besar Bahasa Indonesia (KBBI). Aplikasi ini hanya mendukung file pdf. Hasil model diterapkan ke layanan web dengan output dalam bentuk data JSON. Data JSON berisi daftar kata-kata yang memiliki nilai benar atau salah, jumlah kata dokumen, jumlah kata yang benar, jumlah kata yang salah, dan waktu pelaksanaan program.

Kata-kata kunci: Teks, Dokumen, Bahasa Indonesia, Kesalahan Ketik, *Jaro-Winkler Distance*

Abstract - Text is one of the media used by humans to communicate and interact every day, especially in the field of education, for example, in writing a final project report. The most common thing in writing text is typographical errors. Based on these problems, an application is needed to help the writer to be able to identify typographical errors in the Indonesian Language document. The application developed using Laravel version 5.8 for web application and Python version 3 for processing datasets, developing model, and developing web services. Model built uses the NLTK library and Jaro-Winkler distance algorithm implemented using the pylibjaro library. The dataset uses an open-source dataset in the form of a list of words from KBBI. This application only supports pdf files. The results of the model are applied to the web services with output in the form of JSON data. The JSON data contains a list of words that have true or false values, the number of document words, the number of correct words, the number of incorrect words, and the time of program execution.

Keywords: Text, Document, Bahasa Indonesia, Typographical Error, Jaro-Winkler Distance

I. PENDAHULUAN

Teks adalah salah satu media komunikasi yang digunakan manusia dalam kehidupan sehari-hari. Teks memiliki peran penting dalam bidang pendidikan [1] . Selama menjalani masa perkuliahan, mahasiswa dituntut untuk menulis berbagai macam dokumen seperti jurnal, makalah atau pun skripsi. Namun, kenyataan di lapangan saat ini, hal yang sering terjadi pada saat penulisan dokumen adalah kesalahan penulisan atau *typographical error*.

Typographical error tidak bisa lepas dari mahasiswa. Sejak dini mahasiswa telah dilatih kemampuan menulisnya dengan membuat laporan praktikum ataupun makalah untuk memenuhi tugas mata kuliah tertentu. Hal tersebut masih belum cukup karena pengetahuan dasar mahasiswa tentang penulisan Bahasa Indonesia dinilai masih kurang. Menurut Murtingsih, kesalahan penulisan kata yang tidak sesuai konteks sering dilakukan mahasiswa sebanyak 69,2% dari jumlah subjek penelitian yaitu 260 mahasiswa. Selain itu kesalahan penulisan imbuhan dan kata depan sebanyak 17,4% dari total 260 mahasiswa [2]. *Typographical error* cukup sering terjadi saat mahasiswa menulis jurnal, makalah atau pun skripsi. Mahasiswa sering kali mengalami revisi

dikarenakan kesalahan penulisan yang tidak sesuai dengan kaidah Ejaan Yang Disempurnakan (EYD). Hal ini sesuai dengan yang ditulis oleh Javed, Juan dan Nazli bahwa kemampuan menulis lebih sulit dan paling akhir dikuasai [3]

Terdapat tiga faktor utama yang menyebabkan sering terjadinya *typographical error* pada dokumen yang dibuat oleh mahasiswa. Faktor pertama adalah kurangnya pemahaman mahasiswa dalam pembuatan atau penulisan dokumen Bahasa Indonesia yang benar dan sesuai dengan EYD. Hal ini sering sekali ditemukan pada tulisan-tulisan mahasiswa seperti penempatan tanda baca yang kurang tepat, penggunaan huruf kapital dan huruf kecil yang kurang tepat dan lain sebagainya. Faktor kedua adalah kurangnya kosakata standar yang dipahami oleh mahasiswa. Kurangnya intensitas waktu membaca literatur membuat pengetahuan mahasiswa tentang kosakata standar dinilai kurang. Faktor ketiga yaitu kesulitan dalam menyatukan ide mahasiswa dan kutipan dari bacaan yang digunakan. Pada penulisan skripsi atau tugas akhir, mahasiswa dituntut untuk menulis sesuatu berdasarkan fakta yang diambil dari sumber-sumber terpercaya seperti jurnal, *paper*, skripsi atau tugas akhir yang sudah ada maupun buku yang berkaitan. Untuk menghindari plagiarisme mahasiswa harus menggunakan kutipan tersebut dengan merangkum atau menyatukan dengan idenya sendiri tanpa mengurangi makna dari kutipan tersebut.

