BAB 2
LANDASAN TEORI

Pada bagian ini dijabarkan teori-teori yang mendasari penelitian. Penyajiannya difokuskan pada definisi, konsep, dan batasan matematis yang secara langsung berkaitan dengan arsitektur *Artificial Intelligence* (AI), khususnya sistem *Retrieval-Augmented Generation* (RAG). Penjelasan dalam bab ini berfokus pada mekanisme dasar pemrosesan bahasa alami, teknik pemotongan dokumen (*chunking*), representasi *embedding*, serta pencarian vektor yang menjadi fondasi eksperimen.

2.1 *Artificial Intelligence* dan *Natural Language Processing*

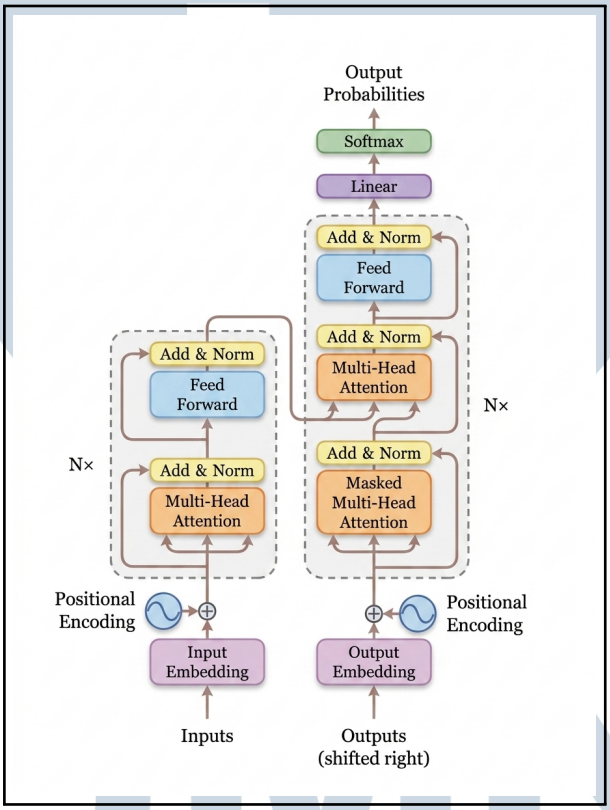
Artificial Intelligence (AI) atau Kecerdasan Buatan merupakan cabang keilmuan komputer yang berfokus pada pengembangan sistem komputasi dengan kemampuan untuk melakukan tugas-tugas yang secara tradisional membutuhkan kecerdasan kognitif manusia, seperti penalaran, pembelajaran, dan pengambilan keputusan. Dalam perkembangannya, salah satu sub-bidang AI yang mengalami akselerasi paling signifikan adalah *Natural Language Processing* (NLP) atau Pemrosesan Bahasa Alami. NLP berfokus pada interaksi antara komputer dan bahasa manusia, memungkinkan mesin untuk membaca, mengurai, memahami, dan mereproduksi bahasa alami secara kontekstual [2].

Secara historis, sistem pencarian informasi sangat bergantung pada pencocokan kata kunci (*keyword-based search*) yang kaku. Namun, dengan kemajuan NLP, sistem bergeser menuju pencarian semantik (*semantic search*) yang mampu mencocokkan makna antara kueri pengguna dan isi dokumen melalui perhitungan kemiripan semantik [3]. Kapabilitas ini menjadi krusial dalam lingkungan akademik. Pertanyaan mahasiswa sering kali menggunakan bahasa sehari-hari yang tidak selalu sama persis dengan frasa baku yang tertulis di dalam buku panduan atau dokumen resmi kampus.

2.1.1 *Large Language Models* (LLM) dan Fenomena Halusinasi

Large Language Models (LLM) adalah arsitektur jaringan saraf tiruan (*neural networks*) berskala masif—umumnya berbasis arsitektur *Transformer*—yang dilatih menggunakan miliaran hingga triliunan korpus teks data dari internet

[1, 4]. Proses prapelatihan (*pre-training*) ini memungkinkan model, seperti arsitektur GPT (*Generative Pre-trained Transformer*), untuk mempelajari pola struktur bahasa dan relasi semantik dalam teks [2]. LLM beroperasi dengan cara memprediksi probabilitas token (kata atau subkata) berikutnya dalam sebuah sekuens teks berdasarkan konteks token-token sebelumnya [13]. Gambaran umum arsitektur *Transformer* yang menjadi fondasi LLM modern disajikan pada Gambar 2.1.



