BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Pada bagian ini, penulis meninjau berbagai penelitian yang dapat menjadi dasar ilmiah dalam perancangan solusi terhadap permasalahan yang telah dijelaskan pada Bab I. Pemahaman terhadap konsep-konsep yang relevan serta metode yang telah digunakan pada penelitian sebelumnya diperlukan untuk mengembangkan arsitektur baru yang dapat memperoleh waktu muat halaman yang lebih cepat, menyediakan fitur pencarian yang lebih toleran terhadap variasi dan kesalahan input (*fault tolerance*), serta mengoptimalkan proses pencarian melalui penerapan Bloom Filter sebagai mekanisme *early rejection*.

2.1.1 *Features of Rest API Development for High-Load Systems* [9]

Penelitian berjudul "*Features of Rest API Development for High-Load Systems*" yang ditulis oleh Viktor Bogutskii, membahas prinsip-prinsip krusial dalam pengembangan REST API untuk sistem yang menangani beban kerja tinggi (*high-load systems*). Penelitian ini mengeksplorasi metode pengembangan yang bertujuan meningkatkan kinerja, mengurangi latensi, dan memastikan stabilitas sistem melalui integrasi berbagai pendekatan optimasi infrastruktur dan protokol HTTP. Dalam konteks penanganan data, penelitian ini secara spesifik menyoroti tantangan yang muncul saat memproses kumpulan data yang besar (*large datasets*). Penulis menegaskan bahwa untuk pemrosesan data dalam jumlah besar (ribuan atau lebih), penggunaan paginasi (*pagination*) dan penyaringan data (*data filtering*) adalah hal yang sangat esensial.

Metode standar tanpa pembatasan data dinilai kurang efisien dalam sistem beban tinggi karena dapat membebani sumber daya komputasi. Penelitian ini menempatkan *pagination* sejajar dengan teknik optimasi vital lainnya, seperti *caching* dan pemrosesan asinkron, sebagai alat

utama untuk meningkatkan kinerja REST API. Secara lebih mendalam, validitas penggunaan *pagination* sebagai solusi untuk masalah waktu muat (*load time*) dibuktikan melalui dampak teknisnya terhadap server. Penelitian menunjukkan bahwa penerapan *pagination* berfungsi untuk meminimalkan komputasi yang tidak perlu pada sisi server. Dengan mengurangi beban komputasi tersebut, kinerja sistem secara keseluruhan akan meningkat.

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

- Pengolahan *dataset* besar menuntut mekanisme *pagination* sebagai implementasi vital, di mana metode ini diklasifikasikan secara eksplisit sebagai alat utama (*methods and tools*) dalam pengembangan API untuk mengatasi inefisiensi metode standar tanpa pembatasan data.
- Validitas solusi dibuktikan melalui dampak teknis pada server, di mana implementasi *pagination* terkonfirmasi dapat meminimalkan komputasi yang tidak perlu (*minimizes unnecessary computations*), yang secara langsung berkontribusi pada penurunan beban kerja dan peningkatan kinerja sistem secara keseluruhan.

2.1.2 *Efficient Similarity Measures for Texts Matching* [10]

Penelitian berjudul "*Efficient Similarity Measures for Texts Matching*" oleh Akinwale dan Niewiadomski, mengkaji efektivitas algoritma berbasis *n-gram* dalam menangani pencarian teks yang rentan terhadap kesalahan ejaan (*typo*). Melalui studi komparatif terhadap berbagai metode pencarian seperti *Generalized n-gram* dan *Bigram*, penelitian ini membuktikan bahwa pendekatan *Trigram* memiliki keunggulan signifikan dalam mengelompokkan dan mengambil kata-kata relevan dari database terminologi medis maupun jawaban ujian siswa. Kemampuan untuk menangani data dengan variasi *typo* yang tinggi ini memberikan dasar empiris yang kuat untuk menempatkan *Trigram*

sebagai mekanisme seleksi awal (*candidate generator*), di mana algoritma ini bertugas mengidentifikasi potensi kecocokan dalam dataset besar secara cepat sebelum dilakukan analisis kesamaan yang lebih mendalam.

Validitas penggunaan *Trigram* sebagai metode penyaringan utama dibuktikan melalui hasil eksperimen yang mencatatkan nilai *Recall* tertinggi sebesar 0.923, jauh mengungguli metode kompetitor seperti *Bigram* (0.761) dan *Generalized n-gram* (0.505) yang dapat dilihat pada tabel 2.1. Nilai *Recall* yang superior ini menjamin bahwa algoritma mampu "memanggil kembali" hampir seluruh kandidat data yang relevan meskipun mengandung *typo*.

Tabel 2.1 Nilai *precision*, *recall*, dan *f-measure* untuk ketiga metode

Methods	*Gen-n gram*	*Bigram*	*Trigram*
Precision	0.408	0.453	0.944
Recall	0.505	0.761	0.923
F-measure	0.451	0.567	0.931

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

- Trigram memiliki nilai *Recall* tinggi (0.923) yang krusial untuk *Candidate Generation*. Tingginya nilai *Recall* membuktikan bahwa metode berbasis *Trigram* sangat efektif dalam menjaring semua kandidat data yang relevan (termasuk yang memiliki *typo*), sehingga menjamin tidak ada hasil pencarian yang valid yang terbuang saat dilakukan penyempitan data awal.