Pembangunan aplikasi identifikasi *typographical error* pernah dilakukan sebelumnya oleh beberapa peneliti. Y. Rochmawati dan R. Kusumaningrum membandingkan beberapa algoritma pencarian s*tring* untuk mengidentifikasi kesalahan pengetikan teks [4]. Hasil dari penelitian ini adalah algoritma Jaro-Winkler Distance dapat melakukan pengecekan kata dengan nilai *mean average precision* (MAP) sebesar 0,87, Hamming Distance memiliki nilai MAP 0,46, Levenshtein Distance sebesar 0,74, dan Damerau Levenshtein sebesar 0,85. Algoritma Jaro Winkler Distance dinilai paling baik dibanding dengan tiga metode lainnya. Algoritma Jaro-Winkler Distance juga digunakan untuk pengkoreksian dan *suggestion word* pada kata kunci [5] dan pengoreksi kesalahan penulisan Bahasa Indonesia [6].

Selain Jaro-Winkler, *dictionary lookup* dan algoritma Damerau-Levenshtein Distance juga digunakan untuk mendeteksi kesalahan pada ejaan dan merekomendasikan perbaikan kata [7]. Algoritma Damerau-Levenshtein Distance lebih unggul jika dibandingkan dengan algoritma Levenshtein Distance. A. I. Fahma, I. Cholissodin, dan R. S. Perdana menggunakan metode *N-gram* dan Levenshtein Distance untuk melakukan prediksi kata, koreksi ejaan serta menentukan kandidat kata [1]. Hasil dari penelitian ini yaitu pendekatan *dictionary lookup* dapat diterapkan dengan baik untuk proses pencarian kata yang mengalami *typographical error* dalam dokumen Bahasa Indonesia yang diinputkan dan Metode Levenshtein Distance dapat digunakan untuk menentukan kandidat kata yang hasilnya sesuai dengan nilai aktual yang diharapkan pengguna. Selain itu, W. W. A. Umboh, S. R. Sentinuwo dan A. M. Sambul menggunakan metode *full text indexing* untuk mendeteksi kesalahan atau *typographical error* [8].

C. I. Ratnasari, S. Kusumadewi, and L. Rosita mengevaluasi kinerja *spell checker* untuk keluhan pasien berbahasa Indonesia [9]. *Spell checker* ini menggunakan metode *dictionary lookup* dan algoritma Levenshtein Distance. Metode *dictionary lookup* sebagai metode identifikasi kesalahan kata memiliki akurasi sebesar 97,59 % sedangkan algoritma Levenshtein Distance sebagai algoritma untuk mengoreksi kesalahan kata memiliki akurasi sebesar 94,03%. Semakin banyak kata yang disimpan dalam kamus makan semakin akurat hasil yang dikeluarkan. Algoritma Levenshtein Distance juga digunakan untuk membantu *autocomplete* dan *spell checking* [10]. Hema P.H dan Sunitha C [11] melakukan survey mengenai metode deteksi kesalahan kata yang digunakan oleh beberapa *tool spell checker* untuk *non-word error* pada Bahasa yang berbeda. Dalam penelitian ini dikatakan bahwa teknik yang paling sering digunakan dalam mendeteksi kesalahan kata mencakup *non-word error* adalah *dictionary lookup* dan analisis *N-gram*. A. Prasetyo, W. M. Baihaqi, and I. S. Had berhasil menerapkan algoritma Jaro Winkler untuk *autocorrect* dan *spelling suggestion* pada penulisan naskah berita di BMS TV [12]. Algoritma Jaro-Winkler juga digunakan untuk memperbaiki *spell checking* pada teks berbahasa Arab [13]. Algoritma lainnya yang digunakan untuk permasalahan *spelling checker* adalah algoritma Peter Norvig dan BK-Trees [14].

Untuk membantu mahasiswa dalam mengatasi masalah kesalahan penulisan dokumen tersebut diperlukan suatu aplikasi yang dapat melakukan pengecekan kesalahan penulisan atau pengetikan teks. Saat ini sudah terdapat beberapa aplikasi yang serupa namun masih sangat kurang untuk pengecekan dokumen berbahasa Indonesia. Untuk itu dibuat suatu aplikasi *website* sederhana yang dapat melakukan pengecekan kesalahan penulisan atau pengetikan teks pada dokumen berbahasa Indonesia yang menerapakan *natural language processing* dengan menggunakan algoritma Jaro-Winkler Distance. Dengan adanya aplikasi ini, diharapkan dapat membantu mahasiswa dalam

mengatasi masalah kesalahan penulisan atau pengetikan teks.

II. METODE

Metode yang digunakan dalam membangun dan mengintegrasikan aplikasi ini adalah sebagai berikut:

A. Studi Pustaka

Langkah pertama yang dilakukan pada penelitian ini adalah melakukan studi pustaka. Langkah ini berfokus untuk mencari referensi dan melakukan pembelajaran mengenai algoritma Jaro-Winkler *distance* dan penerapannya pada model NLP serta penggunakan pemrograman paralel pada Python. Pada langkah ini dilakukan pengumpulan literasi-literasi yang dapat digunakan untuk membantu proses penelitian.

