BAB 2
LANDASAN TEORI

Landasan teori merupakan bagian yang memuat konsep, teori, serta pedoman yang mendukung pelaksanaan penelitian. Bab ini akan mengulas teori-teori yang berkaitan dengan topik penelitian, termasuk rumus dan logika algoritma yang berperan dalam pengembangan aplikasi. Selain itu, pendekatan ini bertujuan untuk memastikan bahwa aplikasi yang dikembangkan memiliki dasar ilmiah yang kuat.

2.1 Artificial Intelligence (AI)

Artificial Intelligence atau AI adalah sistem komputer yang dapat meniru kecerdasan manusia. AI dikembangkan untuk dapat melakukan tugas-tugas yang memerlukan kecerdasan manusia seperti mengenali bahasa alami, pengambilan keputusan, dan mempelajari data. Teknologi ini penting pada jaman sekarang karena kegunaannya dalam membantu mengoptimasi serta beradaptasi seiring waktu.

AI telah banyak digunakan dalam berbagai bidang seperti pendidikan, bisnis, dan kesehatan. Pada era industri 4.0, AI banyak mengambil peran penting seperti mengambil keputusan dan memberikan ide baru. AI memiliki kemampuan untuk mengeksekusi berbagai tugas yang pada umumnya memerlukan kecerdasan manusia, seperti berbicara, mendengar, melihat, belajar, berpikir, dan menyelesaikan masalah [9].

2.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) adalah sub-disiplin ilmu komputer yang menjembatani antara bahasa alami dan komputer [10]. NLP memungkinkan untuk komputer memproses, memahami, menganalisis, dan menghasilkan teks atau lisan seperti manusia. Dengan *natural language processing* memungkinkan untuk membuat aplikasi *chatbot* untuk membantu berinteraski dengan manusia.

2.3 Generative AI (GenAI)

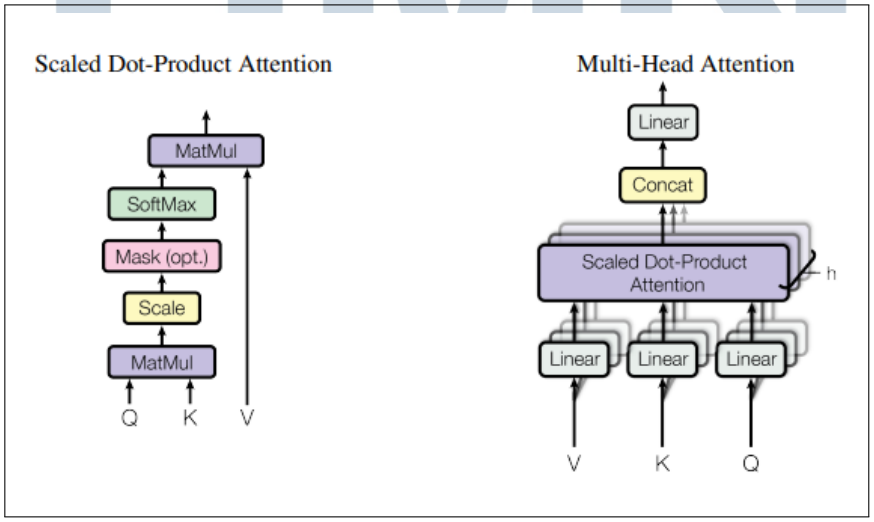
Generative AI atau GenAI adalah kecerdasan buatan yang berfokus pada membuat konten seperti teks, gambar, audio dan video. GenAI telah banyak meningkatkan kreativitas dan efisiensi dalam berbagai bidang industri. Sebagai contohnya pada industri kreatif, GenAI dapat membuat iklan, menulis caption dan bahkan menulis artikel. Kolaborasi AI dan manusia adalah kunci untuk mengatasi tantangan dan meraih peluang yang diciptakan oleh AI generatif [11].

2.4 Transformer

Transformer adalah model *deep learning* yang di desain untuk memproses data atau teks secara efisien. Salah satu konsep yang membuat *transformer* terkenal adalah *self-attention*. Konsep berguna untuk mengetahui nilai sebuah kata dengan melihat keseluruhan kalimat yang mengandung kata tersebut untuk mengetahui konteks tiap kata dan mengerti urutan kata pada kalimat tersebut.

Pada Gambar 2.1 adalah mekanisme *Scaled-Dot Attention* dan *Multi-Head Attention* [2]. Pada *Scaled-Dot Attention* digunakan untuk mencari nilai relevan setiap kata dalam sebuah kalimat. Dengan mendapatkan nilai relevan, sistem dapat mengetahui konteks dari kalimat tersebut.

Pada mekanisme *Multi-Head Attention* adalah pencarian nilai menggunakan *Scaled-Dot Attention* yang dilakukan secara pararel. Masing-masing melakukan pencarian nilai relevan dengan bobot yang berbeda untuk mengetahui nilai relevan dari sudut pandang yang berbeda.