2.1.3 *A Comparison of Techniques for Name Matching* [11]

Penelitian berjudul "*A Comparison of Techniques for Name Matching*" oleh Peng et al., mengevaluasi kinerja lima algoritma *string matching* berbasis karakter yang populer, yaitu Levenshtein, Smith-Waterman, Jaro, Jaro-Winkler, dan *Q-Gram*. Studi ini menyimpulkan bahwa untuk kasus pencocokan nama (*name matching*),

tidak ada satu teknik yang absolut terbaik di segala situasi, namun Jaro-Winkler dan Jaro secara konsisten menunjukkan kinerja yang lebih unggul dibandingkan metode lainnya, terutama pada dataset yang memiliki tingkat kesalahan (*error rate*) tinggi. Keunggulan Jaro-Winkler terletak pada modifikasi algoritmanya yang memberikan bobot lebih (*weight boost*) tinggi pada kesamaan awalan (*prefix*), didasarkan pada studi empiris bahwa kesalahan penulisan jarang terjadi di bagian awal kata.

Validitas penggunaan Jaro-Winkler untuk mendukung metrik kecepatan waktu muat dan *tolerance level* dibuktikan secara signifikan melalui hasil eksperimen waktu eksekusi dan ketahanan terhadap *typo*. Dalam analisis waktu pemrosesan (*timing performance*), penelitian ini menemukan bahwa Jaro-Winkler membutuhkan waktu paling sedikit (*costs the least time*) dibandingkan keempat algoritma lainnya, menjadikannya metode yang paling efisien secara komputasi. Selain unggul dalam kecepatan, Jaro-Winkler juga terbukti memberikan akurasi terbaik pada dataset dengan tingkat kesalahan tinggi (*high error rate*), mengungguli algoritma standar seperti Levenshtein. Kombinasi antara efisiensi waktu eksekusi yang tinggi dan kemampuan menangani *typo* yang kompleks memberikan justifikasi kuat untuk menempatkan Jaro-Winkler sebagai algoritma penilaian akhir yang cepat dan presisi setelah proses penyaringan kandidat.

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

- Penelitian ini membuktikan bahwa Jaro-Winkler adalah algoritma tercepat (*costs the least time*) di antara metode yang diuji (Levenshtein, Smith-Waterman, *Q-Gram*), yang secara langsung mendukung target perbaikan metrik *load time*.

- Jaro-Winkler terbukti dapat bekerja dengan baik pada kondisi data dengan tingkat kesalahan tinggi (*high error rate*), menjamin bahwa

sistem tetap dapat mengenali nama target meskipun pengguna melakukan banyak kesalahan ketik (*typo*).

2.1.4 *Building Verified Neural Networks for Computer Systems with Ouroboros* [12]

Penelitian berjudul "*Building Verified Neural Networks for Computer Systems with Ouroboros*" oleh Wei et al., mendefinisikan Bloom Filter sebagai struktur data probabilistik yang digunakan secara luas dalam sistem komputer untuk menguji keanggotaan suatu elemen dalam himpunan yang telah ditentukan (*membership test*). Penelitian ini menjelaskan bahwa karakteristik utama Bloom Filter adalah kemampuannya untuk memastikan jika sebuah elemen tidak ada dalam himpunan (*negative lookup*), meskipun memiliki kemungkinan *false positive* (mengatakan "ada" pada elemen yang sebenarnya tidak ada). Dalam eksperimennya menggunakan dataset *Crimes in Boston* yang berisi 300.000 data lokasi, Bloom Filter digunakan sebagai baseline kinerja untuk memverifikasi keberadaan data secara efisien.

Efisiensi kinerja Bloom Filter ditunjukkan melalui data komparatif pada gambar 2.1, di mana performa Bloom Filter standar dibandingkan dengan pendekatan berbasis *Neural Network* (NN). Berdasarkan hasil pengujian, Bloom Filter standar mencatatkan waktu latensi (*latency*) yang sangat rendah, yaitu hanya 3μs per *request*, jauh lebih cepat dibandingkan pendekatan *classic neural network* yang membutuhkan waktu 84μs. Selain itu, Bloom Filter standar menunjukkan *throughput* yang sangat tinggi mencapai 333.000 hingga 345.500 permintaan per detik dengan ukuran memori yang kecil (325-382 KB). Data ini membuktikan bahwa Bloom Filter adalah solusi komputasi yang sangat ringan dan cepat, sehingga cocok untuk diterapkan sebagai lapisan optimasi guna memangkas waktu pemrosesan pada data yang tidak ditemukan.