Gambar 2.1. Arsitektur model *Transformer*. Diadaptasi dari [13]

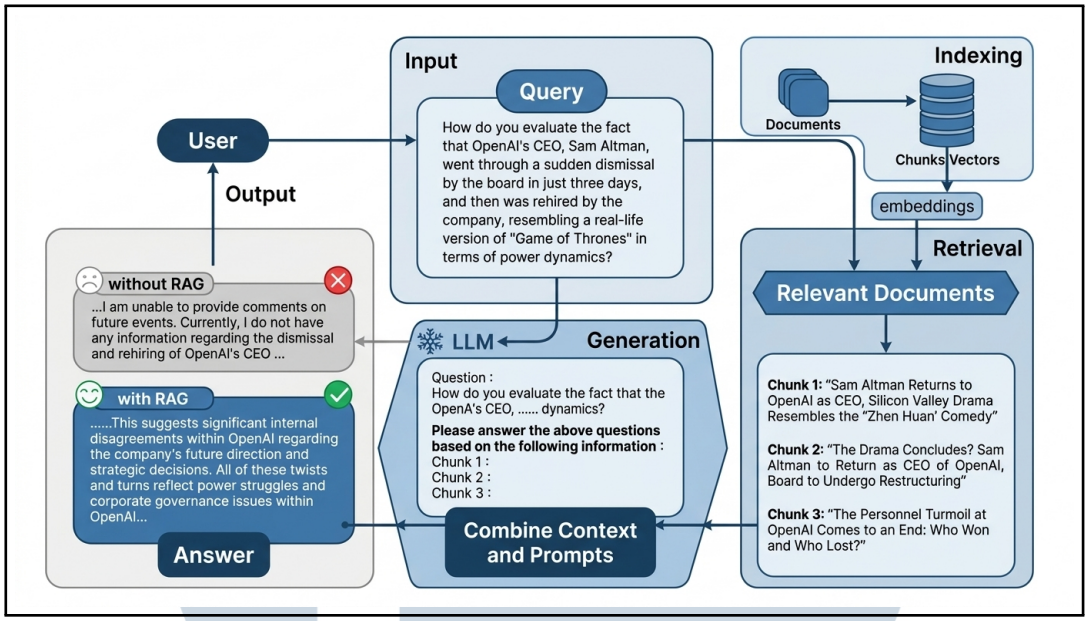
Meskipun LLM memiliki kemampuan generatif yang canggih, model ini memiliki kelemahan penting yang dikenal sebagai Halusinasi AI (*AI Hallucination*) [6]. Halusinasi merujuk pada kondisi saat LLM menghasilkan keluaran (jawaban) yang secara tata bahasa terlihat meyakinkan dan fasih, tetapi secara faktual salah, tidak masuk akal, atau tidak memiliki dasar sumber yang valid [6, 5]. Fenomena ini terjadi karena LLM pada dasarnya beroperasi berdasarkan pengetahuan parametrik (*parametric knowledge*), yaitu bobot probabilitas statis yang dipelajari selama masa pelatihan, dan tidak memiliki akses langsung ke pangkalan data eksternal secara *real-time* [5]. Dalam konteks asisten akademik perguruan tinggi, halusinasi

merupakan risiko serius, karena AI dapat menghasilkan informasi yang tidak sesuai dengan prosedur kemahasiswaan, persyaratan beasiswa, atau ketentuan organisasi yang berlaku di institusi. Berbagai penelitian mengelompokkan halusinasi berdasarkan penyebab kemunculannya, bentuk kesalahannya, serta pendekatan mitigasi yang dapat diterapkan [6, 5].