Gambar 2.1. Mekanisme *transformer* [2]

Pada Rumus 2.1, *Attention* adalah ketertarikan kata dengan kata lain. Terdapat tiga variabel penting yaitu *Query(Q)*, *Key(K)* dan *Value(V)*. Variabel *query(Q* adalah kalimat dalam bentuk vektor yang digunakan untuk mencari tahu nilai relevan dengan kata lain. Variabel *key(K* adalah kata dalam bentuk vektor yang digunakan untuk menjadi target dari variabel *query(Q*. Terakhir variabel *value(K)* adalah kalimat dalam bentuk vektor yang menyimpan nilai keseluruhan kalimat. Variabel ini memiliki ukuran matriks dengan baris sebesar jumlah kata dan kolom sebesar dimensi *embedding*.

Pada tahap awal adalah input yang sudah dalam bentuk vektor dikalikan dengan *weight* tiap variabel untuk mendapatkan tiga variabel (Q, K dan V). Kemudian variabel *query(Q)* dikali dengan *transpose* variabel *key(K)* untuk mendapatkan *attention score* yang disimpan pada *value(V)*. Kemudian, normalisasikan *attention score* dengan dibagi dengan akar dari dimensi(d) dibagi kepala atensi(h). Normalisasi ini berguna agar saat diterapkan *softmax*, nilai probabilitas tidak dominan ke salah satu kata. Semakin besar dimensi, nilai h akan semakin besar yang berarti aspek hubungan antar kata semakin jelas. Namun nilai h yang terlalu tinggi akan membebani komputasi.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

Hasil normalisasi *attention score* diterapkan pada *softmax* untuk mendapatkan nilai perhatian pada setiap kata input. Terakhir, hasil dari *softmax* dikalikan dengan variabel *value(V)* untuk mendapatkan nilai konteks pada kata tersebut dalam bentuk vektor.

Berikut adalah inisiasi variabel pada Rumus 2.2. Pada contoh ini, kalimat yang digunakan adalah *"The cat sat"* dengan dimensi 4. Kata yang akan dicari hubungannya dengan kata lain adalah *"cat"*. Dengan begitu, variabel *query(Q)* akan menyimpan nilai kata *"cat"* dalam bentuk vektor.

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

$$W_Q = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 \\ 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.3}$$

$$W_K = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.4}$$

$$W_V = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \\ 0.9 & 1.0 & 1.1 & 1.2 \\ 0.4 & 0.3 & 0.2 & 0.1 \end{bmatrix} \tag{2.5}$$

$$Q = X(cat) \times W_Q = \begin{bmatrix} 2 & 1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$K = X \times W_K = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \end{bmatrix} \tag{2.7}$$

$$V = X \times W_V = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \\ 0.9 & 1.0 & 1.1 & 1.2 \end{bmatrix} \tag{2.8}$$

Berikut adalah contoh perhitungan *attention* pada Rumus 2.9. Langkah awal yaitu melakukan transpose pada variabel *key* dan dikalikan dengan variabel *query*. Kemudian kita normalisasi hasil perkalian membagi nilai QK^T dengan akar dari dimensi yaitu 4. Dengan ini, hasil dari *softmax* akan lebih seimbang. Hasil dari *softmax* adalah ketertarikan kata pada variabel *query* dengan kata lain.

$$K^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{2.9}$$

$$QK^T = \begin{bmatrix} 2 & 1 & 5 \end{bmatrix} \tag{2.10}$$

$$\frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{2}{2} & \frac{1}{2} & \frac{5}{3} \end{bmatrix} \tag{2.11}$$

$$= \begin{bmatrix} 1 & 0.5 & 2.5 \end{bmatrix} \tag{2.12}$$

$$sum(exp) = exp(1) + exp(0.5) + exp(2.5) \tag{2.13}$$

$$= 2.718 + 1.649 + 12.182 \tag{2.14}$$

$$= 16.549 \tag{2.15}$$

$$softmax(\frac{QK^T}{\sqrt{d_k}}) = \begin{bmatrix} \frac{exp(1)}{sum(exp)} & \frac{exp(1)}{sum(exp)} & \frac{exp(1.5)}{sum(exp)} \end{bmatrix} \tag{2.16}$$

$$= \begin{bmatrix} \frac{2.718}{16.549} & \frac{1.649}{16.549} & \frac{12.182}{16.549} \end{bmatrix} \tag{2.17}$$

$$= \begin{bmatrix} 0.164 & 0.099 & 0.736 \end{bmatrix} \tag{2.18}$$

$$softmax(\frac{QK^T}{\sqrt{d_k}}) \times V = \begin{bmatrix} 0.728 & 0.828 & 0.928 & 1.028 \end{bmatrix} \tag{2.19}$$

Berikut adalah hasil perhitungan *softmax* sebelum dikali variabel *value* pada Tabel 2.1. Kata dengan nilai tertinggi pada tabel adalah kata yang memiliki ketertarikan paling tinggi dengan kata pada variabel *query*. Dilihat dari perhitungan *softmax* tersebut, elemen ketiga memiliki nilai paling tinggi yaitu kata *"sat"*. Dengan begitu, kata *"cat"* memiliki ketertarikan dengan kata *"sat"* yang berarti kemungkinan kata selanjutnya dari *"cat"* adalah *"sat"*. Terakhir yaitu hasil *softmax* dikali dengan variabel *value(V)* untuk mendapatkan nilai representasi baru dari variabel *query*. Dalam contoh ini berarti representasi nilai baru untuk kata *"cat"*.