application	model	throughput	latency (μs or ns)	size (KB)	avg/max err	FP%	accuracy
LearnedIndex	verified NN4Sys	9.8M	351 μs	47	245/906	–	–
	classic NN4Sys	9.8M	377 μs	47	239/905	–	–
	RMI (49KB)	24.5M	41 ns	49	0.7/8	–	–
	RMI (6KB)	20.4M	49 ns	6	1.2/20	–	–
	B-Tree	5.1M	195 ns	337	–	–	–
BloomFilter	verified NN4Sys	147.1K	85 μs	1965	–	2.2%	–
	classic NN4Sys	148.1K	84 μs	1965	–	56.2%	–
	Bloom Filter (1%)	333.0K	3 μs	382	–	1.0%	–
	Bloom Filter (2%)	345.5K	3 μs	325	–	2.0%	–
LatPredictor	verified NN4Sys	172.9K	77 μs	366	–	–	97.7%
	classic NN4Sys	173.2K	81 μs	366	–	–	96.3%
	LinnOS-binary	172.5K	76 μs	12	–	–	96.1%

Gambar 2.1 *Execution performance of baselines and verified neural networks*

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

- Bloom Filter didefinisikan sebagai struktur data probabilistik yang secara efektif menguji keanggotaan elemen (*membership test*) dan menangani kasus elemen yang tidak ada (*non-existing elements*) dengan probabilitas yang dapat dikontrol, seperti *false positive rate* sebesar 1-2%.

- Eksperimen membuktikan superioritas kecepatan Bloom Filter dengan latensi pemrosesan hanya 3μs, yang secara signifikan lebih cepat dibandingkan metode pemrosesan kompleks lainnya, menjadikannya ideal untuk mengurangi beban komputasi pada fitur pencarian.

2.1.5 *Detection of Text Similarity for Indication Plagiarism Using Winnowing Algorithm Based K-gram and Jaccard Coefficient* [13]

Penelitian berjudul "*Detection of Text Similarity for Indication Plagiarism Using Winnowing Algorithm Based K-gram and Jaccard Coefficient*" oleh Puspaningrum et al., membahas penerapan algoritma *Winnowing* untuk mendeteksi tingkat kesamaan teks secara efisien. Studi ini menjelaskan bahwa *Winnowing* bekerja dengan memecah teks menjadi serangkaian karakter (*k-gram*), mengubahnya menjadi nilai hash, dan kemudian memilih sebagian kecil dari nilai tersebut melalui proses *windowing* untuk membentuk "sidik jari" (*fingerprint*) dokumen. Dalam konteks komputasi, studi ini menegaskan bahwa metode *Winnowing*

dirancang untuk meningkatkan waktu pencarian (*improves the search time*) dengan akurasi yang lebih baik dalam proses deteksi dibandingkan metode konvensional yang mungkin memerlukan waktu komputasi relatif lama.

Secara teknis, validitas algoritma ini sebagai metode *similarity* diukur menggunakan koefisien Jaccard untuk membandingkan nilai *fingerprint* antar dokumen. Eksperimen menunjukkan bahwa algoritma ini sangat efektif menangani variasi teks, termasuk kasus penyalinan (*copy paste*) dan relokasi kata (*relocate words*). Penelitian juga menyajikan bukti numerikal terkait sensitivitas parameter algoritma yang dapat dilihat pada tabel 2.2, di mana penentuan nilai *k-gram* secara signifikan mempengaruhi hasil deteksi. Nilai *k-gram* terkecil (k=2) menghasilkan rata-rata persentase kesamaan tertinggi sebesar 40.023%, sedangkan *k-gram* yang lebih besar (k=10) menghasilkan rata-rata 11.229%. Data ini menjustifikasi bahwa dengan parameter yang tepat, *Winnowing* dapat diandalkan untuk mengukur tingkat kesamaan (*similarity level*) secara granular dan efisien.

Tabel 2.2 Hasil test *k-gram*

k-Gram	*Average Precentage Similarity* (%)
2	40,023
3	23,898
4	14,307
5	14,085
8	11.901
9	11.517
10	11.229

Beberapa poin penting yang dapat diambil oleh penulis adalah sebagai berikut:

- Penelitian menyatakan secara faktual bahwa metode *Winnowing* "meningkatkan waktu pencarian" (*improves the search time*) dan mampu melakukan proses deteksi dalam "waktu yang cukup cepat" (*fairly fast time*), yang menjadi dasar justifikasi efisiensi algoritma ini dalam sistem.

- Justifikasi penggunaan metode ini terletak pada konsep *fingerprinting*, di mana dokumen diringkas menjadi sekumpulan nilai hash kecil yang efisien untuk dicari kembali (*retrieval*) dalam dokumen besar, menggantikan pemindaian teks penuh yang lambat.

Berdasarkan solusi dari penelitian terdahulu yang diambil oleh penulis, berikut beberapa poin yang menjadi acuan penelitian yang dilakukan oleh penulis.

Tabel 2.3 Rangkuman Poin Acuan Penulis Pada Solusi Penelitian Terdahulu

Judul Penelitian (Tahun)	**Poin yang menjadi acuan penulis**
Features of Rest API Development for High-Load Systems (2025)	• Dalam penelitian ini, implementasi *pagination* adalah hal yang krusial untuk mengatasi inefisiensi metode pengiriman data tradisional. Hal ini relevan untuk merancang solusi pengiriman data dengan waktu muat (*load time*) yang lebih cepat.
Efficient Similarity Measures for Texts Matching (2015)	• Penggunaan *Trigam* sebagai *candidate generation* menjadi acuan penulis untuk membuat kandidat sebelum *similarity matching*. Penelitian membuktikan *Trigram* sangat efektif dalam menjaring semua kandidat data yang relevan (termasuk yang memiliki *typo*).

