BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian mengenai deteksi kemiripan teks telah banyak dilakukan dengan berbagai pendekatan. Menurut Sari dan Wibisono (2021), implementasi algoritma Jaro–Winkler dalam sistem deteksi judul tugas akhir memberikan akurasi hingga 85% pada threshold 0.85. Sementara itu, Hidayat et al. (2020) melaporkan bahwa kombinasi Jaro–Winkler dan Levenshtein memberikan hasil yang lebih stabil dalam mendeteksi kemiripan dokumen akademik.

Penelitian oleh Rahman et al. (2022) mengembangkan sistem deteksi judul skripsi berbasis hybrid string matching yang menggabungkan tokenization dan edit distance. Hasilnya menunjukkan bahwa optimasi threshold secara sistematis dapat meningkatkan nilai F1-score sebesar –10% dibanding threshold default.

Selain itu, studi yang dilakukan oleh Maharani et al. (2020) pada konteks record linkage menunjukkan bahwa pemilihan threshold optimal dapat bervariasi tergantung karakteristik data dan panjang string. Penelitian ini menjadi dasar penting dalam pengembangan metode adaptif untuk penyaringan teks yang mirip.

Secara umum, penelitian-penelitian terdahulu menunjukkan bahwa meskipun algoritma string similarity seperti Jaro–Winkler dan Levenshtein memiliki keunggulan masing-masing, performanya sangat bergantung pada nilai threshold yang digunakan.Oleh karena itu, diperlukan penelitian lanjutan untuk menemukan

nilai threshold optimal yang sesuai dengan konteks penyaringan judul skripsi di lingkungan akademik.

2.2 Kajian Teori

2.2.1 Konsep String Similarity

String similarity merupakan metode untuk mengukur tingkat kemiripan antara dua atau lebih string teks. Konsep ini penting dalam berbagai bidang seperti information retrieval, record linkage, dan deteksi plagiarisme (Christen, 2012). String similarity juga merupakan sebuah proses kuantitatif untuk mengukur tingkat kemiripan antara dua string berdasarkan perhitungan karakter atau pola tertentu, sehingga dua teks yang tidak identik dapat tetap dinilai mirip dalam konteks komputasi.

Menurut Yu, Li, Deng, dan Feng (2016), string similarity memungkinkan sistem mendeteksi kemiripan teks dengan membandingkan struktur dan perbedaan karakter untuk memperoleh nilai kesamaan yang terukur. Sementara itu, Wang (2020) menekankan bahwa metode string similarity dapat menggunakan teknik berbasis jarak seperti Levenshtein Distance maupun teknik berbasis token dan statistika untuk menghasilkan skor kemiripan antar string. Dengan demikian, string similarity berfungsi sebagai dasar dalam berbagai aplikasi yang menuntut pencocokan teks yang toleran terhadap kesalahan, variasi penulisan, maupun typo. Tujuan utama dari pengukuran kemiripan string adalah untuk menentukan seberapa besar dua teks memiliki kesamaan pola atau struktur karakter. Pengukuran ini umumnya dibagi menjadi dua pendekatan utama, yaitu *edit-based similarity*

dan *token-based similarity* (Navarro, 2001).

Pendekatan edit-based berfokus pada jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lain, seperti dalam algoritma Levenshtein Distance. Sementara itu, pendekatan token-based mengukur kemiripan berdasarkan kesamaan kata atau urutan token, seperti pada algoritma Jaro–Winkler yang memberi bobot lebih pada awalan yang sama (Winkler, 1999). Pemilihan algoritma yang tepat bergantung pada konteks dan kebutuhan sistem yang dikembangkan.

2.2.2 Algoritma Levenshtein Distance

Algoritma Levenshtein Distance diperkenalkan oleh Vladimir Levenshtein pada tahun 1966 untuk mengukur jarak antara dua string berdasarkan jumlah operasi edit (penambahan, penghapusan, atau penggantian karakter) yang diperlukan untuk mengubah satu string menjadi string lainnya (Levenshtein, 1966). Nilai jarak yang lebih kecil menunjukkan tingkat kemiripan yang lebih tinggi.