Tabel 2.1. Hasil perhitungan *softmax* sebelum dikali value

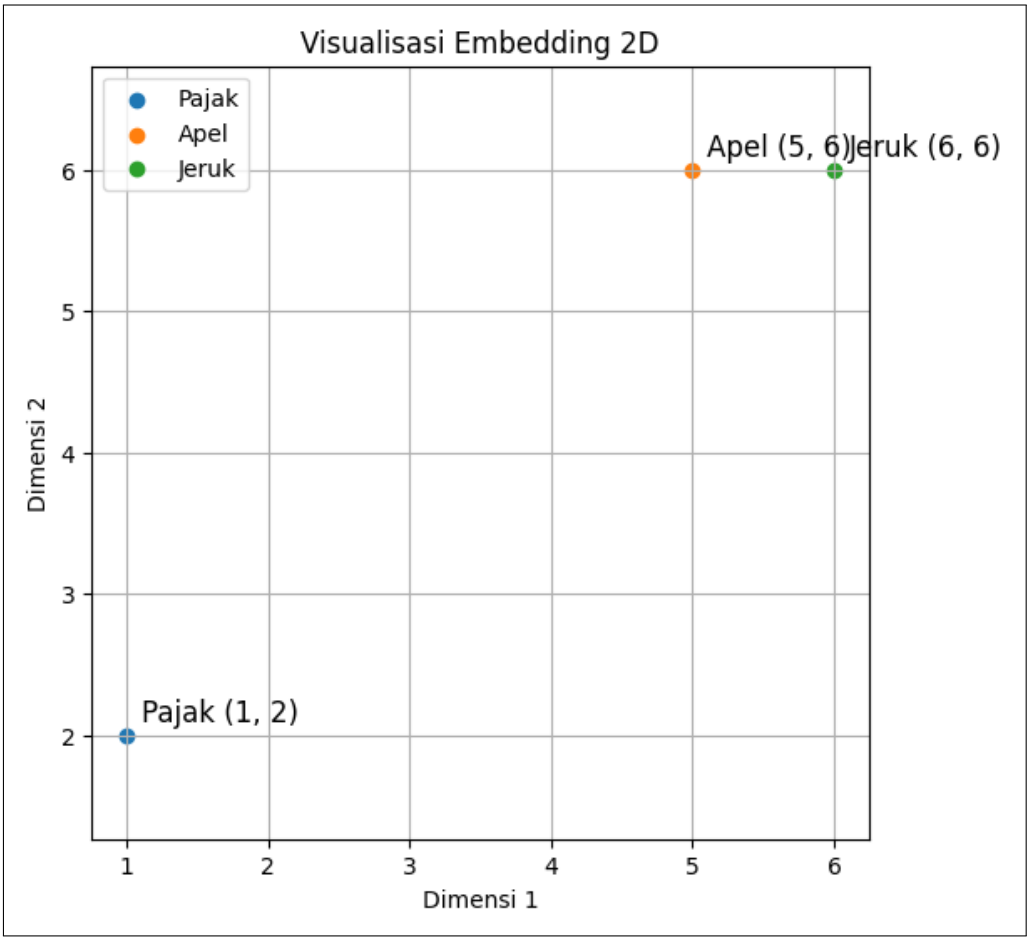
Kata	Nilai
The	0.164
cat	0.099
sat	0.736

2.5 Embedding

Embedding adalah proses mengubah bentuk data menjadi numerik yaitu vektor. Data yang dapat diubah menjadi vektor adalah teks, gambar, video dan audio. *Embedding* mempunyai model yang tiap model mempunyai hasil *embed* yang berbeda. Masing-masing model mempunyai dimensi yang berbeda. Dimensi model yang tinggi berarti data vektor yang dihasilkan semakin rumit yang membuat data yang disimpan lebih kontekstual.

Embedding model dapat menghasilkan vektor yang berbeda karena tiap model menggunakan algoritma yang berbeda-beda. Algoritma yang digunakan model memiliki dua fokus dalam pelatihan. Pertama yaitu *statis* yang mengubah kata ke dalam *vector* tanpa melihat konteks kata tersebut dalam kalimat seperti *Word2Vec*. Kedua yaitu kontekstual yang mengubah kata ke dalam *vector* berdasarkan konteks kata tersebut dalam kalimat seperti *BERT*. Selain itu, data yang diberikan ke algoritma untuk melatih model juga mempengaruhi hasil dari *embedding*.

