BAB 2
LANDASAN TEORI

Studi ini memanfaatkan berbagai sumber literatur sebagai bahan rujukan dan panduan dalam merumuskan penelitian skripsi ini. Melalui telaah literatur yang dilakukan, diidentifikasi sejumlah teori serta rumus-rumus logika algoritma yang relevan untuk digunakan dalam pengembangan perangkat lunak yang dijelaskan dalam skripsi ini. Pendekatan ini memungkinkan integrasi konsep-konsep teoretis dengan aplikasi, memastikan bahwa pengembangan program didasarkan pada fondasi ilmiah yang kuat dan terkini.

2.1 Artificial Intelligence (AI)

AI adalah sistem komputer yang mampu melakukan tugas-tugas yang biasanya memerlukan kecerdasan manusia, termasuk *learning, reasoning, dan self-correction*. Teknologi ini sangat penting dalam era industri 4.0, memungkinkan otomatisasi dan pengambilan keputusan berdasarkan analisis data yang dilatih untuk model AI tersebut.

AI telah banyak diadopsi dalam berbagai sektor dan sangat berdampak bagi optimalisasi proses kerja, serta meningkatkan efisiensi dan hasil produksi. AI berperan penting dalam transformasi industri dimana semua proses dan operasi dapat dilakukan dan dipelajari oleh sistem sehingga memungkinkan proses untuk dilakukan secara lebih efisien, mengurangi lead time, meningkatkan fleksibilitas, dan mengoptimalkan penggunaan sumber daya [11].

2.2 Large Language Model

Large Language Models adalah jenis *Artificial Intelligence* yang dapat meniru kecerdasan manusia. Teknik yang digunakan dalam *Large Language Models* adalah dengan menggunakan model statistik dan teknik *Deep Learning* untuk memproses dan memahami data berbentuk teks dalam jumlah yang besar. Model ini dapat mempelajari pola rumit dan hubungan yang ada dari setiap data sehingga dalam dunia *Generative AI* memungkinkan untuk menghasilkan konten baru yang sangat mirip dengan gaya dan karakteristik berdasarkan data yang diberikan terhadap model tersebut.

Generative AI dirancang untuk menghasilkan kontek baru baik itu berupa teks, gambar, audio, maupun video. Salah satu implementasi dari *LLM* adalah *ChatGPT* yang merupakan jenis *generative AI* yang dirancang khusus untuk menghasilkan respons teks yang dapat dimengerti oleh manusia berdasarkan prompt yang diberikan. Model-model ini dilatih dengan jumlah data teks yang sangat besar, menggunakan teknik *Unsupervised Learning* untuk mempelajari pola bahasa. *LLM* memiliki kemampuan untuk memahami dan menghasilkan teks yang relevan secara kontekstual, menjadikannya sebuah teknologi yang canggih dan dapat dimanfaatkan untuk berbagai aplikasi seperti *translator*, *chat bot*, atau *text generation* [12].

LLM merupakan fondasi penting dalam pengembangan *generative AI* berbasis teks, dengan munculnya aplikasi seperti *GPT* dari *OpenAI*, merupakan salah satu contoh terbaik dari kemajuan dalam dunia *AI*. *LLM* dikembangkan menggunakan arsitektur transformer yang detail sehingga memungkinkan *LLM* untuk melakukan berbagai tugas seperti menjawab pertanyaan dengan kemampuan yang jauh melampaui sistem yang sebelumnya tersedia untuk publik dimana hal tersebut membuka potensi besar mereka dalam berbagai aspek lainnya khususnya dalam *Generative AI* [13].

2.2.1 Vector Database

Vector Database adalah jenis database yang menyimpan data dalam bentuk vektor yang merupakan representasi matematis dari data. *Vector Database* dirancang untuk *searching* dan juga *retrieval* berdasarkan kemiripan nilai vector yang tersimpan. Vector database memiliki kecepatan dan keakuratan yang tinggi dan mendukung data yang kompleks dan tidak terstruktur seperti teks, gambar, audio, ataupun video [6].

Vector Database menggunakan teknik khusus seperti *sharding*, *partitioning*, *caching*, dan *replication* untuk mendistribusikan beban kerja dan mengoptimalkan penggunaan sumber daya. Untuk melakukan pencarian, *Vector Database* menggunakan *Nearest Neighbor Search (NNS)* dan *Approximate Nearest Neighbor Search (ANNS)* untuk menemukan data yang paling mirip berdasarkan jarak atau kemiripan vektor [6].