Levenshtein Distance juga merupakan algoritma perhitungan string similarity yang mengukur jarak edit minimum untuk mengubah satu string menjadi string lainnya melalui tiga operasi dasar, yaitu penyisipan (insertion), penghapusan (deletion), dan penggantian karakter (substitution). Algoritma ini banyak digunakan dalam berbagai bidang Teknik Informatika, termasuk approximate matching, koreksi ejaan, text mining, natural language processing, dan data cleaning karena kemampuannya dalam mendeteksi kemiripan kata meskipun terdapat kesalahan penulisan maupun variasi karakter.

Menurut Khalid (2022), Levenshtein Distance secara formal didefinisikan sebagai jumlah minimum modifikasi karakter yang diperlukan untuk mentransformasikan sebuahstring menjadi string pembanding. Selain itu, Zhang (2023) menegaskan bahwa Levenshtein Distance tetap menjadi metode jarak berbasis karakter yang paling fleksibel dan efektif untuk menangani variasi teks pada aplikasi modern yang memerlukan proses pencocokan string toleran-kesalahan.

Levenshtein Distance banyak digunakan dalam bidang *spell checking*, *optical character recognition*, dan deteksi plagiarisme karena sifatnya yang sederhana namun efektif (Yujian & Bo, 2007). Meskipun demikian, algoritma ini tidak mempertimbangkan kesamaan posisi karakter di awal string, sehingga kadang kurang akurat dalam kasus teks pendek atau mirip sebagian (Gomaa & Fahmy, 2013).

Dalam konteks sistem penyaringan judul skripsi, algoritma Levenshtein digunakan untuk mengidentifikasi seberapa besar perubahan yang terjadi antar judul, misalnya antara ―Analisis Sistem Informasi Akademik‖ dan ―Analisis Sistem Akademik Berbasis Web.‖ Hasil pengukuran jarak kemudian dikonversi menjadi skor kemiripan antara 0 hingga 1 dengan menggunakan transformasi invers (Rahman et al., 2022).

Pada algoritma Levenshtein Distance terdapat tiga jenis operasi utama yang digunakan dalam proses perhitungan, yaitu:

2.2.2.1 *Insertion* (penambahan karakter)

2.2.2.2 *Deletion* (penghapusan karakter),

2.2.2.3 *Substitution* (penggantian karakter).

2.2.2.4 Setiap operasi memiliki bobot nilai sebesar 1.

Dengan demikian, total distance diperoleh dari jumlah minimum seluruh operasi yang diperlukan untuk mengubah string pertama menjadi string kedua.

Secara matematis, algoritma Levenshtein Distance dihitung menggunakan pendekatan rekursif atau *dynamic programming*. Rumus dasar Levenshtein Distance adalah:

$$max(i, j) \text{if } min(i, j) = 0$$

$$lev_{a,b}(i, j) = \begin{cases} min\{ \\ lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + cost \end{cases}$$

Keterangan:

- $lev_{a,b}(i, j)$= nilai Levenshtein Distance antara string $a$dan b
- i= panjang string pertama
- j= panjang string kedua
- $cost$= nilai biaya operasi substitusi
- bernilai 0 jika karakter sama
- bernilai 1 jika karakter berbeda

Rumus tersebut bekerja dengan membandingkan setiap karakter pada kedua string secara bertahap hingga diperoleh jumlah operasi minimum yang diperlukan. Metode *dynamic programming* digunakan agar proses perhitungan lebih efisien dengan menyimpan hasil perhitungan sebelumnya ke dalam matriks.

Cara kerja algoritma Levenshtein Distance dimulai dengan membentuk matriks berdasarkan panjang kedua string yang dibandingkan. Baris matriks merepresentasikan

string pertama, sedangkan kolom matriks

Keterangan:

- $lev_{a,b}(i,j)$= nilai Levenshtein Distance antara string $a$dan b
- i= panjang string pertama
- j= panjang string kedua
- $cost$= nilai biaya operasi substitusi
- bernilai 0 jika karakter sama
- bernilai 1 jika karakter berbeda

Rumus tersebut bekerja dengan membandingkan setiap karakter pada kedua string secara bertahap hingga diperoleh jumlah operasi minimum yang diperlukan. Metode *dynamic programming* digunakan agar proses perhitungan lebih efisien dengan menyimpan hasil perhitungan sebelumnya ke dalam matriks.

Cara kerja algoritma Levenshtein Distance dimulai dengan membentuk matriks berdasarkan panjang kedua string yang dibandingkan. Baris matriks merepresentasikan string pertama, sedangkan kolom matriks merepresentasikan string kedua. Setiap sel dalam matriks berisi nilai minimum operasi edit yang diperlukan untuk mencapai kesamaan karakter pada posisi tertentu. Dalam matriks berisi nilai minimum operasi edit yang diperlukan untuk mencapai kesamaan karakter pada posisi tertentu.

