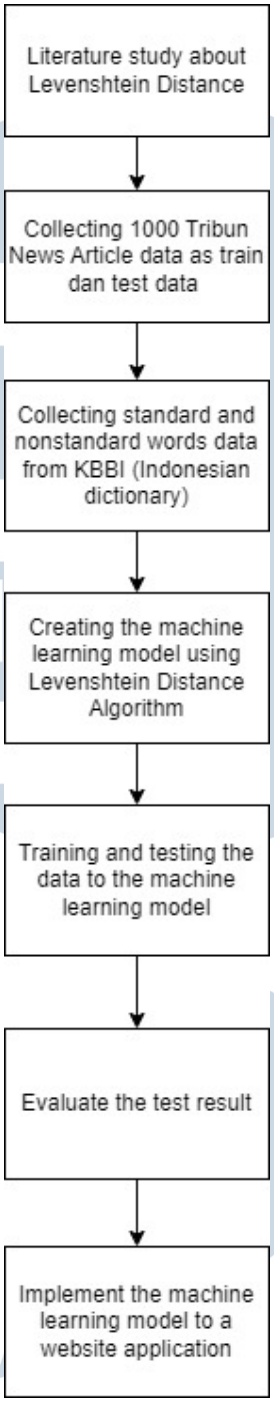
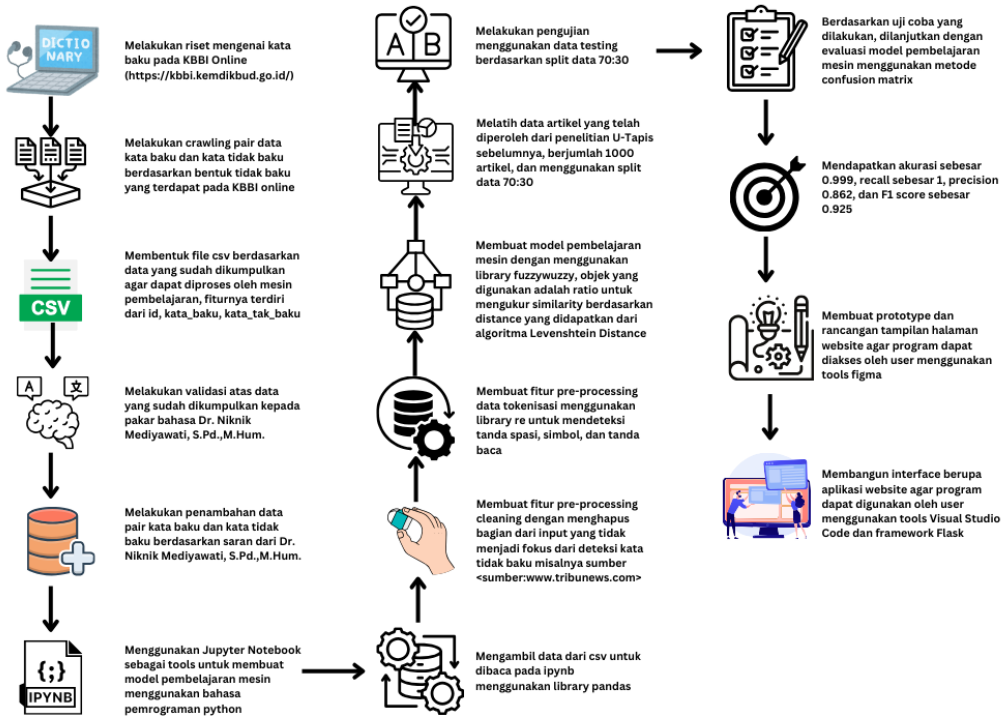
BAB 3
METODOLOGI PENELITIAN

