BAB 3
METODOLOGI PENELITIAN

3.1 Studi Literatur

Studi literatur dilakukan untuk mengumpulkan dan menganalisis informasi yang relevan dari sumber-sumber ilmiah seperti jurnal atau publikasi lainnya yang berkaitan dengan topik penelitian. Tujuannya adalah untuk memahami latar belakang teori, metodologi yang ada, dan perkembangan terbaru dalam domain penelitian yang terkait, khususnya tentang *generative AI*, *Large Language Model* seperti GPT, dan teknik *Retrieval Augmented Generation (RAG)*. Studi ini membantu menetapkan dasar teori dan kerangka kerja untuk penelitian.

3.2 Identifikasi Kebutuhan dan Permasalahan

Tahap ini melibatkan analisis kebutuhan dan identifikasi permasalahan spesifik yang akan ditangani oleh penelitian dimana akan dilakukan proses identifikasi batasan dan tantangan yang ada dalam implementasi model *Generative AI* dan bagaimana model *RAG* bisa menjadi solusi. Tujuan dari tahap ini adalah untuk menentukan fokus penelitian dan mengembangkan pertanyaan penelitian yang jelas dan terarah.

3.2.1 Analisis Kebutuhan

Dalam implementasi teknik *Retrieval-Augmented Generation* (RAG) untuk mengukur tingkat akurasi *large language model* berbasis *vector database*, dilakukan analisis kebutuhan yang komprehensif. Analisis yang dilakukan dibagi menjadi kebutuhan fungsional dan non-fungsional, yang mendefinisikan spesifikasi teknis dan kriteria operasional yang diperlukan untuk kesuksesan penerapan teknik ini.

A Kebutuhan Fungsional

Kebutuhan fungsional merujuk pada deskripsi layanan atau fungsi-fungsi spesifik yang dapat dilaksanakan oleh sistem selama operasi berlangsung di dalam

sistem tersebut [15]. Dalam penelitian ini, terdapat kebutuhan fungsional sebagai berikut.

1. Aplikasi dilengkapi tampilan antarmuka pengguna untuk memfasilitasi interaksi pengguna.
2. Aplikasi mampu menerima *input* berbasis teks dari pengguna.
3. Setelah menerima *input*, Aplikasi meneruskannya ke model utama untuk diproses lebih lanjut.
4. Ketika *input* pengguna sesuai dengan data dalam *database*, model akan menerapkan teknik *Retrieval-Augmented Generation* (RAG).
5. Berdasarkan teknik RAG, model memberikan tanggapan yang akurat dan tepat kepada *input* pengguna.

B Kebutuhan Non-Fungsional

Kebutuhan non-fungsional menggambarkan batasan-batasan yang berkaitan dengan layanan atau fungsi yang ada dalam sistem [15]. Dalam penelitian ini, terdapat kebutuhan non-fungsional sebagai berikut.

1. Sistem dapat diakses menggunakan perangkat mobile ataupun desktop melalui web browser.
2. Sistem ini menggunakan *vector database* sebagai sistem basis data yang menyimpan semua *knowledge base* sistem.
3. LLM yang digunakan merupakan model GPT versi 3.5.

3.3 Pemilihan Dataset

Dataset yang akan diujikan dalam penelitian ini adalah data berupa dokumen PDF yang merupakan penjabaran algoritma dan struktur data dari Universitas Birmingham, UK. Buku ini dipilih karena mencakup pembahasan algoritma dan struktur data secara lengkap dan komprehensif, serta relevan dengan topik penelitian.

Dataset ini nantinya akan dipecah menjadi bagian-bagian kecil dan disimpan ke dalam vector database. Penyimpanan dalam bentuk vektor ini bertujuan agar data

tersebut dapat dijadikan referensi di kemudian waktu. Dengan menggunakan vector database, potongan-potongan dokumen dapat dengan mudah diakses dan digunakan oleh Large Language Model (LLM) untuk menjawab pertanyaan user secara efektif.