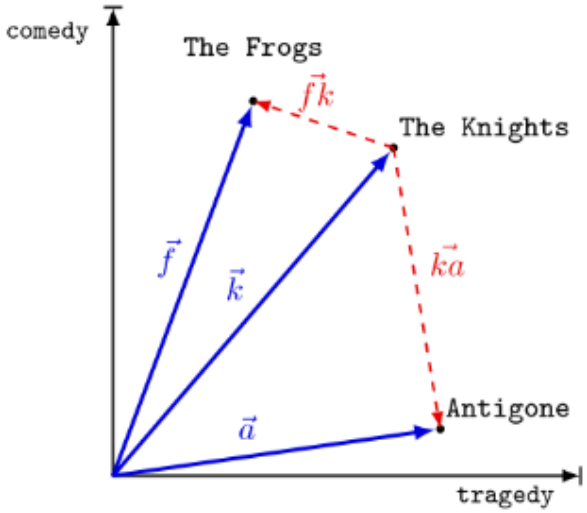
Arsitektur Vector Database melibatkan beberapa komponen utama seperti:

1. **Sharding:** Mendistribusikan *vector database* ke beberapa mesin atau kluster berdasarkan kriteria tertentu seperti fungsi hash atau rentang kunci.

2. **Partitioning:** Membagi database vektor menjadi bagian yang lebih kecil dan dapat dikelola berdasarkan kriteria seperti lokasi geografis, kategori, atau frekuensi.

3. **Caching:** Menyimpan data yang sering diakses atau baru digunakan di memori yang cepat dan dapat diakses seperti RAM untuk mengurangi latensi dan meningkatkan kinerja pengambilan data.

4. **Replication:** Membuat salinan data vektor dan menyimpannya di node atau kluster yang berbeda untuk meningkatkan ketersediaan, durabilitas, dan kinerja dari *vector database*.



Gambar 2.1. Visualisasi data vector database
Sumber: Taipalus, 2024 [14]

A Cara kerja vector database

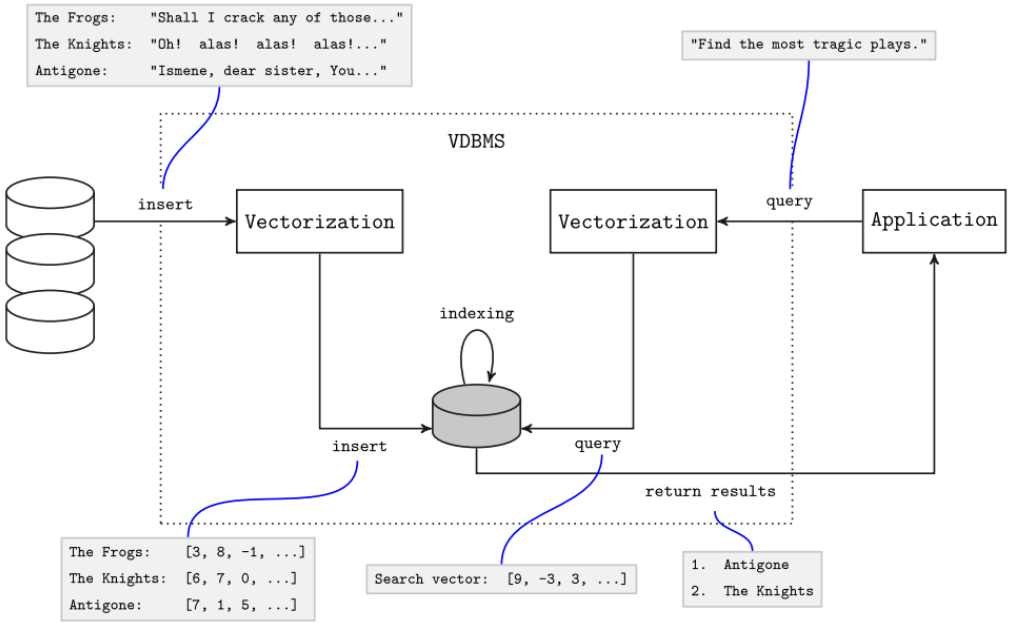
Cara kerja vector database dapat digambarkan sebagai berikut:

1. **Penyimpanan data:** Vector databases menyimpan data dalam format angka yang merepresentasikan nilai dari object yang akan disimpan kedalam database. Data yang diterima akan dirubah terlebih dahulu kedalam bentuk format *vector* atau *array* yang disebut dengan proses *embedding*. Data yang dapat disimpan oleh vector database dapat berupa gambar, audio, maupun teks. Data vektor disimpan dalam tabel yang terdiri dari baris dan kolom, di

mana setiap baris mewakili satu objek dan setiap kolom mewakili atribut atau fitur dari objek tersebut.