A Comparison of Techniques for Name Matching (2012)	● Penelitian ini membuktikan bahwa Jaro-winkler adalah algoritma tercepat dalam *name matching*. Algoritma ini relevan dalam merancang solusi fitur pencarian yang tahan terhadap kesalahan ketik (*typo*), namun tetap memiliki proses yang cepat.
Building Verified Neural Networks for Computer Systems with Ouroboros (2023)	● Bloom filter yang terbukti ampuh dalam mendeteksi *non-existing elements* menjadi acuan penulis dalam membuat fitur pencarian yang efisien dan cepat. Implementasi Bloom filter pada solusi fitur pencarian akan mengurangi beban komputasi yang tidak perlu.
Detection of Text Similarity for Indication Plagiarism Using Winnowing Algorithm Based K-gram and Jaccard Coefficient (2020)	● Penelitian ini membuktikan bahwa konsep *fingerprinting* pada algoritma *Winnowing* dapat mendeteksi *similarity* pada teks dengan proses yang lebih cepat dan akurat. Algoritma *Winnowing* menjadi acuan penulis untuk membuat solusi fitur pencarian yang cepat dan tahan akan kesalahan ejaan.

2.2 Tinjauan Teori

Pada bagian ini, akan dipaparkan teori-teori dan konsep fundamental yang menjadi landasan utama dalam penelitian, khususnya berkaitan dengan solusi yang ingin dipaparkan. Pembahasan disusun secara sistematis ke dalam beberapa kelompok utama. Pertama, mekanisme penyajian dan akses data yang berperan dalam mengendalikan volume data serta menjaga efisiensi pemrosesan pada sistem. Kedua, algoritma *string matching* yang digunakan

untuk mendukung pencarian data berbasis kemiripan (*similarity detection*). Ketiga, struktur data probabilistik berupa Bloom Filter yang dimanfaatkan sebagai mekanisme penyaring awal untuk mengurangi beban komputasi sebelum proses pencarian lanjutan dilakukan. Terakhir, dibahas metode evaluasi sistem yang digunakan dalam penelitian ini, meliputi perangkat lunak pengujian beban (*load testing*) Grafana k6, dan *ground truth* sebagai acuan evaluasi.

2.2.1 Mekanisme Penyajian dan Akses Data

Dalam pengembangan aplikasi yang menangani data dalam jumlah besar, mekanisme penyajian dan akses data menjadi faktor penting dalam menjaga performa dan responsivitas sistem. Mekanisme ini berperan dalam mengatur bagaimana data diminta, dibatasi, dan dikirimkan dari server ke klien secara efisien. Oleh karena itu, bagian ini membahas dua komponen pendukung utama, yaitu *pagination* sebagai teknik pembatasan volume data yang ditampilkan, serta REST API sebagai mekanisme komunikasi antara klien dan server dalam proses pertukaran data. Kedua pembahasan ini akan berperan dalam meningkatkan metrik *load time*.

2.2.1.1 *Pagination*

Pagination adalah sebuah proses teknis yang membagi konten atau dataset dalam jumlah besar menjadi segmen-segmen yang lebih kecil dan dapat dikelola, dengan tujuan utama mengatasi tantangan performa pada antarmuka pengguna berskala besar [14]. Teknik ini muncul sebagai solusi kritis untuk menangani aplikasi yang padat data, di mana menampilkan volume data yang berlebihan sekaligus dapat menyebabkan konsumsi memori yang tinggi dan waktu muat yang lambat. Dengan membatasi jumlah data yang disajikan pada satu waktu, *pagination* tidak hanya memastikan efisiensi sistem tetapi juga meningkatkan responsivitas antarmuka dan pengalaman navigasi bagi pengguna akhir (*end user*), mencegah terjadinya degradasi performa saat merender dataset yang ekstensif.

Dalam penerapannya, terdapat dua metode utama yang sering digunakan untuk membagi data:

- *Traditional offset-based* pagination, bekerja dengan mengambil rentang data yang tetap dan sederhana untuk diimplementasikan, namun metode ini sering mengalami masalah skalabilitas pada dataset yang sangat besar karena memerlukan kueri database yang berulang dan berpotensi tidak efisien.
- *Cursor-based* pagination, memanfaatkan pengenalan unik (*unique identifiers*) untuk melacak posisi data terakhir yang diakses. Metode ini dianggap lebih tangguh (*robust*), karena mampu menjaga konsistensi hasil navigasi dan mengurangi kemungkinan terjadinya duplikasi atau data yang terlewati saat pengguna berpindah halaman.

Efisiensi *pagination* tidak hanya bergantung pada metode pengambilan data, tetapi juga dikategorikan berdasarkan teknik optimasi di sisi server (*server-side*) dan sisi klien (*client-side*). Pada sisi server, integrasi pagination dengan strategi *caching* dapat mengoptimalkan waktu pengambilan data. Sementara itu, pada sisi klien, teknik seperti *lazy loading* dan *infinite scrolling* memungkinkan pengiriman konten secara mulus [14]. Dalam perancangan solusi yang diusulkan, mekanisme *Cursor-based Pagination* diterapkan untuk membatasi volume data yang dikirimkan dalam satu waktu, dengan tujuan utama mengoptimalkan performa sistem.