Embedding biasa digunakan untuk mencari kemiripan antara data. Sebagai contoh pada Gambar 2.2, terdapat tiga kata yaitu pajak, apel dan jeruk. Tiap kata akan diubah menjadi vektor 1x2. Ketiga kata dilakukan proses *embed* dengan hasil pajak menjadi [1, 2], apel menjadi [5, 6] dan jeruk menjadi [6, 6]. Data apel memiliki nilai yang dekat dengan jeruk karena mereka memiliki kesamaan yaitu buah sedangkan kata pajak tidak memiliki konteks yang dekat antara kata apel dan jeruk.



Gambar 2.2. Plot dua dimensi pada *text embedding*

Salah satu representasi lain dari *embedding* adalah *One Hot Encoding* yaitu mengubah kategorial data menjadi bentuk numerik. Sebagai contoh pada Tabel 2.2, terdapat tiga warna yaitu biru, hijau dan merah. Warna tersebut diubah menjadi numerik yaitu angka 0 melambangkan warna biru, angka 1 melambangkan warna hijau dan angka 2 melambangkan warna merah.

Tabel 2.2. Contoh *One Hot Encoding*

Numerik	Biru	Hijau	Merah
0	1	0	0
1	0	1	0
2	0	0	1

2.6 Large Language Model (LLM)

Large Language Model (LLM) adalah kecerdasan buatan yang dirancang untuk meniru manusia seperti memahami dan menghasilkan teks dalam bahasa alami. *LLM* menggunakan arsitektur *transformer* yaitu jaringan saraf yang mempelajari konteks dan makna dalam data berurutan. Dengan begitu, *LLM* dapat melakukan berbagai tugas seperti menjawab pertanyaan atau membuat keputusan.

Salah satu contoh dari *LLM* adalah ChatGPT yang menghasilkan teks dalam bahasa alami yang relevan berdasarkan input. Model pada *LLM* umumnya dilatih dengan menggunakan sejumlah besar data teks dan menggunakan teknik pembelajaran mendalam atau *deep learning* untuk mempelajari pola dan struktur bahasa [12]. Model *LLM* dilatih dengan menggunakan teknik *Unsupervised Learning*, yaitu model belajar dengan data yang tidak dilabel atau tanpa bantuan manusia.

Proses pelatihan model dengan data teks dalam jumlah yang besar menggunakan arsitektur *transformer*. Setiap kalimat akan dicari pola nilai kedekatan antara kata. Dari pola nilai digunakan untuk menghasilkan teks yang mirip dengan manusia.

2.7 Cosine Similarity

Metode *Cosine Similarity* merupakan salah satu metode yang digunakan untuk mengukur kemiripan dokumen dengan membandingkan antara *vector query* dengan *vector* dokumen, dengan *query* dan dokumen direpresentasikan dalam model ruang *vector* [13]. Pada Rumus 2.20 adalah rumus untuk menghitung kemiripan antara data. Cara menggunakan rumus ini dengan menggantikan variabel A dan B sebagai vektor pada data untuk menghitung kemiripan antara kedua data tersebut. Hasil dari perhitungan ini memiliki nilai di *range* 0 sampai 1. Kedua data dikatakan mirip jika nilai *cosine* semakin mendekati angka 1.

$$Cosine\ Similarity = \frac{A \times B}{||A|| \times ||B||} \tag{2.20}$$

Sebagai contoh, kita mau menghitung kemiripan kata antara pajak dan apel dengan rumus *cosine*. Pertama kata jeruk dan apel akan diubah dalam bentuk

vektor sebagai contoh kita ubah ke dalam bentuk vektor 1x2. Berikut contoh perhitungannya pada Rumus 2.21 yang didapatkan nilai 0.9962. Dari hasil tersebut didapatkan bahwa kata Jeruk dan Apel mirip karena nilainya mendekati 1.

$$\text{Vektor Jeruk} = [2,3] \tag{2.21}$$

$$\text{Vektor Apel} = [5,6] \tag{2.22}$$

$$\frac{A \times B}{||A|| \times ||B||} = \frac{2(5)+3(6)}{\sqrt{2^2+3^2} \times \sqrt{5^2+6^2}} \tag{2.23}$$