2.1.2 *Retrieval-Augmented Generation* (RAG)

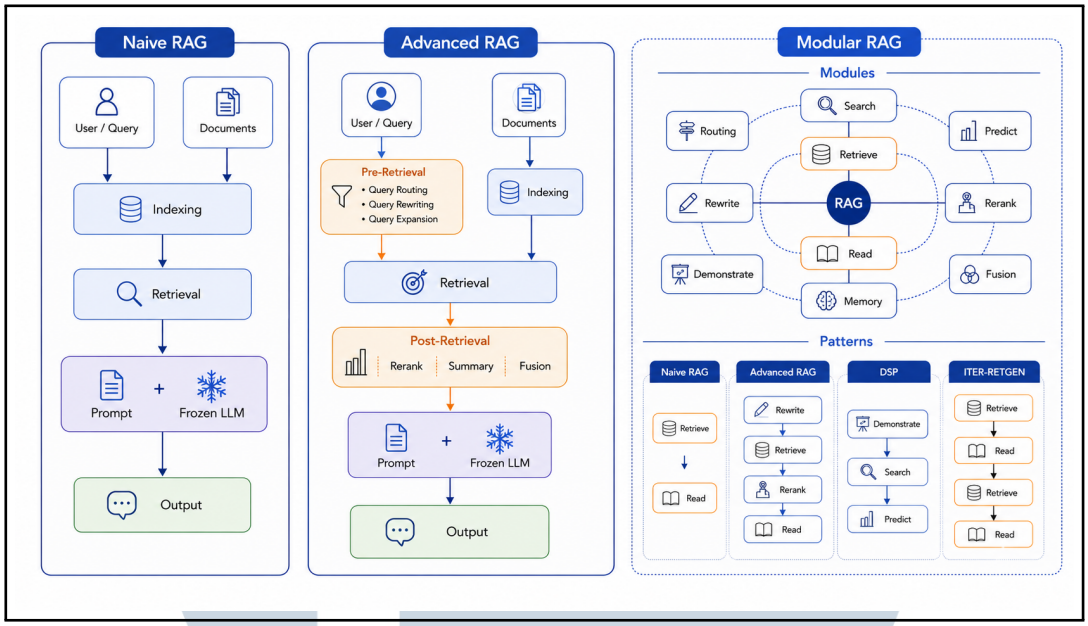
Untuk mengatasi masalah halusinasi pada model bahasa, sebuah paradigma hibrida yang dinamakan *Retrieval-Augmented Generation* (RAG) diperkenalkan. RAG merupakan kerangka kerja komputasi yang menggabungkan kemampuan inferensi semantik dari model parametrik (LLM) dengan pengetahuan non-parametrik yang diambil dari pangkalan data eksternal [7]. Dengan pendekatan ini, alih-alih hanya mengandalkan memori internal model yang statis dan rentan kedaluwarsa, sistem akan terlebih dahulu mencari (*retrieve*) dokumen relevan dari korpus basis pengetahuan yang telah divalidasi, untuk kemudian memasukkannya (*augment*) sebagai konteks tambahan sebelum model menghasilkan jawaban (*generate*) [3].

Arsitektur RAG umumnya terdiri dari tiga fase utama: (1) Fase Ingesti, yaitu kumpulan dokumen mentah dipotong dan diubah menjadi representasi vektor numerik; (2) Fase Pencarian (*Retrieval*), yaitu sistem membandingkan kueri pengguna dengan vektor dokumen untuk menemukan potongan teks yang paling relevan; dan (3) Fase Generasi (*Generation*), yaitu LLM menyintesis konteks yang berhasil ditarik tersebut menjadi jawaban yang koheren [3]. Dalam penerapannya, fase ingesti dan pencarian memanfaatkan basis data vektor, sedangkan fase generasi memanfaatkan model bahasa untuk menyintesis konteks menjadi jawaban. Kerangka kerja ini membantu meningkatkan keterlacakan jawaban terhadap dokumen sumber, sebuah karakteristik penting untuk aplikasi tingkat institusi atau akademik. Ilustrasi arsitektur umum sistem RAG disajikan pada Gambar 2.2.