Sebagai contoh, misalkan terdapat dua string:

- String 1 = ―KITTEN‖
- String 2 = ―SITTING‖

Untuk mengubah kata —KITTEN‖ menjadi —SITTING‖, diperlukan beberapa operasi edit sebagai berikut:

1. Mengganti huruf K menjadi S (*substitution*)
2. Mengganti huruf E menjadi I (*substitution*)
3. Menambahkan huruf G di akhir kata (*insertion*)

Dengan demikian total operasi edit yang diperlukan adalah 3, sehingga nilai Levenshtein Distance antara kedua string tersebut adalah:

$$Levenshtein\ Distance(KITTEN, SITTING) = 3$$

Nilai tersebut menunjukkan bahwa kedua string memiliki tingkat kemiripan yang cukup tinggi karena hanya memerlukan tiga operasi perubahan. Dalam implementasinya, algoritma Levenshtein Distance sering digunakan untuk mendeteksi kesalahan pengetikan pada dokumen maupun pencarian teks. Sebagai contoh, apabila pengguna mengetik kata — algorima‖, sistem dapat membandingkannya dengan kata — algoritma‖ menggunakan Levenshtein Distance. Karena hanya terdapat satu karakter yang hilang, maka distance yang dihasilkan kecil sehingga sistem dapat menyimpulkan bahwa kedua kata tersebut memiliki kemiripan tinggi. (Sari et al., 2023).

2.2.3 Algoritma Jaro–Winkler

Algoritma Jaro–Winkler dikembangkan oleh Matthew Winkler

berdasarkan algoritma Jaro (Jaro, 1989; Winkler, 1999). Algoritma ini digunakan untuk menghitung kesamaan dua string dengan memperhatikan jumlah karakter yang cocok dan posisi relatifnya. Nilai kesamaan Jaro dihitung berdasarkan karakter yang cocok (*matching characters*) dan jumlah transpositions atau pertukaran posisi karakter. Jaro–Winkler Distance merupakan algoritma string similarity yang dirancang untuk mengukur kemiripan teks dengan menitikberatkan pada jumlah karakter yang sama, urutan kemunculan karakter, serta panjang prefix yang identik di awal string. Algoritma ini merupakan pengembangan dari Jaro Distance dengan penambahan prefix scale untuk memberikan bobot lebih tinggi pada karakter awal yang cocok, sehingga cocok diterapkan pada pencocokan nama, pencarian entitas, dan record linkage. Menurut Assaf dan Saad (2022), Jaro–Winkler mampu memberikan skor kemiripan lebih akurat pada data teks pendek karena mempertimbangkan matching characters, transpositions, dan prefix boosting. Selain itu, Khadivi et al. (2021) menegaskan bahwa algoritma ini lebih efektif dibandingkan metode jarak lainnya ketika digunakan pada domain yang sensitif terhadap kesamaan karakter di awal string, seperti *entity resolution*, *data deduplication*, dan *identity matching* dalam sistem informasi modern.

Perbaikan yang dilakukan oleh Winkler memberikan bobot tambahan (prefix scale) terhadap kesamaan karakter di awal string, karena secara empiris awalan yang identik sering kali menandakan kesamaan semantik yang lebih tinggi (Cohen et al.,

2003). Nilai kemiripan Jaro–Winkler berkisar antara 0 (tidak mirip) hingga 1 (identik).

Dalam sistem penyaringan judul skripsi, algoritma Jaro–Winkler cenderung unggul dalam mendeteksi kemiripan antar judul dengan perbedaan kecil pada bagian tengah atau akhir string. Contohnya, antara ―Sistem Informasi Akademik Mahasiswa‖ dan ―Sistem Informasi Akademik Dosen,‖ algoritma ini akan memberikan nilai kemiripan yang cukup tinggi karena awalan kata ―Sistem Informasi Akademik‖ identik (Sari & Wibisono, 2021).