2.2.1.2 *Representational State Transfer Application Programming Interface* (REST API)

REST (*Representational State Transfer*) merupakan sebuah gaya arsitektur API (*Application Programming Interface*) yang menyediakan standar komunikasi antara klien dan server untuk aplikasi web melalui protokol HTTP [15]. Arsitektur ini dirancang

dengan tiga prinsip utama, yaitu *addressability*, *uniform interface*, dan *statelessness*, yang memungkinkannya diimplementasikan secara fleksibel tanpa terikat pada protokol transfer tertentu, meskipun umumnya beroperasi di atas HTTP. Dalam mekanisme kerjanya, REST mendefinisikan *endpoints* menggunakan struktur direktori melalui URI (*Uniform Resource Identifier*) untuk mengekstraksi data. Operasi dalam REST bekerja berdasarkan prinsip CRUD (*Create*, *Read*, *Update*, *Delete*), yang berkorespondensi langsung dengan fungsi-fungsi standar pada penyimpanan data persisten seperti SQL (INSERT, SELECT, UPDATE, dan DELETE). Pertukaran data antara klien dan server dalam arsitektur ini umumnya menggunakan format JSON (*Javascript Object Notation*) yang ringan, meskipun format XML juga dimungkinkan [15]. REST API merupakan gaya arsitektur komunikasi yang digunakan dalam pengembangan aplikasi Talent Discovery sebagai mekanisme pertukaran data antara klien dan server. Pemilihan REST API bertujuan untuk mempertahankan konsistensi dengan sistem yang telah ada, sehingga solusi yang diusulkan dapat diintegrasikan tanpa memerlukan perubahan signifikan pada struktur dasar sistem.

Gambar 2.2 Cara kerja REST API

2.2.2 Algoritma String Matching

Solusi yang diusulkan untuk mengatasi keterbatasan toleransi kesalahan pencarian eksisting *substring matching* adalah mekanisme pencarian yang toleran terhadap variasi input atau kesalahan ejaan (*fault tolerance*) yang lebih baik, dan mampu beroperasi dengan efisiensi tinggi. Untuk merealisasikan hal tersebut, diperlukan pemahaman mendalam

mengenai teori *Approximate String Matching* beserta algoritma pendukungnya seperti *Winnowing*, *Trigram* dan Jaro-Winkler. Pembahasan dari ketiga algoritma ini penting untuk membuat pencarian yang memiliki metrik *tolerance level* yang tinggi.

2.2.2.1 *Approximate String Matching*

Approximate string matching didefinisikan sebagai proses menemukan sebuah string *t* dalam himpunan string *T* yang memiliki kemiripan (*similarity*) dengan string pencarian *s*, di mana *t* mungkin tidak identik dengan *s* [16]. Teknik ini menjadi krusial dalam sistem informasi karena pencarian yang hanya mengandalkan pencocokan *substring* (*substring matching*) akan mengalami kegagalan jika terdapat variasi ejaan atau kesalahan penulisan pada data masukan. Dalam konteks interaksi pengguna, variasi string sering kali disebabkan oleh kesalahan pengetikan (*typing errors*). Penelitian menunjukkan bahwa jenis kesalahan yang paling umum terjadi meliputi substitusi satu huruf, penghilangan (*omission*), penyisipan (*insertion*), dan pembalikan (*reversal*) karakter yang bersebelahan. Bahkan, studi klasik oleh Damerau mencatat bahwa lebih dari 80% kesalahan pengetikan terdiri dari jenis kesalahan tunggal tersebut.

Dalam implementasinya pada dataset berskala besar, efisiensi algoritma *string matching* menjadi tantangan utama karena pencarian serial secara menyeluruh (*exhaustive serial searches*) membutuhkan akses data yang berbanding lurus dengan jumlah total data, yang tidak efisien untuk sistem dengan volume data besar. Oleh karena itu, strategi pencarian aproksimasi pada himpunan data besar (*large sets*) tidak hanya bergantung pada algoritma *string matching* itu sendiri, tetapi juga memerlukan strategi partisi atau pengindeksan untuk menghindari pemindaian seluruh database [16].

2.2.2.2 *Winnowing*

Algoritma *Winnowing* merupakan metode efisien yang digunakan untuk mendeteksi kesamaan antar dokumen dengan cara menghasilkan *fingerprint* dari sebuah teks. Algoritma ini dinilai sangat relevan untuk pencocokan string karena memenuhi tiga prasyarat utama dalam deteksi kemiripan, yaitu:

- *whitespace insensitivity*, pencarian tidak terpengaruh oleh spasi, tanda baca, atau huruf kapital.
- *noise suppression*, kemampuan mengabaikan kata-kata umum yang terlalu pendek atau tidak esensial.
- *position independence*, kemampuan mendeteksi kesamaan teks meskipun posisi atau urutan kalimatnya berbeda.