Gambar 2.2. Arsitektur umum sistem *Retrieval-Augmented Generation* (RAG) untuk *question answering*. Diadaptasi dari [3]

Dalam perkembangannya, paradigma RAG telah berevolusi menjadi tiga generasi utama, yaitu *Naive RAG*, *Advanced RAG*, dan *Modular RAG*. *Naive RAG* mengikuti alur sederhana berupa indeksasi, pencarian, dan generasi secara berantai. *Advanced RAG* menambahkan optimasi pra-pencarian dan pasca-pencarian untuk meningkatkan kualitas konteks. Sementara itu, *Modular RAG* menawarkan fleksibilitas lebih tinggi dengan modul-modul fungsional yang dapat dipertukarkan dan alur yang tidak harus sekuensial [3]. Perbandingan ketiga paradigma tersebut diilustrasikan pada Gambar 2.3. Paradigma *Naive RAG* banyak digunakan ketika tujuan utamanya adalah menjaga alur sistem tetap sederhana dan mudah diimplementasikan.



Gambar 2.3. Perbandingan tiga paradigma RAG: *Naive*, *Advanced*, dan *Modular*. Diadaptasi dari [3]

2.1.3 *Vector Space Model* dan *Cosine Similarity*

Pada arsitektur pencarian semantik modern, pencocokan teks tidak lagi dilakukan secara leksikal (karakter per karakter), melainkan direpresentasikan di dalam ruang vektor berdimensi tinggi yang dikenal sebagai *Vector Space Model* (VSM). Dalam model ini, dokumen dan kueri pengguna dikonversi menjadi larik angka matematis (*vector embeddings*) menggunakan model *embedding*. Dalam praktiknya, dimensi vektor sering diseragamkan agar konsisten dengan skema pada basis data vektor. Representasi vektor tersebut memungkinkan kalimat-kalimat yang memiliki makna semantik serupa ditempatkan pada titik koordinat yang saling berdekatan di dalam ruang multidimensi, terlepas dari perbedaan kosakata yang digunakan secara harfiah [3].

Untuk mengukur tingkat kemiripan antara vektor kueri pengguna ($\vec{q}$) dan vektor dokumen dalam basis data ($\vec{d}$), metrik *Cosine Similarity* sering digunakan dalam pencarian semantik [3]. *Cosine Similarity* mengukur kosinus sudut antara dua vektor berdimensi bebas. Perhitungan *Cosine Similarity* ditunjukkan pada Rumus 2.1.

$$\text{Cosine Similarity}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{\|\vec{q}\|\,\|\vec{d}\|} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}} \tag{2.1}$$

Nilai hasil perhitungan *Cosine Similarity* berkisar dari -1 hingga 1. Nilai yang mendekati 1 merepresentasikan bahwa arah kedua vektor sangat sejajar, yang mengindikasikan tingkat kesamaan semantik yang sangat tinggi antara pertanyaan pengguna dan isi dokumen [3]. Keunggulan utama dari metrik ini dibandingkan metrik jarak lainnya, seperti *Euclidean Distance*, adalah kekebalannya terhadap disparitas panjang teks; ia hanya mengevaluasi ”arah makna” (*orientation*) tanpa terdistorsi oleh perbedaan jumlah token secara absolut.