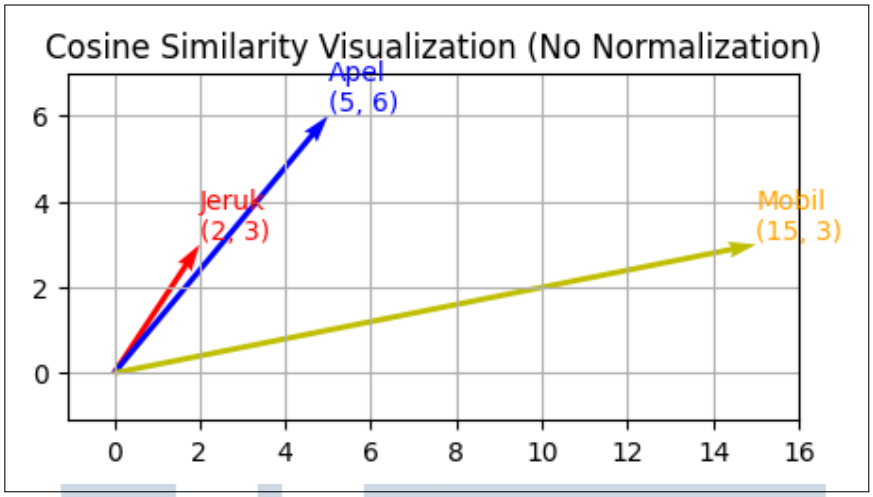
$$= \frac{10+18}{\sqrt{4+9} \times \sqrt{25+36}} \tag{2.24}$$

$$= \frac{28}{\sqrt{13} \times \sqrt{61}} \tag{2.25}$$

$$= \frac{28}{28.1602} \tag{2.26}$$

$$= 0.9962 \tag{2.27}$$

Dari rumus tersebut didapatkan contoh nilai vektor jeruk [2,3] dan nilai vektor apel [5,6]. Pada Gambar 2.3 adalah *angle plot* dari nilai vektor jeruk dan apel. *Angle plot* digunakan karena dimensi vektor adalah dua. Dapat dilihat kedua arah panah memiliki *angle* yang kecil yang berarti memiliki nilai kemiripan yang tinggi. Sebagai contoh tambahan kata mobil dengan vektor [15,3]. Terlihat arah panah mobil dengan jeruk dan apel jauh karena memiliki konteks yang berbeda.

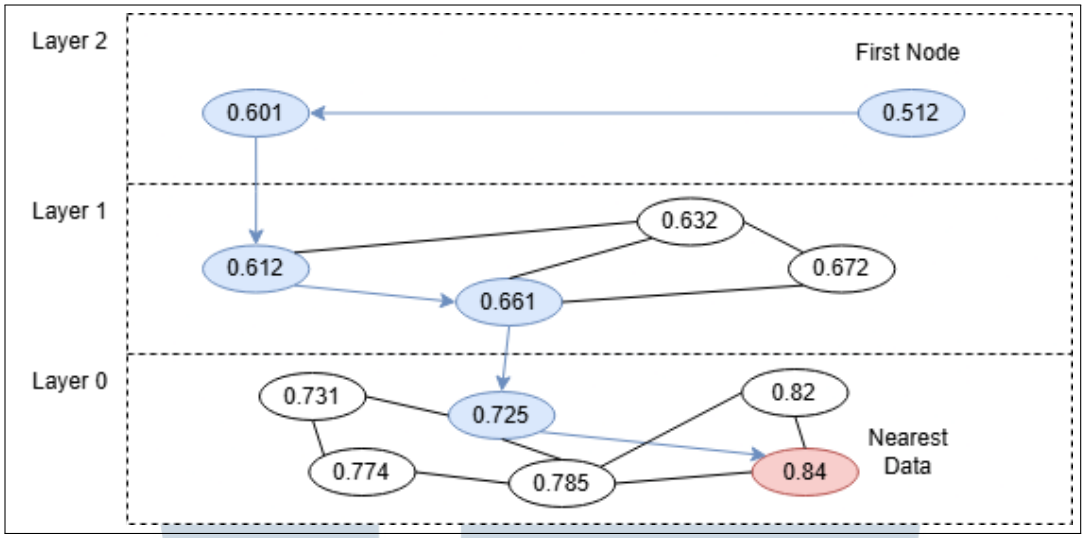


Gambar 2.3. *Vector* plot hasil *cosine simialrity*

2.8 Vector Database

Vector Database adalah database yang menyimpan datanya dalam bentuk vektor yang merupakan representasi dari objek dalam ruang berdimensi tinggi. Berbeda dengan database yang lain, *vector database* menggunakan algoritma salah satunya adalah *Hierarchical Navigable Small World (HNSW)*, yaitu teknik pencarian berdasarkan kedekatan dari nilai vektor yang disimpan dalam bentuk *graph*. *Vector database* dapat menangani analisis dan pemrosesan data berskala besar dan real-time, yang yang sangat penting untuk ilmu data modern dan aplikasi AI [14].

Pada Gambar 2.4 adalah bentuk dari *graph* algoritma *Hierarchical Navigable Small World (HNSW)* [15]. Algoritma ini digunakan untuk mempermudah pencarian data melalui kemiripan vektor. Proses kerjanya adalah mencari data dengan vektor yang paling mendekati dengan vektor yang diinput. Pencarian dimulai dari data *random* pada level teratas. Kemudian dilakukan *greedy search* dengan perhitungan jarak *cosine* untuk mencari data vektor yang lebih dekat dengan vektor sebelumnya. Jika tidak ada data tetangga yang lebih mendekati, maka pencarian akan turun ke level dibawahnya untuk penyempitan pencarian hingga menemukan data dengan skor yang paling mirip. Pada proses input data, hal yang sama dilakukan tetapi pada akhir proses dilakukan penambahan data vektor ke tetangga dari data yang paling mirip.