Proses kerja algoritma ini dimulai dengan tahap *text preprocessing* untuk membuang karakter yang tidak relevan, dilanjutkan dengan pembentukan *n-grams* (rangkaian potongan karakter sepanjang *n*). Setiap gram kemudian dikonversi menjadi nilai numerik menggunakan fungsi *rolling hash*, yang diformulasikan sebagai berikut:

$$H\,(c_1 \ldots c_k) = c_1\, x\, b^{k-1} + c_2\, x\, b^{k-2} + \ldots + c_k$$

Di mana *H* adalah nilai hash, *c* adalah nilai karakter, *b* adalah bilangan basis, dan *k* adalah panjang gram. Nilai-nilai hash ini kemudian dikelompokkan ke dalam *window* dengan ukuran tertentu, dan algoritma akan memilih nilai hash terkecil dari setiap *window* untuk disimpan sebagai *fingerprint* dokumen.

Dalam konteks optimasi akurasi pencarian, pemilihan nilai parameter seperti ukuran *n-gram* dan ukuran *window* memegang peranan krusial. Hasil pengujian menunjukkan bahwa nilai *n-gram* berbanding terbalik dengan persentase kemiripan, nilai *n-gram* yang lebih kecil cenderung menghasilkan persentase kemiripan

yang lebih tinggi (lebih sensitif), sedangkan nilai yang lebih besar mengurangi sensitivitas tersebut. Studi eksperimental menyimpulkan bahwa kombinasi parameter tertentu (seperti *N*=8 dan *Window*=6 dengan bilangan prima tertentu) dapat memberikan hasil deteksi yang paling efektif dan merepresentasikan kemiripan dokumen secara akurat [17]. Algoritma *Winnowing* akan diuji keefektifan nya dalam mendeteksi kemiripan teks, yang kemudian akan dikomparasikan kinerjanya dengan *substring matching*, dan kombinasi metode *Trigram* dan Jaro-Winkler guna menentukan pendekatan yang paling tahan terhadap toleransi kesalahan (*fault tolerance*), seperti kesalahan pengetikan (*typo*) atau variasi ejaan.

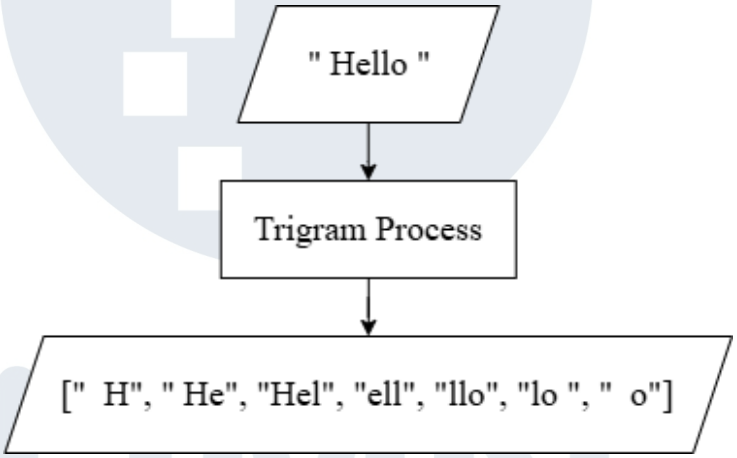
2.2.2.3 *Trigram*

Dalam pemrosesan teks dan pencarian string, analisis *n-gram* didefinisikan sebagai metode ekstraksi serangkaian *n* karakter yang berurutan dari sebuah kata. Studi spesifik mengenai *trigram* (*n*=3) menunjukkan bahwa teknik ini merupakan kompromi yang ideal antara *bigram* (dua karakter) yang kurang kuat dan *n-gram* tingkat tinggi yang membutuhkan sumber daya komputasi berlebihan. Proses pembentukan *trigram* dilakukan dengan teknik *sliding window* yang bergerak satu karakter setiap langkahnya. Prinsip dasar di balik penggunaan *trigram* untuk mendeteksi kemiripan atau kesalahan ejaan bergantung pada fakta bahwa hanya sebagian kecil dari total kemungkinan kombinasi *trigram* yang benar-benar muncul dalam teks yang valid, sehingga keberadaan *trigram* yang tidak biasa dapat menjadi indikator kuat adanya variasi atau kesalahan pada string input.

Selain kemampuannya dalam mendeteksi keberadaan kesalahan, analisis *trigram* memiliki keunggulan spesifik dalam melokalisasi posisi kesalahan (*error location*) dalam sebuah string, yang berbeda dengan metode pencarian kamus (*dictionary lookup*) yang hanya dapat menerima atau menolak kata secara

keseluruhan. Berdasarkan eksperimen pada dataset leksikal yang besar, algoritma berbasis *trigram* mampu menentukan posisi kesalahan secara akurat (dengan selisih tidak lebih dari satu karakter dari posisi sebenarnya) pada 94% kasus kesalahan ejaan.

Kemampuan untuk memecah string menjadi komponen-komponen kecil dan melacak posisi penyimpangan ini menjadikan *trigram* sebagai fitur yang sangat relevan untuk menghasilkan kandidat kemiripan yang presisi [18]. Oleh karena itu algoritma *Trigram* akan digunakan sebagai *candidate generator*, yang nantinya para kandidat ini diukur *similarity score* nya oleh Jaro-Winkler untuk menghasilkan output pencarian paling relevan dari input.