B. Pengumpulan Bahan

Pada langkah ini dilakukan pencarian dan pengumpulan *dataset* daftar kata Bahasa Indonesia. *Dataset* daftar kata Bahasa Indonesia ini bersumber dari Kamus Besar Bahasa Indonesia (KBBI). Penulis menggunakan *dataset* yang sudah ada yaitu diambil dari Github Jim Geovedi yang dimodifikasi kembali.

C. Analisis Dataset

Pada langkah ini dilakukan analisis *dataset*. Analisis *dataset* ini dilakukan untuk menyempurnakan *dataset* yang sudah ada agar sesuai dengan kebutuhan. *Dataset* yang dibutuhkan berupa daftar kata Bahasa Indonesia yang sesuai dengan yang ada di Kamus Besar Bahasa Indonesia (KBBI).

D. Pembuatan Model

Pada langkah ini dilakukan proses pembuatan model identifikasi *typographical error*. Model ini mengimplementasikan algoritma Jaro-Winkler *distance* untuk menentukan suatu kata dianggap benar atau salah. Model menggunakan *library pyjarowinkler* 1.8 yang mengimplementasikan algoritma Jaro-Winkler *distance*. Pemrograman paralel digunakan untuk mempercepat proses eksekusi model. Model ini juga menggunakan *library* nltk untuk memproses kata.

E. Preprocessing Data

Pada langkah ini dilakukan *preprocessing* data. *Preprocessing* data meliputi pembersihan data teks yaitu dengan menghapus karakter khusus dan melakukan tokenisasi teks.

F. Evaluasi Model

Pada langkah ini dilakukan pengujian model untuk mendapatkan nilai evaluasi. Nilai evaluasi ini didapatkan dengan melakukan pengujian manual pada model dengan menginputkan dokumen pdf dengan jumlah kata tertentu lalu mengkonversi nya menjadi data teks.

G. Pembuatan Web Service

Pada langkah ini dilakukan pembuatan *web service*. Pembuatan *web service* dilakukan agar aplikasi *web* dapat mengakses model dengan lebih mudah. *Web service* dibuat menggunakan Python dengan memanfaatkan *framework flask*. *Web service* di-*deploy* menggunakan *server* Apache Ubuntu.

H. Pembuatan Aplikasi Web

Langkah ini meliputi pembuatan aplikasi berbasis *web* menggunakan Laravel 5.8. Aplikasi *web* ini mengakses model identfikasi *typographical error* melalui *web service* yang telah dibuat.

I. Pengujian Aplikasi Web

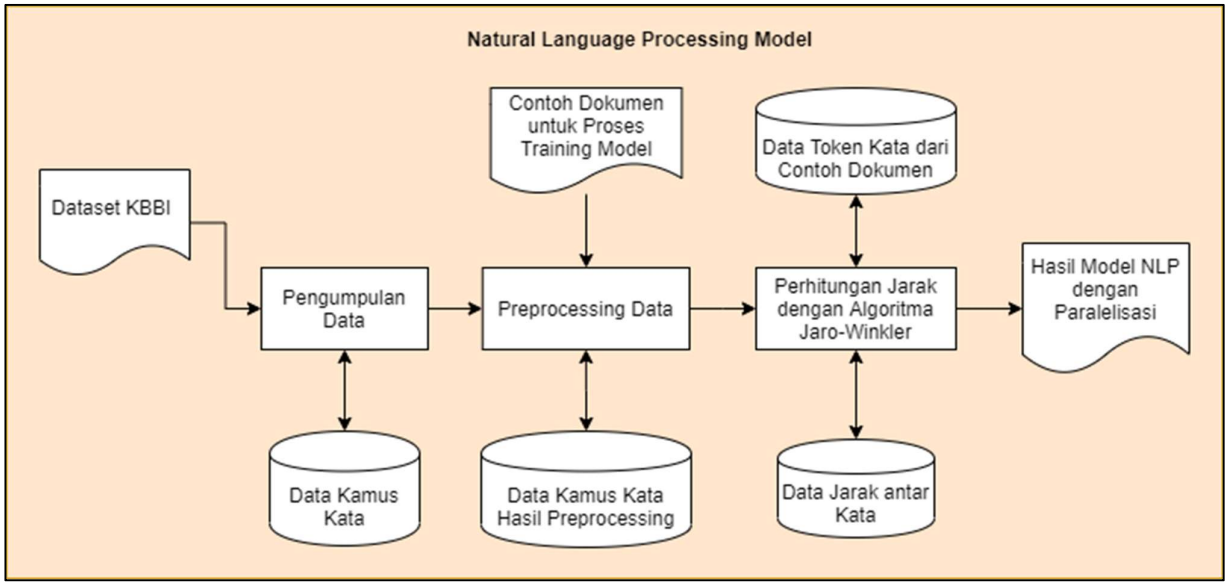
Langkah ini meliputi pengujian akhir yaitu pengujian aplikasi *web*. Aplikasi *web* dicek dengan mengunggah *file* dokumen. Selanjutnya dicek kata mana saja yang salah berdasarkan *dataset* yang ada. Metode untuk mengukur akurasi dari aplikasi *web* ini masih dilakukan secara manual yaitu dari penilaian pengguna secara langsung setelah melakukan pengecekan *typographical error* menggunakan aplikasi *web*.