2. **Indexing:** Setelah data vector disimpan kedalam vector database, maka vector database akan melakukan proses indexing. Proses indexing dapat dilakukan secara otomatis ataupun manual oleh pengguna database. Proses indexing secara otomatis juga disebut sebagai *indexing on-the-fly*. Tujuan dari proses indexing ini adalah supaya data yang tersimpan dalam vector database dapat dicari dengna cepat dan juga optimal ketika pengguna melakukan pencarian data dalam vector database.

3. **Query dan Analisis:** Pengguna database dapat melakukan query terhadap vector database dengan cara melakukan prompt konteks data yang ingin dicari, kemudian prompt dari user tersebut akan diubah kedalam bentuk vector melalui proses *Embedding* kemudian vector yang dihasilkan melalui proses embedding tersebut akan dilakukan pencarian kedalam database untuk mencari vektor dengan karakteristik yang mirip dengan algoritma *Nearest Neighbor Search* atau *Approximate Nearest Neighbor Search*.



Gambar 2.2. Visualisasi query vector database
Sumber: Taipalus, 2024 [14]

Vector Database memainkan peran penting dalam mendukung *Generative AI*, terutama dalam metode *Retrieval Augmented Generation (RAG)*, dengan

menyediakan kemampuan untuk secara efisien menyimpan, mencari, dan mengambil vektor data yang relevan yang digunakan untuk melengkapi proses *text generation* oleh model *Generative AI*. Hal ini memungkinkan *Generative AI* untuk memproduksi hasil yang lebih akurat, relevan, dan berdasarkan informasi terkini dengan mengakses *vector database* sebelum diproses oleh mesin *LLM* [6].

Retrieval-Augmented Generation (RAG) adalah suatu metode yang dapat dipakai untuk meningkatkan kemampuan *Large Language Models (LLM)* dengan menggabungkan pengetahuan dari *database* eksternal. Hal ini memungkinkan model untuk menghasilkan respons yang lebih akurat dan relevan dengan memanfaatkan informasi terkini yang disimpan dalam *database*.

Cara kerja metode *Retrieval Augmented Generation (RAG)* dapat dilakukan melalui tiga langkah utama: pengambilan (retrieval), generasi (generation), dan augmentasi (augmentation) yang dapat menghasilkan sebuah sistem yang dapat mengatasi tantangan dalam *LLM* seperti memberi informasi yang salah atau biasanya disebut dengan istilah halusinasi [7].

1. *Retrieval*

   Langkah awal dari teknik RAG adalah *Rertrieval* yang mengacu kepada teknik pengambilan data untuk mengambil informasi yang relevan berdasarkan *prompt* dari pengguna kedalam database yang digunakan. Salah satu jenis database yang dapat digunakan untuk melakukan pencarian data yang relevan terhadap prompt yang diberikan oleh user adalah vector database dimana prompt dari user akan di konversi terlebih dahulu melalui *LLM* untuk mendapatkan *keyword* yang optimal untuk dilakukan *query* kedalam database, Kemudian akan dilakukan proses embedding berdasarkan hasil *query* yang di hasilkan oleh *LLM* untuk mendapatkan nilai vectornya kemudian akan dicarikan data yang mirip berdasarkan vector tersebut kedalam database untuk mendapatkan kumpulan data yang relevan sesuai dengan query yang diberikan oleh sistem.

2. *Augmentation*

   Setelah proses *Retrieval* selesai dan data-data relevan telah ditemukan berdasarkan permintaan pengguna, langkah selanjutnya adalah *Augmentation*. Tahap ini merupakan proses untuk mengoptimalkan jawaban yang akan dihasilkan oleh *LLM*. Dalam proses ini, sebuah template pertanyaan akan dibuat berdasarkan data-data relevan yang ditemukan

melalui proses *Retrieval*, serta pertanyaan yang diajukan oleh pengguna. Template ini kemudian akan digunakan untuk mengajukan pertanyaan kepada *LLM*. Selain itu, proses *Augmentation* juga melibatkan penyempurnaan konten dari hasil Retrieval yang diperoleh. Misalnya, dengan menyusun kembali atau menyempurnakan struktur kalimat, menambahkan konteks yang relevan, atau memperjelas informasi yang ditemukan dari database.