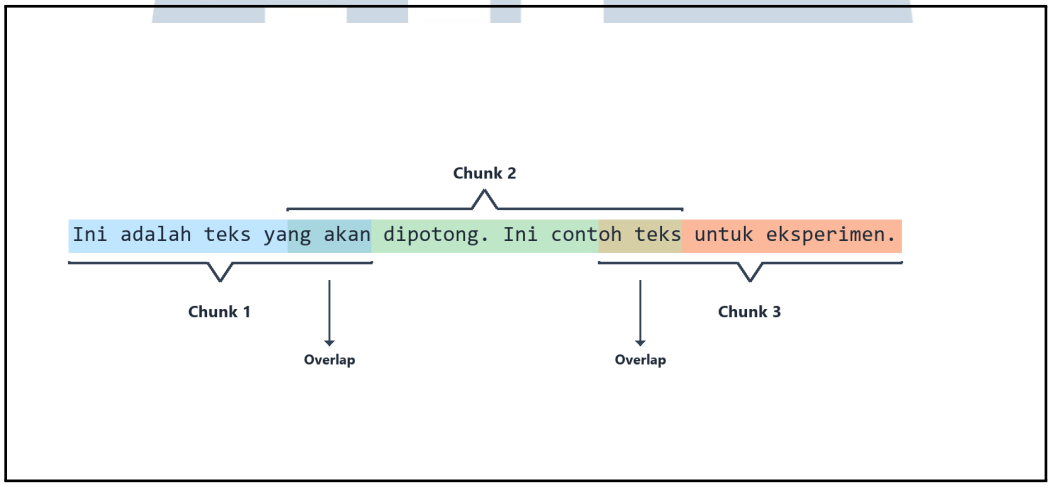
2.1.4 Strategi *Text Chunking*

Dalam arsitektur *Retrieval-Augmented Generation* (RAG), dokumen sumber yang panjang tidak dapat langsung dimasukkan ke dalam model bahasa secara utuh karena adanya batasan *context window* pada LLM [3]. Oleh karena itu, dokumen harus terlebih dahulu dipecah menjadi potongan-potongan teks yang lebih kecil melalui proses yang disebut *chunking*. *Chunking* merupakan tahapan krusial dalam fase ingesti data RAG. Kualitas potongan teks yang dihasilkan secara langsung memengaruhi kualitas pencarian semantik dan, pada akhirnya, kualitas jawaban yang dihasilkan oleh LLM [3].

Terdapat beberapa strategi *chunking* yang umum digunakan. Strategi paling sederhana adalah *Fixed-Size Chunking*, yaitu teks dipecah berdasarkan jumlah karakter atau token yang seragam tanpa memperhatikan batas logis kalimat maupun paragraf. Strategi kedua adalah *Recursive Character Splitting*, yaitu teks dipecah secara bertingkat menggunakan pemisah hierarkis, seperti paragraf, kalimat, lalu kata, agar batas potongan lebih menghormati struktur teks asli. Strategi ketiga adalah *Semantic Chunking*, yaitu strategi yang menganalisis pergeseran topik dalam teks untuk menempatkan batas potongan pada titik transisi semantik sehingga setiap *chunk* memiliki koherensi tematik yang tinggi [3].

Dua parameter utama dalam strategi *chunking* adalah ukuran potongan (*chunk size*) dan tumpang tindih (*overlap*). Ukuran *chunk* menentukan jumlah karakter atau token dalam setiap potongan. Apabila ukuran terlalu kecil, konteks informasi dapat terpotong dan makna utuh kalimat hilang. Sebaliknya, apabila

ukuran terlalu besar, potongan tersebut berpotensi mengandung banyak informasi yang tidak relevan (kebisingan) yang dapat mengacaukan proses pencarian [3, 10]. Parameter *overlap* berfungsi menjaga kontinuitas konteks antar-potongan dengan cara menduplikasi sebagian teks di perbatasan dua *chunk* yang bersebelahan. Dalam praktik *chunking*, nilai *overlap* biasanya dipilih secara moderat agar kontinuitas konteks terjaga tanpa menambahkan redundansi berlebihan. Penentuan kombinasi terbaik dari kedua parameter ini merupakan salah satu fokus utama eksperimen yang dilakukan. Ilustrasi proses *chunking* dengan *overlap* disajikan pada Gambar 2.4.



Gambar 2.4. Ilustrasi proses *chunking* dengan *overlap*

Salah satu strategi yang banyak digunakan adalah *Recursive Character Splitting* melalui modul `RecursiveCharacterTextSplitter` dari *framework* LangChain, yang memecah teks secara bertingkat menggunakan pemisah hierarkis (paragraf, kalimat, lalu kata) agar setiap potongan tetap menghormati batas-batas semantik alami dari teks asli. Strategi ini memberikan keseimbangan antara kesederhanaan implementasi dan kualitas potongan teks. Dibandingkan *Fixed-Size Chunking* yang memotong teks secara kaku tanpa memperhatikan batas kalimat, *Recursive Character Splitting* menghasilkan potongan yang lebih koheren secara kontekstual. Sementara itu, *Semantic Chunking* memerlukan komputasi tambahan untuk mendeteksi pergeseran topik melalui analisis *embedding* antar-kalimat, yang menambah kompleksitas dan biaya komputasi. Dengan menggunakan *Recursive Character Splitting*, pengaruh ukuran *chunk* dan *overlap* dapat dianalisis secara lebih bersih dalam rancangan eksperimen.