III. HASIL DAN PEMBAHASAN

Untuk dapat mendeteksi kesalahan penulisan pada dokumen dibutuhkan suatu model yang menerapkan *natural language processing (NLP)*. Gambar 1 menunjukkan model NLP untuk proses deteksi kesalahan penulisan pada dokumen. Model yang dihasilkan akan digunakan dalam aplikasi dalam bentuk Web Service dan akan dijelaskan pada bagian F.

A. Pengumpulan Data

Pada langkah ini dilakukan pencarian dan pengumpulan dataset daftar kata Bahasa Indonesia. Dataset daftar kata Bahasa Indonesia ini bersumber dari Kamus Besar Bahasa Indonesia (KBBI). Penulis menggunakan dataset yang sudah ada yaitu diambil dari Github Jim Geovedi yang dimodifikasi kembali oleh penulis (https://github.com/geovedi/indonesian-wordlist/blob/master/01-kbbi3-2001-sort-alpha.lst).



Gambar 1. Model NLP untuk deteksi kesalahan penulisan

B. Preprocessing Data

Salah satu hal terpenting dalam pemrosesan teks adalah tahap *preprocessing* data. Tahap ini dilakukan untuk membersihkan data mentah yang dalam hal ini adalah data teks. Ketika sebuah model memproses data teks, tidak semua data digunakan dan perlu ditentukan bentuk data yang digunakan pada model. Misalnya dalam kasus klasifikasi teks artikel, tidak semua kata pada dokumen perlu diproses seperti *stop words,* sehingga *stop words* tersebut perlu dihilangkan terlebih dahulu dengan menggunakan sebuah fungsi untuk menghapus *stop words.* Pada penelitian ini dibutuhkan keseluruhan kata yang ada pada dokumen karena setiap kata dicek apakah kata tersebut benar atau tidak berdasarkan dataset yang ada. Oleh karena itu, dibutuhkan proses tokenisasi teks yang bertujuan untuk memecah teks dokumen menjadi bagian-bagian kata yang terpisah. Gambar 2 menunjukkan contoh hasil proses tokenisasi.

C. Implementasi Jaro-Winkler pada Model

Pembuatan model menggunakan *library pyjarowinkler* versi 1.8. *Library* ini merupakan *library* Python untuk menghitung nilai Jaro-Winkler *distance*. Konsep penerapan Jaro-Winkler *distance* pada model identifikasi *typographical error* ini adalah setiap kata yang dimasukkan dibandingkan dengan kata yang ada di *dataset*. *Dataset* ini merupakan representasi kata yang dianggap benar. Setiap kata dicek satu persatu apakah nilai Jaro-Winkler *distance* nya bernilai 1 atau bukan. Jika bernilai satu maka kata yang dimasukkan tersebut dianggap benar. Perulangan digunakan agar dapat melakukan pengecekan keseluruhan kata yang dimasukkan. Kode fungsi untuk proses ini dapat dilihat pada Gambar 3.

```
['Teks', 'adalah', 'salah', 'satu', 'media', 'komunikasi', 'yang', 'digunakan', 'manusia', 'dalam',
'kehidupan', 'sehari', 'hari', 'Teks', 'berperan', 'penting', 'dalam', 'bidang', 'pendidikan', '1',
'Dalam', 'dokumen', 'formal', 'seperti', 'skripsi', 'atau', 'tugas', 'akhir', 'semuanya', 'masih', 'm
enggunakan', 'teks', 'sebagai', 'media', 'utamanya', 'Hal', 'yang', 'paling', 'sering', 'terjadi', 'p
ada', 'saat', 'pembuatan', 'dokumen', 'adalah', 'kesalahan', 'penulisan', 'atau', 'typographical', 'e
rror']
```

Gambar 2. Contoh hasil proses tokenisasi kalimat

```
def check_correct_word(document, output):
    winkler = True
    scaling = 0.1
    correct_word = None

    wordlist = read_dictionary()
    for word in wordlist:
        pylibjaro = distance.get_jaro_distance(word, document, winkler, scaling)

        if pylibjaro == 1.0 :
            correct_word = document
            break
    output.put(correct_word)

```

Gambar 3. Kode fungsi implementasi library Jaro-Winkler *distance*

D. Implementasi Parallel Processing pada Model

Model identifikasi *typographical error* ini menggunakan dataset dengan jumlah yang cukup banyak yaitu 61.309 daftar kata Bahasa Indonesia yang diambil dari Kamus Besar Bahasa Indonesia (KBBI). Artinya, model melakukan pengecekan setiap iterasi kata inputannya dengan kata yang ada di dataset. Dengan banyaknya jumlah perulangan dan iterasi ini, maka untuk meningkatnya performa dan kecepatan waktu eksekusi diimplementasikan *parallel processing*. Ada beberapa package atau library python yang mendukung *parallel processing* salah satunya adalah *multiprocessing*.