Metode penelitian modul deteksi dan perbaikan kesalahan kata baku terdiri dari beberapa proses. Pertama, dilakukan studi literatur mengenai algoritma Levenshtein Distance untuk memperoleh pemahaman yang mendalam mengenai konsep dan aplikasi algoritma tersebut dalam pemrosesan teks. Kedua, dilakukan pengumpulan data berupa 1000 artikel sebagai data latih dan uji untuk melatih dan menguji model yang akan dibangun. Ketiga, data kata tidak baku dan kata baku dikumpulkan dari sumber KBBI sebagai referensi utama. Keempat, dilakukan pembuatan model Machine Learning berdasarkan data latih yang telah terkumpul. Kelima, model yang telah dibangun diuji coba menggunakan data latih dan data uji. Selanjutnya, hasil uji coba dievaluasi untuk menilai kinerja model yang telah dikembangkan. Setelah itu, model Machine Learning diimplementasikan ke dalam aplikasi website agar dapat diakses secara praktis. Terakhir, dilakukan penyusunan laporan hasil penelitian skripsi sebagai dokumentasi dan penyebaran hasil penelitian yang telah dilakukan. Bagan metode penelitian dapat dilihat pada gambar 3.1 dan 3.2.



Gambar 3.1. Diagram metode penelitian



Gambar 3.2. Diagram End to End Process

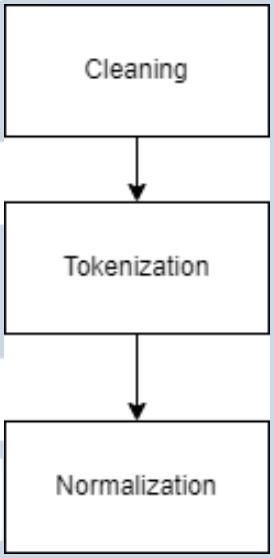
3.1 Studi Literatur

Dalam penelitian, studi literatur dilakukan guna menelaah literatur-literatur yang menunjang penelitian sebagai dasar atau landasan suatu penelitian dilaksanakan. Studi literatur yang dilakukan berkaitan dengan topik kata baku, tata cara penulisan bahasa Indonesia, *similarity variable* antara dua *string*, dan algoritma Levenshtein Distance.

3.2 Pengumpulan Data

Data yang dikumpulkan dibagi menjadi dua kategori, pertama, data kata baku dan kata tidak baku dikumpulkan secara mandiri sebanyak 721 baris data yang ditemukan di Kamus Besar Bahasa Indonesia, dan kedua, data artikel Tribun News, yang berjumlah 1000 artikel, digunakan sebagai data untuk pelatihan dan pengujian.

3.3 *Pre-processing* Teks Berita Daring



Gambar 3.3. Proses *pre-processing* input teks berita

Terdapat beberapa langkah pra-pemrosesan teks berita daring untuk model pendeteksian kata tidak baku seperti yang digambarkan dalam diagram 3.3. Langkah-langkah tersebut memegang peranan krusial dalam memastikan kualitas dan keberhasilan proses analisis teks. Proses pertama adalah tokenisasi, di mana teks berita dibagi menjadi token-token, seperti kata-kata dan tanda baca, untuk memfasilitasi analisis lebih lanjut. Setelah itu, langkah normalisasi dilakukan untuk memastikan konsistensi dalam representasi teks, sering kali dengan mengubah semua huruf menjadi huruf kecil agar tidak terjadi ambiguitas dalam pengenalan kata. Kemudian, proses pembersihan bertujuan untuk menghilangkan karakter-karakter tidak relevan atau tanda baca yang tidak diperlukan dalam analisis, sehingga memastikan teks yang diolah menjadi lebih bersih dan mudah diinterpretasikan oleh model pendeteksian kata tidak baku. Melalui serangkaian langkah ini, teks berita daring dapat disiapkan dengan baik sebelum masuk ke tahap analisis lebih lanjut.