Gambar 2.4. Ilustrasi *Hierarchical Navigable Small World graph*

Vector database adalah tempat penyimpanan data yang berbentuk vektor. Untuk mendistribusikan beban kerja dan mengoptimalkan pemanfaatan sumber daya di beberapa mesin atau cluster [14]. Teknik di atas bertujuan untuk meningkatkan peforma dari pencarian pada *vector database*. *Vector database* sering digunakan sebagai pencarian kesamaan atau *similarity search*. Berikut cara kerja dari *vector database*.

1. Penyimpanan (Storage) Pada tahap ini akan menjelaskan bagaimana cara data disimpan dalam *vector database*. Data yang disimpan dapat dalam bentuk teks, video, atau audio. Sebelum disimpan, data harus melalui proses *embedding* yaitu mengubah bentuk kalimat menjadi vektor. Dalam kasus teks, tiap kata akan diubah menjadi vektor unik berbeda dengan yang lain.

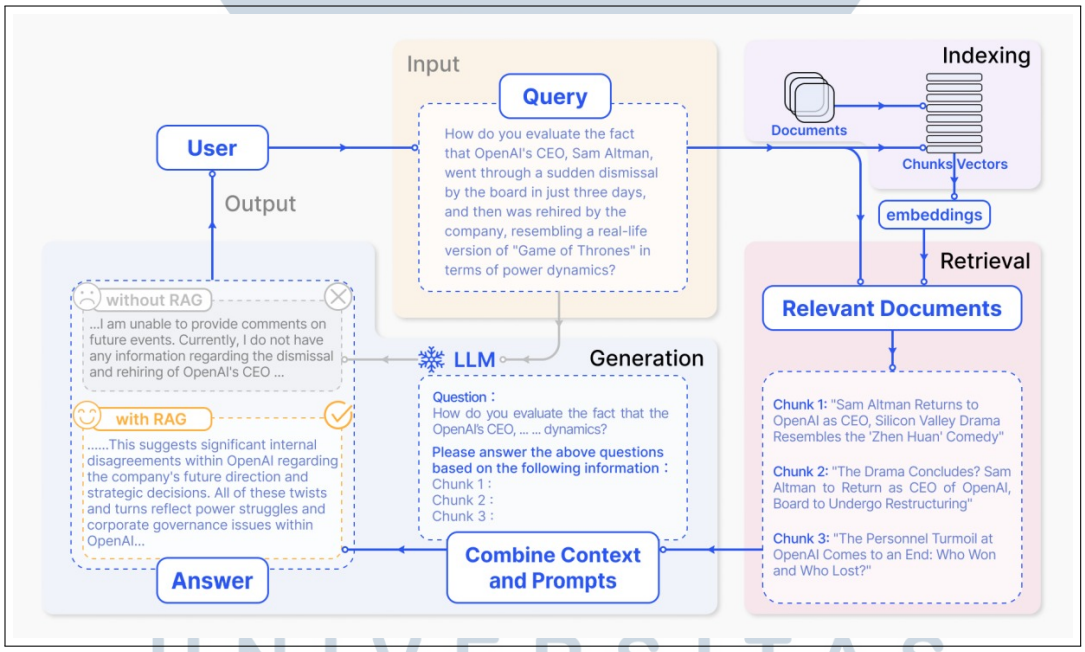
2. Pengindeksan (Indexing) Data yang disimpan akan melalui proses indeks. Proses indeks ini bertujuan untuk mengoptimalkan pencarian pada *vector database* tersebut. Teknik pengindeksan tergantung pada kebutuhan, ukuran data, dan dimensi vektor.

3. Pencarian (Querying) Pada tahap ini adalah cara mencari data dalam *vector database*. *Query* atau input akan diubah dulu ke dalam bentuk vektor melalui proses *embedding*. *Query* yang telah diubah ke vektor akan dicari kedekatannya dengan vektor lain menggunakan metode kesamaan metriks sebagai contoh *Cosine Similarity*. Terakhir, sistem akan memberikan data

yang paling mirip tergantung dari Top-K dimana K adalah jumlah dari data yang diinginkan.