Secara umum, algoritma Jaro-Winkler bekerja dengan menghitung jumlah karakter yang cocok (*matching character*), jumlah transposisi karakter (*transposition*), serta tingkat kesamaan awalan kata pada dua string yang dibandingkan (Rahman et al., 2024). Nilai similarity yang dihasilkan berada pada rentang 0 hingga 1. Semakin mendekati nilai 1, maka tingkat kemiripan kedua string semakin tinggi, sedangkan nilai 0 menunjukkan bahwa kedua string tidak memiliki kemiripan sama sekali (GeeksforGeeks, 2024).

Tahapan pertama dalam algoritma Jaro-Winkler adalah menentukan karakter yang cocok berdasarkan batas pencarian tertentu (*matching window*).Batas pencarian tersebut dihitung menggunakan rumus. Pada rumus tersebut, $s_1$dan $s_2$merupakan string yang dibandingkan. Nilai *match distance* digunakan untuk menentukan seberapa jauh posisi karakter masih dapat dianggap cocok (GeeksforGeeks, 2024).

Setelah karakter yang cocok ditemukan, langkah berikutnya adalah menghitung jumlah transposisi. Transposisi terjadi apabila terdapat karakter yang sama tetapi posisinya tertukar. Jumlah transposisi kemudian digunakan dalam perhitungan nilai Jaro Similarity (Rahman et al., 2024). Nilai Jaro Similarity dihitung menggunakan rumus berikut:

$$J(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$$

Keterangan:

- $J(s_1, s_2)$= nilai Jaro Similarity
- m= jumlah karakter yang cocok
- t= jumlah transposisi
- $|s_1|$= panjang string pertama
- $|s_2|$= panjang string kedua

Rumus tersebut digunakan untuk menghitung tingkat kemiripan dasar berdasarkan jumlah karakter yang sama dan posisi karakter pada kedua string (GeeksforGeeks, 2024).

Selanjutnya, algoritma Jaro-Winkler memberikan bobot tambahan pada kesamaan awalan kata menggunakan rumus:

$$JW = J + (l \times p \times (1 - J))$$

Keterangan:

- JW= nilai Jaro-Winkler
- J= nilai Jaro Similarity
- l= panjang prefix yang sama pada awal string (maksimal 4 karakter)
- p= konstanta prefix scaling, biasanya bernilai 0,1

Penambahan nilai prefix dilakukan karena string yang memiliki awalan sama dianggap memiliki tingkat kemiripan yang lebih tinggi dibandingkan string dengan awalan berbeda (Statology, 2024). Sebagai contoh, apabila terdapat dua string yaitu ―MARTHA‖ dan ―MARHTA‖, maka diperoleh enam karakter yang cocok dengan satu transposisi karakter. Berdasarkan hasil tersebut diperoleh nilai Jaro sebesar:

$$JW = 0.944 + (3 \times 0.1 \times (1 - 0.944)) = 0.961$$

Hasil akhir menunjukkan nilai similarity sebesar 0,961 yang berarti kedua string memiliki tingkat kemiripan yang sangat tinggi (Wijaya & Putra, 2022).

Secara umum, algoritma Jaro-Winkler bekerja dengan menghitung jumlah karakter yang cocok (matching characters), jumlah transposisi karakter (transposition), serta tingkat kesamaan awalan kata antara dua string. Proses pertama yang dilakukan adalah menentukan karakter yang cocok pada kedua string berdasarkan batas pencarian tertentu (matching window). Batas pencarian tersebut diperoleh menggunakan rumus. Pada rumus tersebut, s_1 dan s_2 merupakan string yang dibandingkan, sedangkan nilai match distance digunakan untuk menentukan seberapa jauh posisi karakter masih dianggap cocok. Setelah karakter yang cocok ditemukan, langkah berikutnya adalah menghitung jumlah transposisi, yaitu karakter yang sebenarnya sama tetapi berada pada posisi yang berbeda. Nilai transposisi dihitung dengan membagi jumlah karakter yang tertukar dengan dua. Setelah jumlah karakter yang cocok dan jumlah transposisi diperoleh, maka nilai Jaro Similarity dapat dihitung menggunakan rumus:

$$J(s_1, s_2) = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$$

Keterangan:

- $J(s_1, s_2)$= nilai Jaro Similarity
- m= jumlah karakter yang cocok
- t= jumlah transposisi
- $|s_1|$= panjang string pertama$|s_2|$= panjang string kedua