Multiprocessing mendukung suatu sistem untuk melakukan tugas di beberapa unit pemroses atau disebut *cores* pada waktu bersamaan. Maksimal jumlah proses yang dapat dijalankan tergantung dari jumlah prosesor pada komputer. Pada *package multiprocessing* Python terdapat dua objek utama yaitu *process class* dan *pool class*. *Process class* lebih efisien digunakan pada kasus dimana tidak banyak proses yang dieksekusi namun memiliki masukan yang banyak. Sedangkan *pool class* lebih efisien digunakan pada kasus dengan jumlah proses yang besar [15].

Pada penerapannya, digunakan *process class multiprocessing* karena jumlah proses yang dieksekusi sedikit namun memiliki masukan atau iterasi yang cukup besar. Proses *class* mengirim kode ke prosesor segera setelah proses dimulai. Gambar 4 menunjukkan baris kode yang mengimplentasikan *multiprocessing* dan Gambar 5 menunjukkan bagaimana alur kerja sistem *multiprocessing* yang terjadi.

```
def jaro_parallel_runner(inputs):
    global correct_words

    output = multiprocessing.Queue()
    start_time = time.time()
    results = {}
    all_words = []

    # multiprocessing for checking correct words
    processes_correct = [multiprocessing.Process(target=check_correct_word, args=(inputs[i], output))
                         for i in range(len(inputs))]

    for process in processes_correct:
        process.start()

    for process in processes_correct:
        process.join()

    result = [output.get() for process in processes_correct]
    correct_words = list(filter(None, result))

    # multiprocessing for checking all words status based on existing correct words
    for i in range(len(inputs)):
        process = multiprocessing.Process(target=check_all_words, args=(inputs[i], output, i))
        process.start()
        process.join
        hasil = output.get()
#         all_words.update(hasil)
        all_words.append(hasil)

    results['result'] = all_words
    results['number_of_words'] = len(inputs)
    results['correct'] = len(correct_words)
    results['typo'] = (len(inputs) - len(correct_words))
    results['time'] = '{} seconds'.format(time.time() - start_time)

    return results

```

Gambar 4. Implentasi *multiprocessing* pada model menggunakan Python

Gambar 5. Diagram alur kerja sistem *multiprocessing*

Pada baris kode di atas terdapat dua *multiprocessing* untuk dua proses berbeda namun proses kedua dependen dengan proses pertama. Proses pertama adalah proses untuk mengecek apakah kata masukan benar atau salah. Setiap kata masukan mewakili satu iterasi. Setiap satu iterasi melakukan pengecekan kesalahan kata dengan membandingkan kata masukan dengan kata-kata yang terdapat pada dataset. Hasil dari pengecekan kata ini ditampung dalam variabel *result*. Proses kedua adalah proses untuk memberikan status terhadap setiap kata. Kata yang terdapat pada *list result* atau kata yang dianggap benar diberi status *true* dan sebaliknya diberi status *false*.

E. Evaluasi Model

Evaluasi yang dilakukan terhadap model identifikasi *typographical error* dilakukan dengan melakukan pengecekkan dokumen *pdf* dengan jumlah kata yang berbeda. Dari evaluasi tersebut terlihat kata mana saja yang dianggap benar dan salah serta waktu eksekusinya. Hasil evaluasi model ditunjukkan pada Tabel 1 dan Tabel 2.

TABEL I
TABEL PERBANDINGAN WAKTU EKSEKUSI PARALLEL PROCESSING
PADA TIGA KOMPUTER

No	Jumlah Kata	Waktu Eksekusi (detik)			
		Dell Vostro (Windows)	Dell Vostro (Ubuntu)	Asus TUF Gaming	Server grid UAJY
1.	10	46,153	8,067	21,478	2,377
2.	50	171,496	51, 240	105,486	9,758
3.	100	456,440	85,972	227,476	23,796

TABEL II
TABEL PERBANDINGAN WAKTU EKSEKUSI MODEL ANTARA SERIAL DAN PARALEL

No	Jumlah Kata	Waktu Eksekusi (detik)	
		Serial	Paralel
1.	10	46,153	8,067
2.	50	171,496	51, 240
3.	100	456,440	85,972

F. Implementasi Web Service

Agar model yang dibuat dapat digunakan dengan mudah di berbagai platform maka model tersebut diimplementasikan pada *web service*. Pembuatan *web service* ini menggunakan Bahasa Python dengan memanfaatkan *framework* Flask. *Web service* dengan Python dapat dibuat menggunakan *Integrated Development Environment* (IDE) Python seperti Pycharm. Untuk memudahkan proses *deployment*, penulis membuat *web service* menggunakan sistem operasi Ubuntu 18.04 dengan Python versi 3.6.8. Teks editor yang digunakan adalah Nano yaitu teks editor bawaan dari Ubuntu. Selain mempermudah proses *deployment,* waktu eksekusi model pada sistem operasi Ubuntu menjadi lebih cepat dibandingkan dengan menggunakan Windows.