2.9 Retrieval Augmented Generation (RAG)

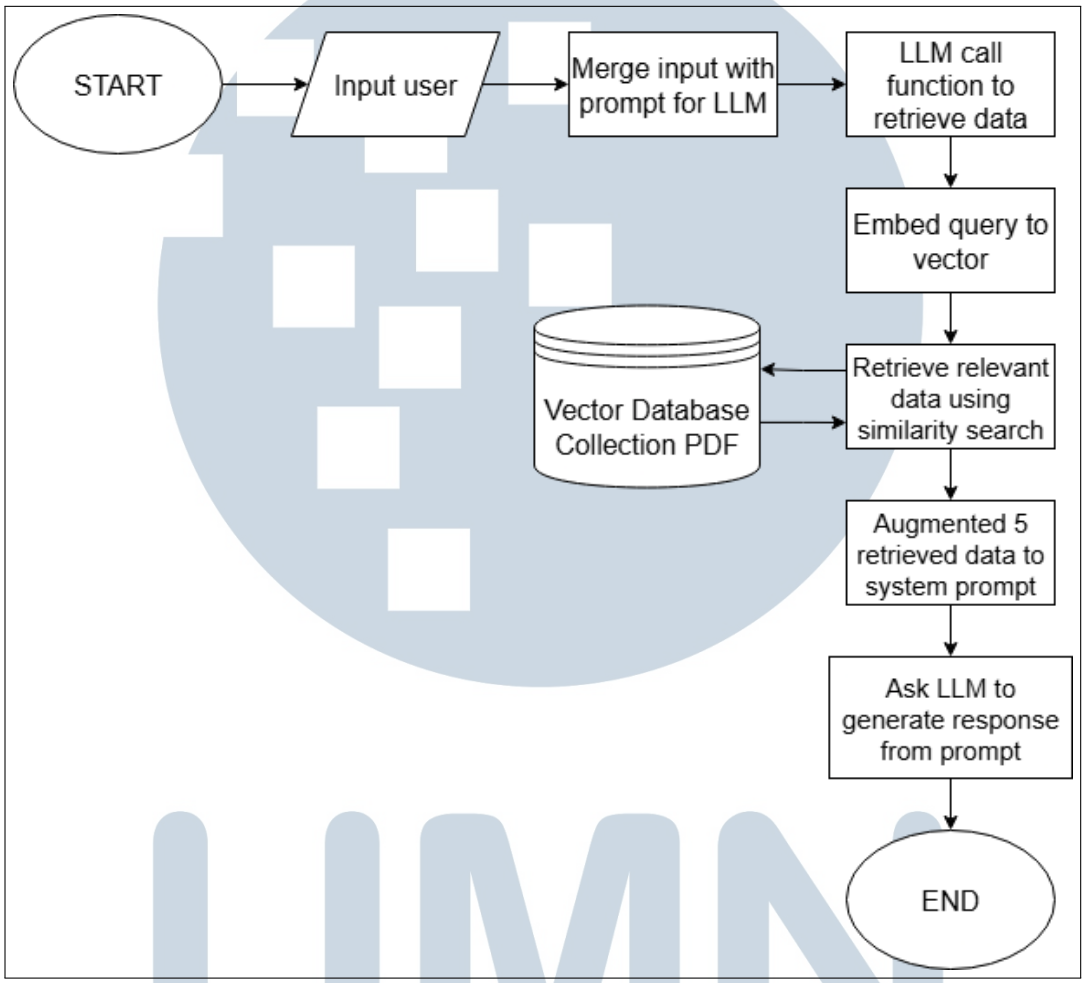
Retrieval Augmented Generation (RAG) adalah teknik yang meningkatkan akurasi dan kualitas respon dari *Large Language Model (LLM)* dengan memanfaatkan *vector database*. Teknik ini terbukti karena respon yang diberikan tidak hanya mengandalkan pengetahuan dari model sendiri, tetapi memanfaatkan informasi yang relevan dan terkini dari *vector database*. Dengan merujuk pada pengetahuan eksternal, RAG secara efektif mengurangi masalah pembuatan konten yang salah secara faktual [3]. Terdapat tiga langkah dalam teknik RAG, yaitu: Pengambilan (*Retrieval*), Augmentasi (*Augmented*), dan Generasi (*Generation*). Berikut adalah penjelasan dari langkah-langkah RAG sesuai pada Gambar 2.5 [3].



Gambar 2.5. Flow RAG

Pada Gambar 2.6 adalah alur *Large Language Model (LLM)* dengan *Retrieval Augmented Generation (RAG)*. Pada proses *input* pengguna, akan masuk ke *prompt* untuk LLM. Dari LLM dapat memanggil *tools* atau *function* untuk melakukan *retrieval* atau mengambil data dari *vector database*. Proses mengambil data dengan melakukan embed pada *input* menjadi vektor dan membandingkan kedekatan data di *vector database* menggunakan rumus *cosine similarity*. Data

yang telah diambil akan dilakukan *augmentation* dengan *prompt* serta *input* pengguna. Dari *prompt* yang telah dibuat, LLM akan melakukan *generation* untuk mendapatkan jawaban terakhir yang diberikan kepada pengguna.



Gambar 2.6. Pipeline LLM dan RAG [3]

1. Pengambilan (*Retrieval*) Langkah pertama saat menerima input yaitu akan dilakukan pengambilan data pada *vector database*. Input yang di berikan akan di saring dahulu menjadi *query* agar pencarian lebih optimal. *Query* yang telah di saring akan diubah bentuknya menjadi vektor melalui proses *embedding*. Kemudian akan dicari vektor yang paling mirip dengan *Query* yang telah diubah ke vektor pada *vector database*.