2.1.5 Parameter *Top-K Retrieval*

Setelah dokumen dipotong dan direpresentasikan sebagai vektor dalam *vector database*, tahap berikutnya dalam arsitektur RAG adalah proses pencarian (*retrieval*). Ketika pengguna mengajukan pertanyaan, kueri tersebut diubah menjadi vektor dan dibandingkan dengan seluruh vektor dokumen menggunakan metrik kesamaan (misalnya *Cosine Similarity*). Pada implementasi basis data vektor, pencarian umumnya dilakukan melalui fungsi yang mengurutkan *chunk* berdasarkan kedekatan kosinus vektor. Parameter *top-k* menentukan berapa banyak potongan dokumen (*chunks*) teratas—berdasarkan peringkat kesamaan tertinggi—yang dipilih untuk dimasukkan ke dalam LLM sebagai konteks tambahan [11].

Penentuan nilai *top-k* memiliki dampak signifikan terhadap kualitas jawaban sistem RAG. Pada nilai *top-k* yang terlalu kecil, potongan dokumen yang memuat informasi penting dapat tidak terjangkau dalam hasil pencarian (*missed the top ranked documents*) [9], terutama pada pertanyaan yang membutuhkan informasi dari beberapa bagian dokumen sekaligus. Sebaliknya, nilai *top-k* yang terlalu besar memperkenalkan potongan dokumen yang kurang relevan ke dalam konteks LLM. Penelitian oleh Cuconasu et al. (2024) membuktikan bahwa kehadiran dokumen yang mengganggu (*distracting documents*), yaitu dokumen yang terkait secara semantik namun tidak memuat jawaban, dapat menurunkan akurasi jawaban LLM secara signifikan [11]. Temuan ini menunjukkan bahwa penambahan jumlah konteks tidak selalu meningkatkan kualitas jawaban, karena dokumen yang mengganggu tersebut dapat mengalihkan fokus model dari informasi yang benar-benar relevan.

Yoran et al. (2024) juga menunjukkan bahwa model RAG sangat sensitif terhadap konteks yang tidak relevan, yang dapat memicu terjadinya *cascading errors* terutama dalam skenario penalaran multi-langkah [12].

Oleh karena itu, terdapat *trade-off* fundamental antara meningkatkan *recall* (kemampuan menemukan semua informasi relevan) melalui nilai *top-k* yang besar, dan menjaga *precision* (kualitas relevansi konteks) melalui nilai *top-k* yang kecil. Penemuan titik keseimbangan terbaik dari parameter ini, bersama dengan strategi *chunking*, menjadi fokus eksperimen utama yang dilakukan.

2.1.6 Metrik Evaluasi *Retrieval*

Untuk mengukur efektivitas komponen pencarian (*retrieval*) dalam arsitektur RAG, dua metrik standar dalam bidang *Information Retrieval* digunakan, yaitu *Hit Rate* dan *Mean Reciprocal Rank* (MRR).

A *Hit Rate*

Hit Rate (disebut juga *Hit Rate@K*) adalah metrik biner yang mengukur apakah setidaknya satu dokumen relevan berhasil ditemukan di antara *top-k* hasil pencarian untuk suatu kueri [3]. Metrik ini memberikan gambaran tingkat keberhasilan sistem dalam menemukan informasi yang dibutuhkan.

Untuk satu kueri, nilainya adalah 1 jika dokumen relevan ditemukan dan 0 jika tidak. Secara agregat untuk Q kueri, perhitungan *Hit Rate* ditunjukkan pada Rumus 2.2.

$$Hit\ Rate = \frac{1}{Q}\sum_{i=1}^{Q} hit_i \tag{2.2}$$

Pada persamaan tersebut, $\text{hit}_i = 1$ jika setidaknya satu dokumen relevan berada dalam *top-k* hasil pencarian untuk kueri ke-i, dan $\text{hit}_i = 0$ jika sebaliknya.