Gambar 2.3 Hasil proses Trigram

2.2.2.4 *Jaro-Winkler Distance*

Algoritma *Jaro-Winkler Distance* merupakan varian pengembangan dari metrik *Jaro Distance* yang dirancang untuk mengukur tingkat kesamaan antara dua entitas string [19]. Algoritma ini dinilai memiliki akurasi yang tinggi khususnya dalam pencocokan string yang relatif pendek (seperti nama entitas) dan bekerja dengan kompleksitas waktu kuadratik (*quadratic runtime complexity*), menjadikannya lebih efisien secara komputasi dibandingkan algoritma *edit distance* standar.

Output dari perhitungan ini adalah nilai ternormalisasi, di mana skor 0 menandakan tidak ada kesamaan, sedangkan skor 1 menandakan bahwa kedua string identik atau sama persis.

Secara fundamental, mekanisme dasar algoritma ini melibatkan tiga langkah utama, menghitung panjang string, menemukan jumlah karakter yang sama (*m*) dalam jangkauan pencocokan tertentu, dan menemukan jumlah transposisi (*t*) yaitu karakter yang sama namun berada dalam urutan yang tertukar. Nilai dasar Jaro Distance (d_j) dihitung menggunakan persamaan berikut:

$$d_j = \frac{1}{3} x \left(\frac{m}{s_1} + \frac{m}{s_2} + \frac{m-t}{m}\right)$$

Di mana s_1 dan s_2 merepresentasikan panjang dari masing-masing string yang dibandingkan.

Perbedaan signifikan Jaro-Winkler dibandingkan pendahulunya terletak pada penerapan bobot tambahan untuk kesamaan di awal string (*common prefix*). Algoritma ini menggunakan parameter prefix length (*l*), yaitu panjang karakter awal yang sama (maksimal 4 karakter), dan prefix scale (*p*) yang merupakan konstanta *scaling factor* (standar *p*=0,1). Penambahan ini didasarkan pada asumsi bahwa kesalahan pengetikan lebih jarang terjadi di awal kata. Dengan demikian, formulasi akhir Jaro-Winkler Distance (d_w) adalah:

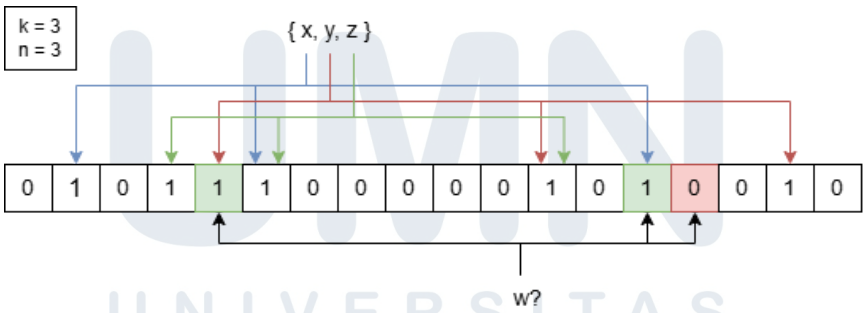
$$d_w = d_j + (l\, x\, p\, (1 - d_j))$$

Persamaan ini memberikan skor kemiripan yang lebih tinggi pada pasangan string yang memiliki kesamaan awalan, sehingga lebih sensitif terhadap pola penulisan yang variatif namun memiliki akar kata yang sama [19]. Algoritma Jaro-Winkler akan di uji keefektifan nya dalam melakukan *similarity scoring*, yang

nantinya akan dikomparasikan oleh algoritma *Winnowing*. Untuk menentukan pendekatan yang paling tahan terhadap toleransi kesalahan (*fault tolerance*), seperti kesalahan pengetikan (*typo*) atau variasi ejaan.

2.2.3 Struktur Data Probabilistik (Bloom Filter)

Bloom Filter didefinisikan sebagai struktur data probabilistik yang memiliki efisiensi ruang (*space efficient*) yang tinggi, yang dirancang untuk mendukung *set membership queries* atau menentukan apakah sebuah elemen merupakan anggota dari suatu himpunan tertentu [20]. Bloom Filter berperan langsung dalam mempercepat metrik *computation time* pada pencarian. Secara struktural, Bloom Filter terdiri dari sebuah array sepanjang *m* bit yang merepresentasikan himpunan *S* dari *n* elemen, di mana pada kondisi awal seluruh bit diatur bernilai nol. Mekanisme penyimpanan data melibatkan penggunaan *k* fungsi *hash* yang memetakan setiap elemen input ke dalam posisi bit acak dalam *array* tersebut. Akurasi dan kinerja dari struktur data ini sangat bergantung pada ukuran filter, jumlah fungsi *hash* yang digunakan, serta jumlah elemen yang dimasukkan ke dalamnya.