2. Augmentasi (*Augmented*) Data yang telah ditemukan oleh sistem akan digunakan pada langkah selanjutnya yaitu Augmentasi. Pada langkah ini, adalah pengabungan data yang telah diambil dengan instruksi sistem. *LLM*

akan mengoptimalkan jawaban yang akan diberikan dengan memproses input dengan data yang sudah diambil melalui proses *retrieval*. Proses augmentasi ini dapat berupa menambah kalimat yang relevan dengan data yang didapatkan dari *vector database*.

3. Generasi (*Generation*) Langkah generasi adalah proses membuat jawaban dari *LLM* dengan bantuan informasi data yang telah diambil dari *vector database*. Dengan bantuan data yang telah diambil, memungkinkan *LLM* untuk lebih memahami konteks input yang diberikan. Bantuan data yang diambil dari *vector database* diharapkan dapat mengurangi halusinasi dan menghasilkan jawaban yang akurat.

2.10 Metode Evaluasi G-Eval

Metode evaluasi *G-Eval* adalah metode evaluasi yang menggunakan *LLM* sebagai penilai *(LLM-as-a-judge)*. *G-Eval* adalah evaluasi berbasis *LLM* dengan *prompt* yang menggunakan konsep *Chain of Thoughts (CoT)* [1]. *LLM* yang menilai akan membuat serangkaian langkah dari berdasarkan permintaan. Dari langkah tersebut, dilakukan proses *Form-Filling* yaitu *LLM* penilai akan mengevaluasi setiap *test case* satu per satu. *Test case* berupa input dan output dari RAG yang akan dievaluasi. Hasil akhir berupa *score correectness* yaitu rata-rata nilai dari semua percobaan pada *test case*.

Chain of Thoughts CoT adalah teknik pemikiran yang digunakan *LLM* untuk berpikir setiap langkah sebelum memberikan jawaban terakhir. Pemikiran ini membuat *LLM* untuk berpikir dahulu agar lebih memahami masalah yang pada pertanyaan. Pemikiran ini dapat meningkatkan akurasi jawaban terutama pada pertanyaan yang kompleks.

Pada Tabel 2.3 adalah perbandingan *metrics* untuk evaluasi *chatbot* dalam mengukur korelasi teks. Kriterianya ada *Naturalness* yaitu seberapa mirip teks dengan manusia, *Coherence* yaitu seberapa mudah dipahami, *Engagingness* yaitu seberapa menarik teksnya dan *Groundedness* yaitu seberapa baik dan berhubungan teks dengan informasi sumber. Tiap kriteria memiliki dua sub kriteria, yaitu *Pearson (r)* yang mengukur hubungan linear antara dua variabel dan *Spearman Rank (ρ)* yang mengukur hubungan motonik antara dua variabel.

Metrics yang dibandingkan ada delapan, yaitu ROUGE-L, BLEU-4, METEOR, BERTScore, USR, UniEval, G-Eval-3.5 dan G-Eval-4. Dari semua *metrics* evaluasi, didapatkan *metrics* G-Eval di antara G-Eval-3.5 dan G-Eval-4

mendominasi setiap kriteria yang diuji. Dengan ini, metode evaluasi untuk menguji *chatbot* akan mengggunakan G-Eval.

Dari Tabel 2.3, Evaluasi menggunakan *LLM* atau G-Eval lebih akurat dibandingkan dari *metrics* yang disebutkan. Dari sisi *naturalness*, *coherence*, *engagingness* dan *groundness*. Oleh karena itu, dalam menguji *chatbot* kita hanya menggunakan *metrics* G-Eval.

Tabel 2.3. Perbandingan *metrics* untuk evaluasi *chatbot* [1]

Metrics	Naturalness		Coherence		Engagingness		Groundedness		AVG	
	r	ρ	r	ρ	r	ρ	r	ρ	r	ρ
ROUGE-L	0.176	0.146	0.193	0.203	0.295	0.300	0.310	0.327	0.243	0.244
BLEU-4	0.180	0.175	0.131	0.235	0.232	0.316	0.213	0.310	0.189	0.259
METEOR	0.212	0.191	0.250	0.302	0.367	0.439	0.333	0.391	0.290	0.331
BERTScore	0.226	0.209	0.214	0.233	0.317	0.335	0.291	0.317	0.262	0.273
USR	0.337	0.325	0.416	0.377	0.456	0.465	0.222	0.447	0.358	0.403
UniEval	0.455	0.330	0.602	0.455	0.573	0.430	0.577	0.453	0.552	0.417
G-EVAL-3.5	0.532	0.539	0.519	0.544	**0.660**	**0.691**	**0.586**	0.567	0.574	0.585
G-EVAL-4	**0.549**	**0.565**	**0.594**	**0.605**	0.627	0.631	0.531	0.551	**0.575**	**0.588**