B *Mean Reciprocal Rank* (MRR)

Berbeda dengan *Hit Rate* yang hanya mengukur keberadaan dokumen relevan, *Mean Reciprocal Rank* (MRR) juga mempertimbangkan posisi peringkat dokumen relevan pertama dalam daftar hasil pencarian [3]. MRR memberikan skor yang lebih tinggi jika dokumen relevan ditemukan pada posisi teratas dan skor yang lebih rendah jika dokumen relevan berada pada posisi yang lebih bawah.

Untuk satu kueri, *Reciprocal Rank* dihitung sebagai $1/R$, dengan R menyatakan posisi peringkat dokumen relevan pertama. Secara agregat, perhitungan *Mean Reciprocal Rank* ditunjukkan pada Rumus 2.3.

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q} \frac{1}{rank_i} \tag{2.3}$$

Pada persamaan tersebut, rank_i adalah posisi peringkat dokumen relevan

pertama untuk kueri ke-i. Jika tidak ada dokumen relevan yang ditemukan, maka $\frac{1}{\text{rank}_i} = 0$.

Kombinasi kedua metrik ini memberikan evaluasi yang komprehensif: *Hit Rate* mengukur apakah sistem berhasil menemukan dokumen yang tepat, sedangkan MRR mengukur seberapa cepat (pada posisi berapa) dokumen yang tepat tersebut ditemukan.

2.1.7 Akurasi Faktual

Selain metrik *retrieval* yang mengevaluasi komponen pencarian, kualitas jawaban akhir yang dihasilkan oleh LLM juga perlu diukur secara terpisah. Hal ini karena performa *retrieval* yang tinggi tidak selalu menjamin jawaban akhir yang benar [6, 5]. Meskipun sistem berhasil mengambil *chunk* yang relevan (*Hit Rate* tinggi dan MRR tinggi), LLM masih dapat gagal dalam menyintesis informasi tersebut menjadi jawaban yang akurat. Kegagalan ini dapat terjadi karena beberapa faktor, antara lain: (1) konteks yang terlalu panjang sehingga LLM kehilangan fokus pada informasi kunci, (2) konteks yang ambigu atau mengandung informasi yang saling bertentangan antar-*chunk*, serta (3) keterbatasan inheren LLM dalam menalar dan menyusun jawaban meskipun informasi yang benar sudah tersedia dalam konteks [11, 12].

Sebagai contoh, pada skenario *top-k* yang tinggi, LLM mungkin menerima banyak *chunk* relevan namun juga disertai *chunk* yang mengganggu (*distracting documents*), yaitu *chunk* yang terkait secara semantik namun tidak memuat jawaban. Keberadaan *chunk* yang mengganggu tersebut dapat mendistraksi model dan menyebabkan jawaban yang keliru meskipun *Hit Rate* mencapai 100%. Oleh karena itu, metrik Akurasi Faktual menjadi pelengkap krusial yang mengukur kualitas jawaban *end-to-end*, bukan hanya kualitas pencarian.

Akurasi Faktual mengukur proporsi jawaban yang dihasilkan oleh sistem RAG yang secara substansi sesuai dengan fakta yang tertulis dalam dokumen sumber aslinya [6, 5]. Perhitungan Akurasi Faktual ditunjukkan pada Rumus 2.4.

$$\text{Akurasi Faktual} = \frac{\text{Jumlah jawaban faktual yang benar}}{\text{Total jumlah pertanyaan uji}} \times 100\% \tag{2.4}$$

Evaluasi Akurasi Faktual dilakukan secara otomatis oleh sistem dengan

membandingkan jawaban terhadap kata kunci yang telah ditetapkan dalam *ground truth* (Buku Panduan Kemahasiswaan 2025 MyUMN). Kombinasi ketiga metrik — *Hit Rate*, MRR, dan Akurasi Faktual — memberikan evaluasi yang komprehensif terhadap sistem RAG: *Hit Rate* mengukur apakah sistem menemukan dokumen yang tepat, MRR mengukur seberapa cepat dokumen tepat tersebut ditemukan, dan Akurasi Faktual mengukur apakah jawaban akhir yang dihasilkan benar-benar sesuai fakta.