Gambar 2.4 Ilustrasi Bloom Filter

Gambar 2.4 mengilustrasikan mekanisme kerja Bloom Filter. Struktur ini terdiri dari sebuah *bitstring* dengan panjang 17 bit. Terdapat tiga elemen yang telah dimasukkan ke dalam himpunan, yaitu *x*, *y*, dan *z*. Setiap elemen tersebut dipetakan menggunakan k = 3 fungsi hash ke dalam posisi bit tertentu pada *bitstring*, di mana bit-bit yang terpilih diatur nilainya menjadi 1. Selanjutnya, ketika dilakukan pencarian terhadap

elemen *w*, elemen tersebut juga dipetakan menggunakan tiga fungsi hash yang sama ke posisi bit tertentu. Pada kasus ini, ditemukan bahwa salah satu posisi bit yang dihasilkan bernilai nol, sehingga Bloom Filter dapat melaporkan dengan benar bahwa elemen tersebut tidak berada di dalam set.

Dalam konteks optimasi sistem informasi, implementasi Bloom Filter terbukti memberikan dampak signifikan terhadap kecepatan pemrosesan, khususnya untuk kasus pencarian data yang tidak tersedia (*non-existing data*). Penelitian menunjukkan bahwa dengan menempatkan Bloom Filter sebagai lapisan pemeriksaan sebelum database, waktu yang diperlukan untuk memproses pencarian data yang tidak ada dapat direduksi secara drastis hingga rata-rata 99,16%. Hal ini terjadi karena sistem tidak perlu meneruskan query ke database ketika Bloom Filter memberikan respons negatif [20]. Dalam perancangan solusi yang diusulkan, penggunaan Bloom Filter bertujuan untuk mengoptimalkan efisiensi sistem dengan menyaring kueri data kosong (*non-existing data*) secara cepat, sebelum permintaan tersebut diproses lebih lanjut oleh algoritma pencarian utama yang memakan sumber daya komputasi lebih besar.

2.2.4 Metode Evaluasi Sistem

Untuk mengevaluasi kinerja sistem yang diusulkan, penelitian ini menggunakan Grafana k6 untuk mengukur waktu muat halaman serta waktu respon proses pencarian (*load time*). Waktu performa digunakan untuk mengevaluasi efektivitas mekanisme *early rejection* menggunakan Bloom Filter pada skenario input pencarian yang tidak memiliki kecocokan di dalam database. Selain itu, evaluasi tingkat toleransi (*tolerance level*) kesalahan pada mekanisme pencarian berbasis *similarity detection* dilakukan dengan memanfaatkan *ground truth* sebagai acuan evaluasi untuk menilai tingkat kesesuaian hasil pencarian dalam menangani variasi ejaan dan kesalahan pengetikan.

2.2.4.1 Grafana K6

Grafana k6 adalah perangkat lunak pengujian beban (*load testing*) *open-source* modern yang dirancang untuk mengukur kinerja, keandalan, dan stabilitas infrastruktur serta aplikasi perangkat lunak. Alat ini mengadopsi pendekatan *developer-centric* yang memungkinkan penulisan skenario pengujian menggunakan bahasa pemrograman JavaScript (ES6) atau TypeScript, sehingga memudahkan integrasi proses pengujian kinerja ke dalam siklus pengembangan perangkat lunak (SDLC) dan *pipeline* CI/CD.

Secara teknis, k6 bekerja dengan mensimulasikan lalu lintas pengguna melalui pembangkitan pengguna virtual (*Virtual Users* atau VUs) yang mengeksekusi skrip permintaan (*request*) secara konkuren terhadap sistem target. Arsitektur ini memungkinkan pengujian skalabilitas yang efisien dengan konsumsi sumber daya sistem yang minimal, namun mampu menghasilkan beban lalu lintas yang masif untuk menguji batas kemampuan aplikasi [21]. Untuk memvalidasi performa dari solusi yang diusulkan, Grafana k6 digunakan alat uji utama untuk membandingkan metrik *load time* antara sistem pencarian eksisting dengan solusi yang diusulkan, dengan tujuan menghasilkan pengukuran efektivitas yang bersifat kuantitatif.

2.2.4.2 *Ground Truth*

Ground truth adalah data referensi yang telah diverifikasi dan dianggap mewakili "jawaban yang benar" yang digunakan untuk evaluasi, validasi, dan pengujian kinerja suatu sistem atau model. *Ground truth* berfungsi sebagai tolak ukur pembanding antara hasil yang dihasilkan oleh suatu mekanisme dengan kondisi yang dianggap benar berdasarkan observasi nyata atau data berlabel yang telah ditetapkan sebelumnya [22].

Pada penelitian ini, *ground truth* digunakan sebagai acuan evaluasi dalam mengukur metrik *tolerance level* pada mekanisme pencarian, dengan membandingkan pendekatan pencarian eksisting berbasis *substring matching* dan pencarian berbasis *similarity detection* yang diusulkan, yaitu metode *Winnowing* serta kombinasi *Trigram* dan Jaro-Winkler. Evaluasi ini bertujuan untuk menilai ketahanan masing-masing pendekatan dalam menangani kesalahan pengetikan (*typo*) dan variasi ejaan. Dengan membandingkan hasil pencarian terhadap *ground truth* yang telah disiapkan, penelitian ini dapat mengevaluasi sejauh mana setiap pendekatan mampu mempertahankan kesesuaian hasil pencarian meskipun terjadi kesalahan atau variasi pada input pengguna.