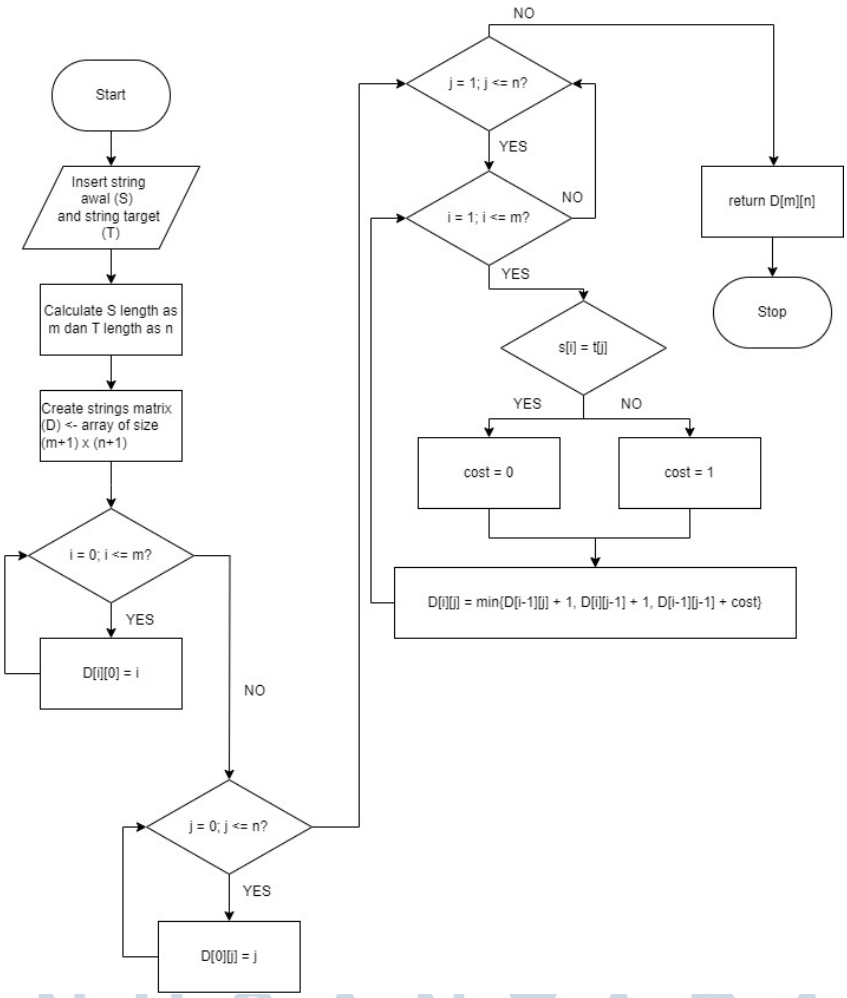
3.4 Algoritma Levenshtein Distance

Algoritma Levenshtein distance memiliki sistematika yang dapat digambarkan dengan dua jenis pemetaan, pseudocode dan diagram alir.

Algorithm 1 Levenshtein Distance Pseudocode

```
1:  procedure LEVENSHTEINDISTANCE(s, t)
2:      m ← length(s)
3:      n ← length(t)
4:      D ← array of size (m+1) × (n+1)
5:      for i ← 0 to m do
6:          D[i][0] ← i
7:      end for
8:      for j ← 0 to n do
9:          D[0][j] ← j
10:     end for
11:     for j ← 1 to n do
12:         for i ← 1 to m do
13:             if s[i] = t[j] then
14:                 cost ← 0
15:             else
16:                 cost ← 1
17:             end if
18:             D[i][j] ← min{D[i−1][j]+1, D[i][j−1]+1, D[i−1][j−1]+cost}
19:         end for
20:     end for
21:     return D[m][n]
22: end procedure
```

Dalam pseudocode 1, algoritma Levenshtein Distance digunakan untuk menghitung jarak edit antara dua string. Variabel m pada pseudocode menampung panjang dari *string* s (*input*) sedangkan n menampung panjang dari *string* t. Algoritma ini menerima dua string sebagai *input* (s dan t) dan mengembalikan nilai jarak *edit* antara kedua *string*. Proses utamanya terjadi dalam dua *loop* bersarang, di mana algoritma membandingkan setiap karakter dari dua string dan menghitung biaya operasi yang diperlukan untuk membuat kedua string menjadi sama. Setelah loop selesai, nilai jarak *edit* Algoritma Levenshtein Distance dapat digunakan untuk banyak hal, seperti membandingkan *string*, melakukan pengecekan ejaan, dan mencari kesamaan dalam basis data.