Rumus tersebut menghasilkan nilai kemiripan dasar berdasarkan kesamaan karakter dan posisi karakter antar string. Namun, pada algoritma Jaro-Winkler dilakukan pengembangan lebih lanjut dengan memberikan bobot tambahan pada kesamaan awalan kata. Hal ini dilakukan karena string yang memilih

$$JW = J + (l \times p \times (1 - J))$$

Pada rumus tersebut:

- JW= nilai Jaro-Winkler
- J= nilai Jaro Similarity
- l= panjang prefix yang sama pada awal string (maksimal 4 karakter)
- p= konstanta prefix scaling, umumnya bernilai 0,1

Cara kerja algoritma Jaro-Winkler dapat dijelaskan melalui contoh perhitungan sederhana. Misalnya terdapat dua string, yaitu ―MARTHA‖ dan ―MARHTA‖. Kedua string tersebut memiliki enam karakter yang cocok, yaitu M, A, R, T, H, dan A. Namun terdapat satu transposisi karena posisi huruf T dan H tertukar. Dengan

demikian diperoleh nilai $m = 6$dan $t = 1$. Selanjutnya nilai Jaro dihitung menggunakan rumus:

$$J = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6}\right) = 0.944$$

Kemudian karena kedua string memiliki awalan yang sama, yaitu —MAR‖, maka panjang prefix adalah 3 karakter. Dengan nilai prefix scaling sebesar 0,1, maka perhitungan Jaro-Winkler menjadi:

$$JW = 0.944 + (3 \times 0.1 \times (1 - 0.944)) = 0.961$$

Hasil akhir menunjukkan nilai similarity sebesar 0,961 yang berarti kedua string memiliki tingkat kemiripan yang sangat tinggi. Konsep Threshold Similarity Threshold similarity adalah nilai ambang yang menentukan apakah dua string dianggap mirip atau tidak (Hidayat et al., 2020). Penentuan nilai threshold yang optimal. Faktor penting untuk menyeimbangkan antara *false positive* dan *false negative*. Nilai threshold yang terlalu tinggi dapat mengabaikan pasangan string yang sebenarnya mirip, sedangkan threshold yang terlalu rendah dapat menghasilkan terlalu banyak pasangan yang salah dikategorikan mirip (Maharani et al., 2020).

Similarity Threshold adalah nilai ambang yang digunakan untuk menentukan tingkat minimal kemiripan antar string agar dua data dianggap match atau relevan dalam proses pencocokan data. Nilai ambang ini bekerja di atas skor kemiripan

yang dihasilkan oleh algoritma string similarity, dengan rentang umum 0 sampai 1, di mana nilai yang lebih tinggi menunjukkan tingkat kemiripan yang semakin kuat. Azizah et al. (2021) menjelaskan bahwa similarity threshold berperan sebagai mekanisme filter keputusan untuk menentukan apakah dua teks memiliki kemiripan yang cukup signifikan sehingga layak diperlakukan sebagai pasangan data yang sama dalam proses *record linkage* atau *data matching*. Dalam konteks penerapan, penentuan threshold sangat bergantung pada domain data dan algoritma yang digunakan, karena setiap metode memiliki sensitivitas yang berbeda terhadap jarak karakter, pola token, maupun tingkat kesalahan penulisan. Yu, Zhang, dan Huang (2022) menekankan bahwa pemilihan nilai ambang yang terlalu rendah dapat meningkatkan *false positive*, yaitu data berbeda yang dianggap mirip, sementara ambang yang terlalu tinggi dapat menimbulkan *false negative*. Data yang seharusnya terhubung justru terabaikan. Oleh karena itu, proses penentuan nilai ambang tidak bisa bersifat sembarangan dan idealnya dilakukan melalui proses evaluasi, validasi, atau pengujian ulang terhadap dataset yang relevan.

Secara praktis, similarity threshold banyak digunakan dalam entity resolution, deduplication, information retrieval, plagiarism detection, hingga identity matching berbasis teks. Pada aplikasi yang sensitif seperti data deduplication atau identity verification, nilai ambang tinggi (≥ 0.85) sering diterapkan untuk meminimalkan kesalahan identitas (Assaf & Saad, 2022). Sebaliknya, pada pencarian teks umum atau approximate retrieval, nilai threshold yang lebih longgar (0.65–0.80) dapat diterapkan karena sistem lebih mengutamakan kelonggaran pencocokan dibanding

ketelitian absolut.