Proses *deployment web service* pada server Apache Ubuntu menggunakan Virtualenv. Virtualenv merupakan *tool* untuk memisahkan *package-package* yang digunakan pada proyek Python dari sistem operasi komputer atau laptop. Hal ini dilakukan untuk menghidari masalah pada sistem operasi ketika *developer* menginstal *package* atau komponen pada proyek Python. *Web service* di-*deploy* di *server* lokal Apache Ubuntu untuk pengembangan dan *server* kampus yaitu grid.uajy.ac.id dengan port 5000. Untuk melakukan *deployment* aplikasi Python yang menggunakan framework Flask dibutukan *Web Server Gateway Interface* (WSGI). Pada Apache HTTP *server* tersedia *mod* yang dapat memungkinkan Apache *server* menjalankan aplikasi Flask yaitu mod_wsgi. Sebelum itu pada *directory* proyek ditambahkan skrip wsgi. Skrip WSGI ini yang nantinya menjalankan aplikasi Flask pada *file* Python. Gambar 6 menunjukkan hasil implementasi model di dalam *web service*. Dapat dilihat untuk kasus suatu kalimat dapat terdeteksi jumlah kata yang benar pada bagian kanan bawah serta waktu yang dibutuhkan untuk mendeteksi kesalahan pada contoh kalimat tersebut. Gambar 6 juga menunjukkan bahwa kata-kata yang tidak ada pada dataset akan diberikan status *false*, seperti *typographical* dan *error*.

G. Implementasi Aplikasi Web

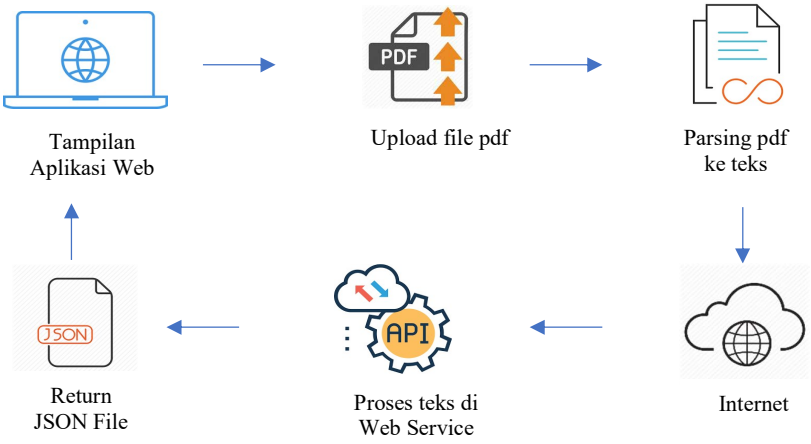
Pemilihan *platform web* berdasarkan pertimbangan kemudahan pengguna untuk menggunakan aplikasi ini sesuai dengan alur penggunaan aplikasi. Salah satu tahap penggunaan aplikasi ini adalah mengunggah *file* pdf. Dengan menggunakan aplikasi *web* lebih mudah untuk melakukan proses unggah dibandingkan dengan menggunakan aplikasi *mobile*. Gambar 7 menunjukkan alur aplikasi secara keseluruhan.

Secara lengkap alur dari proses aplikasi ini adalah sebagai berikut. Pada sisi klien menampilkan tampilan aplikasi *web* sederhana yang memiliki fitur untuk mengunggah *file* pdf (Gambar 8). Terdapat dua alternatif untuk mengunggah *file*, yaitu dengan meletakan file pada kotak yang terdapat pada halaman *web* atau dengan mengklik daerah kosong pada kotak di halaman *web* lalu memilih *file* yang diunggah.