Gambar 3.4. Diagram alir Algoritma Levenshtein Distance

Flowchart Levenshtein Distance pada gambar 3.4 memberikan representasi visual mengenai algoritma Levenshtein Distance. Algoritma ini dimulai dengan

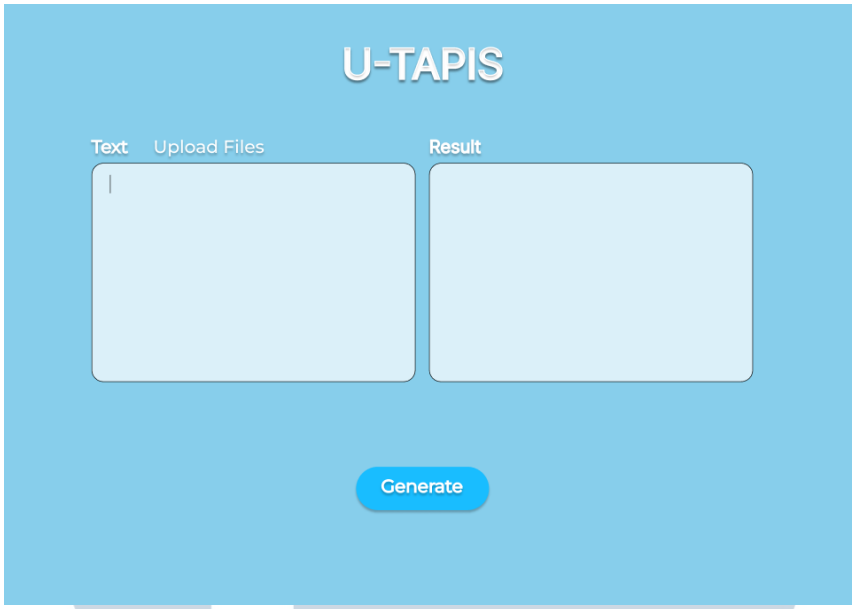
inisialisasi matriks dua dimensi dan iterasi melalui setiap karakter dari kedua string yang akan dibandingkan. Setiap iterasi memeriksa apakah karakter yang dibandingkan sama atau tidak, dan menghitung operasi penghapusan, penambahan, atau substitusi yang diperlukan. Operasi-operasi ini disimpan dalam matriks sebagai nilai jarak edit antara dua string pada setiap titik, dan nilai jarak edit total dihitung berdasarkan nilai terakhir dalam matriks. Flowchart ini memberikan visualisasi yang jelas tentang langkah-langkah algoritma Levenshtein Distance dalam menentukan seberapa berbedanya dua string, memudahkan pemahaman dan implementasi algoritma tersebut.

3.5 Implementasi Algoritma

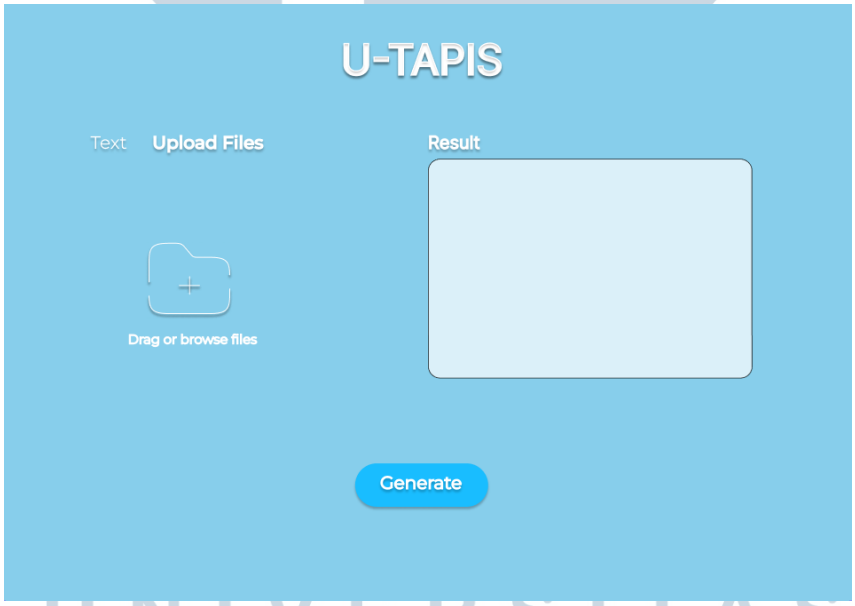
Setelah perancangan algoritma menggunakan diagram alir dilakukan, algoritma akan diimplementasikan dalam bahasa python untuk implementasi dan algoritma dan pembelajaran mesin.