Dengan demikian, similarity threshold berfungsi sebagai parameterpengatur kualitas pencocokan dalam sistem berbasis teks agar proses identifikasi kemiripan menjadi lebih terukur, konsisten, dan dapat dikendalikan. Keberadaannya menjadi krusial karena tanpa batas ambang, sistem pencocokan data berpotensi menghasilkan keluaran yang tidak akurat dan sulit dipertanggungjawabkan secara komputasional. Pada praktik terbaik, pemilihan nilai ambang dilakukan secara adaptif sesuai karakteristik dataset, tujuan aplikasi, dan algoritma yang digunakan, untuk memastikan keseimbangan antara presisi dan recall dalam proses pencocokan. Proses optimasi threshold dilakukan dengan cara menguji beberapa nilai threshold (biasanya antara 0.70–0.95) dan mengevaluasi kinerja algoritma berdasarkan metrik seperti precision, recall, dan F1-score (Jurafsky & Martin, 2023). Pemilihan nilai terbaik dapat menggunakan metode statistik seperti grid search atau pendekatan berbasis Receiver Operating Characteristic (ROC) dan Precision–Recall Curve (Sokolova & Lapalme, 2009).

2.2.4 Evaluasi Kinerja Sistem Similarity

Evaluasi terhadap sistem string similarity biasanya menggunakan metrik performa seperti precision, recall, accuracy, dan F1-score (Powers, 2020). Precision mengukur sejauh mana hasil positif yang diberikan sistem benar-benar relevan. Recall mengukur sejauh mana sistem berhasil menemukan semua pasangan string yang seharusnya mirip. F1-score merupakan rata-rata harmonis antara precision dan recall untuk menilai keseimbangan kinerja algoritma. Dalam penelitian ini,

evaluasi kinerja algoritma Jaro–Winkler dan Levenshtein dilakukan dengan memvariasikan nilai threshold similarity untuk menemukan titik optimal berdasarkan nilai F1-score tertinggi.

2.3 Sistem Informasi

Sistem informasi merupakan kombinasi antara teknologi, manusia, prosedur, serta basis data yang saling terintegrasi untuk mengolah data menjadi informasi yang berguna dalam mendukung proses operasional dan pengambilan keputusan pada suatu organisasi (Martono, 2022). Sistem informasi berfungsi untuk mengumpulkan, mengelola, menyimpan, dan mendistribusikan informasi sehingga dapat membantu meningkatkan efektivitas dan efisiensi kerja dalam suatu instansi atau perusahaan (Aryani et al., 2022).

Menurut Maulana dan Cahyono (2022), sistem informasi adalah suatu sistem berbasis komputer yang digunakan untuk mengelola data menjadi informasi yang memiliki nilai guna bagi pengguna. Sistem informasi tidak hanya berfokus pada penggunaan perangkat lunak, tetapi juga melibatkan pengguna, prosedur kerja, perangkat keras, jaringan komunikasi, serta pengelolaan basis data agar proses pengolahan informasi dapat berjalan secara optimal. Dalam perkembangannya, sistem informasi banyak diterapkan pada berbagai bidang seperti pendidikan, kesehatan, pemerintahan, bisnis, hingga industri. Penerapan sistem informasi mampu membantu organisasi dalam mempercepat proses pengolahan data, meminimalkan kesalahan manusia (*human error*), meningkatkan akurasi

informasi, serta mempermudah penyimpanan dan pencarian data (Gunawan et al., 2022). Selain itu, penggunaan sistem informasi juga dapat mendukung proses pelayanan agar menjadi lebih efektif, efisien, dan terorganisir dengan baik (Andari et al., 2022).

Sistem informasi memiliki beberapa komponen utama yang saling berkaitan, yaitu perangkat keras (*hardware*), perangkat lunak (*software*), basis data (*database*), prosedur, jaringan komunikasi, dan pengguna (*brainware*). Seluruh komponen tersebut bekerja secara terpadu untuk menghasilkan informasi yang akurat, relevan, dan tepat waktu (Martono, 2022). Apabila salah satu komponen tidak berjalan dengan baik, maka proses pengolahan informasi dalam sistem juga dapat terganggu. Secara umum, sistem informasi memiliki tujuan untuk mendukung kegiatan operasional organisasi, membantu proses pengambilan keputusan, serta meningkatkan kualitas layanan dan produktivitas kerja (Aryani et al., 2022). Oleh karena itu, sistem informasi menjadi salah satu teknologi yang sangat penting dalam mendukung kebutuhan pengelolaan data dan informasi.