3. *Generation*

   Proses Generation pada RAG merujuk pada proses penghasilan jawaban yang diajukan kepada LLM berdasarkan data yang telah diambil dari database dan juga permintaan pengguna yang telah dibentuk dalam suatu template khusus. Template yang dibuat akan digunakan untuk bertanya kepada LLM sehingga LLM dapat memahami konteks yang diminta oleh pengguna serta memanfaatkan informasi tambahan dari database. Selama proses *generation*, *LLM* akan menggunakan pengetahuan yang diperoleh dari tahap *retrieval* dan disempurnakan melalui tahap *Augmentation* untuk menghasilkan jawaban yang sesuai dengan permintaan pengguna. Proses ini diharapkan menghasilkan jawaban yang sesuai dengan permintaan atau pertanyaan yang diajukan oleh pengguna karena jawaban yang dihasilkan oleh *LLM* berdasarkan *dataset* yang diambil dari database sehingga memungkinkan *LLM* untuk memahami pertanyaan dan menjawab sesuai konteks informasi yang sudah disiapkan.

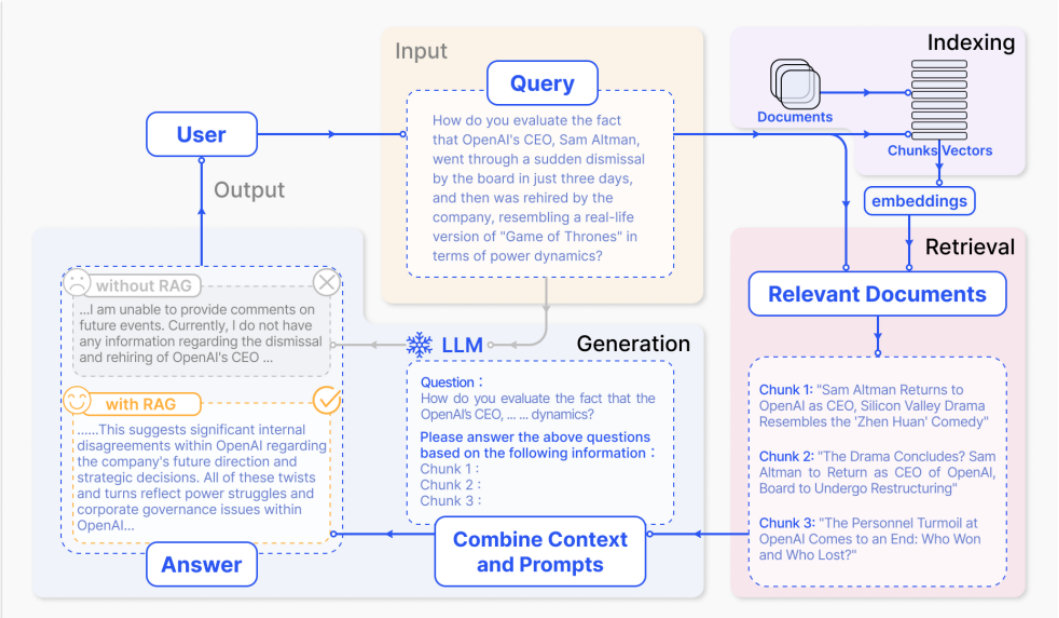
B Limitasi Retrieval Augmented Generation (RAG)

(a) Dalam metode RAG, proses antara Retrieval, Augmentation, dan Generation merupakan proses yang terpisah dimana masing masing dari proses tersebut akan diproses dengan prompt yang berbeda melalui LLM. Hal tersebut mengakibatkan jawaban yang dihasilkan oleh LLM antara proses Retrieval dengan Generation mungkin tidak selalu sinkron yang mengakibatkan memungkinkan beberapa informasi yang tidak diperlukan juga terbawa.

(b) Proses Retrieval pada RAG dijalankan setiap kali pengguna memberikan pertanyaan kepada sistem tanpa membawa pertanyaan pengguna sebelum-sebelumnya. Hal ini memungkinkan hasil dari proses retrieval tidak sesuai dengan apa yang dimaksud oleh pengguna.

(c) Hasil dari proses retrieval dalam satu putaran pertanyaan pengguna tidak dapat diubah ataupun ditambahkan. Sebagai contoh jika hasil retrieval menyarankan untuk adanya pencarian lebih lanjut, maka hal tersebut tidak dapat dilakukan.

Proses Retrieval Augmented Generation secara umum dapat digambarkan melalui diagram sebagai berikut:



Gambar 2.3. Rangkaian proses implementasi Retrieval Augmented Generation
Sumber: Y.Gao, 2023 [7]