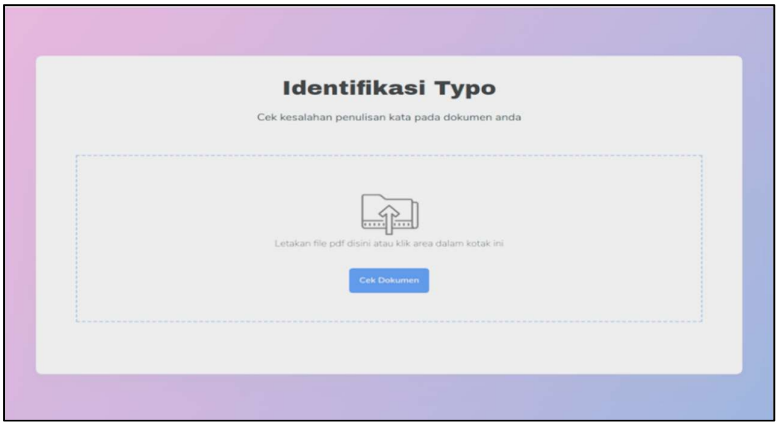
Setelah memilih *file*, lalu klik tombol cek dokumen. Proses pertama yang dilakukan setelah mengklik tombol masih terjadi di sisi klien. Kode Java Script melakukan *parsing* dokumen pdf menjadi teks sederhana. Selanjutnya teks ini yang diproses di *web service*. Proses kedua yaitu memproses teks pada *web service*. Proses ini terjadi di sisi *server*. Web service memproses teks dan mengeluarkan hasil berupa *file* JSON. *File* JSON ini diambil oleh web menggunakan AJAX dan diolah lagi untuk ditampilkan kembali dengan bentuk yang lebih mudah dibaca oleh pengguna. Hasil pengecekan dokumen pdf pada aplikasi ini ditunjukkan pada Gambar 9.


```
{
      ..............
      ..............
    {
      "id": 46,
      "text": "penulisan",
      "status": true
    },
    {
      "id": 47,
      "text": "atau",
      "status": true
    },
    {
      "id": 48,
      "text": "typographical",
      "status": false
    },
    {
      "id": 49,
      "text": "error",
      "status": false
    }
  ],
  "number_of_words": 49,
  "correct": 46,
  "typo": 3,
 "time": "9.61985993385315 seconds"
}
```

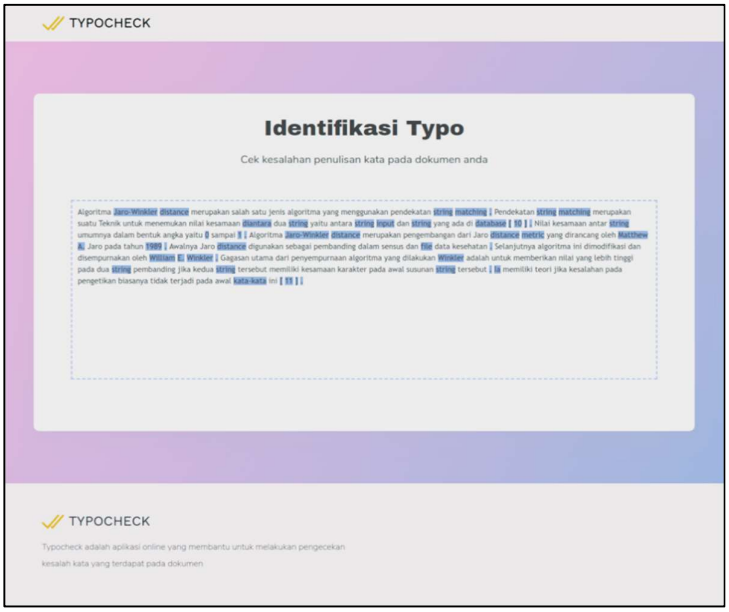

Gambar 6. Hasil implementasi dari *web service*



Gambar 7. Alur proses aplikasi *web*



Gambar 8. Tampilan awal aplikasi *web* bagian unggah file



Gambar 9. Hasil pengecekan *typo* menggunakan aplikasi *Typocheck*

Dari hasil diatas terlihat bahwa kata-kata yang dianggap salah diberi tanda warna biru. Kata-kata tersebut merupakan kata yang tidak ada di Kamus Besar Bahasa Indonesia. Rata-rata kata yang dianggap salah adalah kata yang merupakan entitas nama orang dan tahun. Selain itu kata dalam bahasa asing serta kata berulang masih dianggap salah oleh aplikasi. Aplikasi ini juga belum mampu untuk menampilkan kembali bentuk asli dari teks misalnya bentuk paragraf atau teks yang dicetak miring atau juga bentuk tabel.

IV. PENUTUP

Dari hasil penelitian yang telah dilakukan dapat diambil kesimpulan bahwa penggunaan algoritma Jaro-Winkler *distance* hanya dapat melakukan pengecekan kata berdasarkan *dataset* yang ada yaitu 61.309 daftar kata Bahasa Indonesia yang diambil dari Kamus Besar Bahasa Indonesia (KBBI). Untuk melakukan pengecekan kata-kata entitas dibutuhkan penambahan model untuk *Name Entity Recognation* (NER). Selain itu waktu eksekusi model tergantung dari jumlah *dataset* dan data masukan yang digunakan. Semakin banyak data semakin lama pula waktu eksekusinya. Waktu eksekusi dapat dipercepat menggunakan *parallel processing. Parallel processing* juga tidak menjamin waktu eksekusi dapat meningkat dengan signifikan tergantung dari spesifikasi komputer yang digunakan.