2.2 Penelitian Terdahulu

Beberapa penelitian terdahulu yang relevan dengan topik implementasi *Retrieval-Augmented Generation* (RAG) telah dilakukan. Ringkasan penelitian-penelitian tersebut disajikan pada Tabel 2.1.

Tabel 2.1. Perbandingan penelitian terdahulu

No	Peneliti (Tahun)	Fokus Penelitian	Hasil Utama	Keterkaitan dengan Topik
1	Gao et al. (2023)	Survei komprehensif arsitektur RAG (*Naive*, *Advanced*, *Modular*)	Menyusun taksonomi RAG dan menekankan pentingnya strategi *chunking* dalam alur *retrieval*	Menjadi dasar konseptual untuk pembahasan taksonomi RAG dan peran *chunking* dalam sistem
2	Wang et al. (2024)	Pengaruh strategi *chunking* terhadap performa RAG	Performa RAG sangat sensitif terhadap ukuran *chunk*; ukuran optimal bervariasi per domain	Menegaskan pentingnya pengujian ukuran *chunk* pada domain yang berbeda
3	Cuconasu et al. (2024)	Dampak dokumen yang mengganggu (*distracting documents*) dalam konteks LLM	Dokumen yang mengganggu dalam konteks menurunkan akurasi jawaban LLM secara signifikan	Menjadi dasar teoritis untuk analisis dampak *distracting documents* pada variasi *top-k*

Bersambung ke halaman berikutnya

Tabel 2.1. Perbandingan Penelitian Terdahulu (lanjutan)

No	Peneliti (Tahun)	Fokus Penelitian	Hasil Utama	Keterkaitan dengan Topik
4	Barnett et al. (2024)	Identifikasi titik kegagalan (*failure points*) pada sistem RAG	Mengidentifikasi 7 titik kegagalan RAG, termasuk dokumen jawaban yang terlewat dari hasil *top-k* (*missed the top ranked documents*)	Menunjukkan risiko dokumen relevan tidak terjangkau oleh *top-k* terhadap kualitas *retrieval*
5	Yoran et al. (2024)	Sensitivitas model RAG terhadap konteks tidak relevan	Konteks tidak relevan memicu *cascading errors* dalam penalaran multi-langkah	Menguatkan pembahasan sensitivitas sistem RAG terhadap konteks yang tidak relevan
6	Xu et al. (2024)	Perbandingan *retrieval-augmentation* dengan *long context window* pada LLM	Augmentasi *retrieval* dapat mencapai performa setara dengan *context window* panjang namun dengan biaya komputasi jauh lebih rendah	Menjustifikasi pilihan pendekatan *retrieval-based* (RAG) alih-alih sekadar memperluas *context window*
7	Ramakanth Bhat et al. (2025)	Analisis multi-*dataset* pengaruh *chunk size* pada *long-document retrieval*	*Chunk* kecil (64–128 token) optimal untuk pertanyaan faktual, sedangkan *chunk* besar (512–1024 token) optimal untuk konteks luas; tidak ada satu ukuran optimal universal	Menjadi dasar pemilihan rentang *chunk size* 250–2000 karakter dan justifikasi pengujian multi-level pada Bab 3
8	Liu et al. (2024)	Pemanfaatan *context window* panjang oleh model bahasa (*lost in the middle*)	Model cenderung kurang memanfaatkan informasi di posisi tengah konteks yang panjang, menurunkan kualitas jawaban	Menjadi dasar teoritis fenomena *lost in the middle* pada analisis anomali *top-k* besar di Bab 4

Berdasarkan Tabel 2.1, dapat diidentifikasi bahwa penelitian-penelitian terdahulu umumnya menguji komponen RAG secara terpisah — baik hanya *chunking* maupun hanya *top-k retrieval*. Temuan-temuan tersebut menunjukkan bahwa ruang kajian mengenai pengujian silang antara kedua parameter tersebut pada dokumen institusional berbahasa Indonesia masih terbuka.