3.6 Pengembangan Situs Web

Pengembangan situs web merupakan langkah selanjutnya setelah implementasi algoritma. Setelah program dan algoritma yang diuji rampung, maka pembangunan *user interface* perlu dilakukan agar program dapat digunakan oleh *user*. Situs web memiliki *user interface* yang sederhana dimana *user* hanya perlu memasukkan artikel atau kalimat untuk diproses dan program akan memberikan *output* pendeteksian, saran penggatian, dan penggantian kata tidak baku yang terdeteksi menjadi kata baku.



Gambar 3.5. Rancangan tampak halaman web untuk *input* teks



Gambar 3.6. Rancangan tampak halaman web untuk unggah berkas

Gambar 3.5 dan 3.6 menunjukkan rancangan tampak halaman web untuk *input* teks dan unggah berkas. Aplikasi web dirancang untuk dapat mengambil masukkan dengan dua jenis *input* yakni teks *string* dan unggah berkas dengan format txt. Rancangan ini dapat diakses melalui tautan https://www.figma.com/design/kgS11ChV4iOQhMWJHtymg0/Design-U-TAPIS-kata-baku?node-id=9-18.

3.7 Uji Coba dan Evaluasi

Langkah penting untuk menilai efisiensi dan efektivitas sebuah program adalah evaluasi. Evaluasi yang efektif membantu menentukan kekuatan dan kelemahan program, memberikan wawasan untuk perbaikan, dan memastikan bahwa program mencapai hasil yang diinginkan. Sebuah evaluasi menyeluruh dapat memastikan bahwa program memberikan manfaat terbaik bagi semua pemangku kepentingan.

Tabel 3.1. Tabel Confusion Matrix

	Predicted Class	
	Positive	**Negative**
Actual Positive	**TP** (True Positive)	**FN** (False Negative)
Actual Negative	**FP** (False Positive)	**TN** (True Negative)

Sumber: [32]

Uji coba dan evaluasi dilakukan guna meninjau apakah modul yang dibangun dapat memenuhi ekspektasi dari pengguna maupun peneliti. Uji coba dan evaluasi turut menjadi faktor penentu apakah algoritma Levenshtein Distance mampu menyelesaikan masalah dari penulisan kata tidak baku yang sering terjadi dalam penulisan artikel. Uji coba dilaksanakan dengan menguji sampel teks dari artikel jurnalistik Tribun News dan dilihat apakah hasil yang dikeluarkan sudah memenuhi ekspektasi yang seharusnya. Evaluasi dilakukan dengan menghitung apakah akurasi yang didapatkan program sudah cukup baik atau masih diperlukan pengembangan di masa mendatang.

$$\text{Akurasi} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{3.1}$$

- TP adalah jumlah True Positif.
- TN adalah jumlah True Negatif.
- FP adalah jumlah False Positif.
- FN adalah jumlah False Negatif.

Penghitungan akurasi model dan algoritma menggunakan rumus 3.1. Rumus akurasi digunakan untuk mengukur seberapa baik model atau sistem klasifikasi dalam memprediksi data dengan benar. Ini dihitung dengan membagi jumlah

prediksi yang benar (True Positif dan True Negatif) dengan jumlah total data yang dievaluasi. Akurasi memberikan gambaran tentang seberapa sering model memberikan prediksi yang benar secara keseluruhan. Semakin tinggi nilai akurasi, semakin baik kinerja modelnya.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.2}$$

- *precision precision* adalah rasio true positive (TP) dengan total hasil prediksi positif.
- *recall recall* adalah rasio true positive (TP) dengan total data yang seharusnya positif.

Penghitungan F1 Score dilakukan dengan menggunakan rumus 3.2. F1 Score merupakan metrik evaluasi yang menggabungkan *precision* dan *recall*, memberikan gambaran tentang keseimbangan antara kedua metrik tersebut. F1 Score sangat berguna dalam kasus di mana terdapat ketidakseimbangan kelas (imbalance class) dalam data. Semakin tinggi nilai F1 Score, semakin baik performa model dalam melakukan klasifikasi.