Berdasarkan penelitian yang sudah dilakukan masih banyak yang harus dikembangkan seperti pengecekan untuk kata-kata entitas atau pengecekan secara gramatikal serta dapat mengecek dokumen sesuai dengan aturan Bahasa Indonesia yang standar. Pengembangan pengecekan kata-kata entitas dapat dilakukan dengan mengimplementasikan model *Named Enitiy Recognition* (NER). Dari sisi aplikasi perlu ditambahkan agar aplikasi dapat menampilkan teks dari dokumen sesuai dengan format teks aslinya.

DAFTAR PUSTAKA

[1] A. I. Fahma, I. Cholissodin, and R. S. Perdana, "Identifikasi Kesalahan Penulisan Kata (Typographical Error) pada Dokumen Berbahasa Indonesia Menggunakan Metode N-gram dan Levenshtein Distance," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 2, no. 1, pp. 53–62, 2018.

[2] Murtiningsih S.Pd, "Kesalahan Berbahasa Indonesia Mahasiswa S-1 PGSD STIKIP Nuuwar Fak-fak," *J. Penelit. Ilmu Pendidik.*, vol. 6, no. 1, pp. 74–82, 2013.

[3] M. Javed *et al.*, "A Study of Students' Assessment in Writing Skills of the English Language," *Int. J. Instr.*, vol. 6, no. 2, pp. 129–144, 2013.

[4] Y. Rochmawati and R. Kusumaningrum, "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks," *J. Buana Inform.*, vol. 7, no. 2, pp. 125–134, 2016.

[5] K. M. Suryaningrum and A. T, "Pengkoreksian dan Suggestion Word pada Keyword Menggunakan Algoritma Jaro-Winkler," *J. Teknol. Informasi-AITI*, vol. 13, no. 2, pp. 169–181, 2016.

[6] J. Frando, I. Ruslianto, R. Hidayati, J. Rekayasa, and S. Komputer, "Penerapan Jaro Winkler Distance dalam Aplikasi Pengoreksi Kesalahan Penulisan Bahasa Indonesia Berbasis Web," *Coding J. Komput. dan Apl.*, vol. 07, no. 03, pp. 44–53, 2019.

[7] T. Maghfira, I. Cholissodin, and A. Widodo, "Deteksi Kesalahan Ejaan dan Penentuan Rekomendasi Koreksi Kata yang Tepat Pada Dokumen Jurnal JTIIK Menggunakan Dictionary Lookup dan Damerau-Levenshtein Distance," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 6, pp. 498–506, 2017.

[8] W. W. A. Umboh, S. R. Sentinuwo, and A. M. Sambul, "Rancang Bangun Aplikasi Deteksi Kesalahan Penulisan Naskah Dokumen Skripsi," *E-Journal Tek. Inform.*, vol. 11, no. 1, 2017.

[9] C. I. Ratnasari, S. Kusumadewi, and L. Rosita, "A Non-Word Error Spell Checker for Patient Complaints in Bahasa Indonesia," *Int. J. Inf. Technol. Comput. Sci. Open Source*, vol. 1, no. 1, pp. 18–21, 2017.

[10] M. M. Yulianto, R. Arifudin, and A. Alamsyah, "Autocomplete and Spell Checking Levenshtein Distance Algorithm To Getting Text Suggest Error Data Searching In Library," *Sci. J. Informatics*, vol. 5, no. 1, p. 75, 2018.

[11] P. H. Hema and C. Sunitha, "Spell Checker for Non Word Error Detection : Survey," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 5, no. 3, pp. 360–363, 2015.

[12] A. Prasetyo, W. M. Baihaqi, and I. S. Had, "Algoritma Jaro-Winkler Distance: Fitur Autocorrect dan Spelling Suggestion pada Penulisan Naskah Bahasa Indonesia di BMS TV," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 5, no. 4, p. 435, 2018.

[13] H. Gueddah, A. Yousfi, and M. Belkasmi, "The filtered combination of the weighted edit distance and the Jaro-Winkler distance to improve spellchecking Arabic texts," *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA*, vol. 2016-July, no. May 2019, 2016.

[14] Mutammimah, H. Sujaini, and R. D. Nyoto, "Analisis Perbandingan Metode Spelling Corrector Peter Norvig dan Spelling Checker BK-Trees pada Kata Berbahasa Indonesia," *J. Sist. dan Teknol. Inf.*, vol. 5, no. 1, pp. 1–5, 2016.

[15] R. Gadde and J. Kelly, "Using Multiprocessing to Make Python Code Faster," *Medium*, 2018. [Online]. Available: https://medium.com/@urban_institute/using-multiprocessing-to-make-python-code-faster-23ea5ef996ba. [Accessed: 28-Jul-2019].