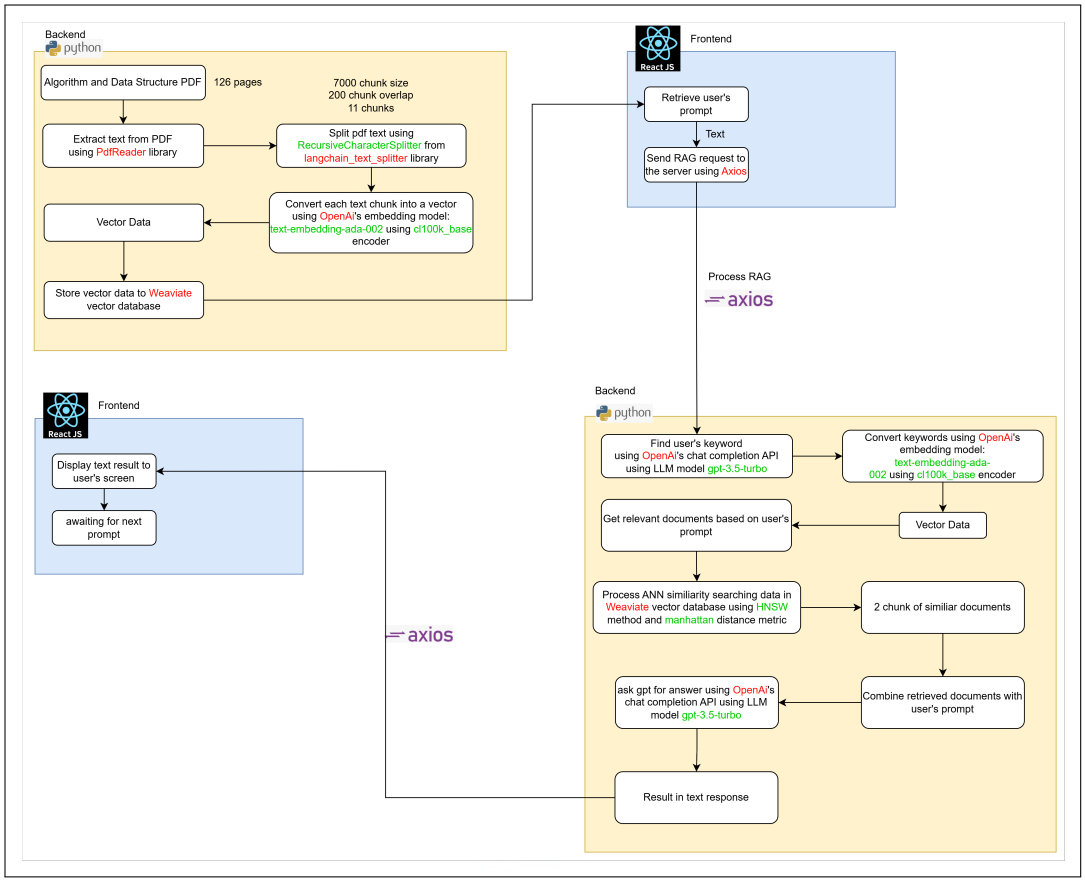
Buku ”Algoritma dan Struktur Data” yang digunakan memiliki total 126 halaman. Setiap bagian dari buku ini akan dianalisis dan diindeks dalam vector database, sehingga LLM dapat memberikan jawaban berdasarkan potongan dokumen yang relevan. Hal ini diharapkan dapat meningkatkan kemampuan LLM dalam memberikan jawaban yang akurat dan relevan sesuai dengan pertanyaan yang diajukan oleh pengguna.

3.4 Perancangan Aplikasi

Pada tahap ini, dirancang arsitektur sistem yang akan dikembangkan, termasuk bagaimana komponen-komponen sistem seperti model *RAG*, *dataset*, dan aplikasi akan bekerja bersama. Perancangan sistem meliputi pemilihan teknologi, penentuan metode interaksi antar modul, dan *User Interface*. Desain sistem ini bertujuan untuk memastikan sistem yang dikembangkan dapat memenuhi kebutuhan yang telah diidentifikasi sebelumnya.

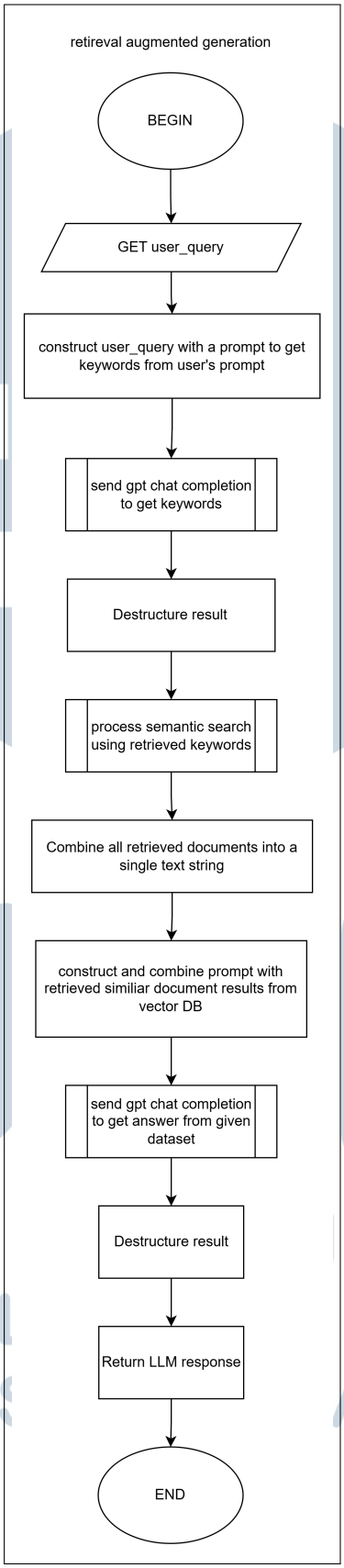
3.4.1 Rancangan RAG

Tahap awal dalam perancangan aplikasi dengan sistem RAG ini adalah dengan membuat arsitektur aplikasi terlebih dahulu. Aristektur RAG yang dibuat memiliki tiga proses besar yaitu proses *Retriveal* untuk mengumpulkan data-data yang relevan yang ada dalam database, kemudian dilanjutkan dengan proses *Augmentation* dimana data-data relevan yang telah diambil sebelumnya akan dilakukan proses penggabungan dengan prompt yang diberikan oleh user, lalu proses *Generation* yang akan menghasilkan jawaban dimana *LLM* akan menjawab prompt dari user berdasarkan pengetahuan yang ada di database. Arsitektur RAG dapat dilihat pada gambar 3.5

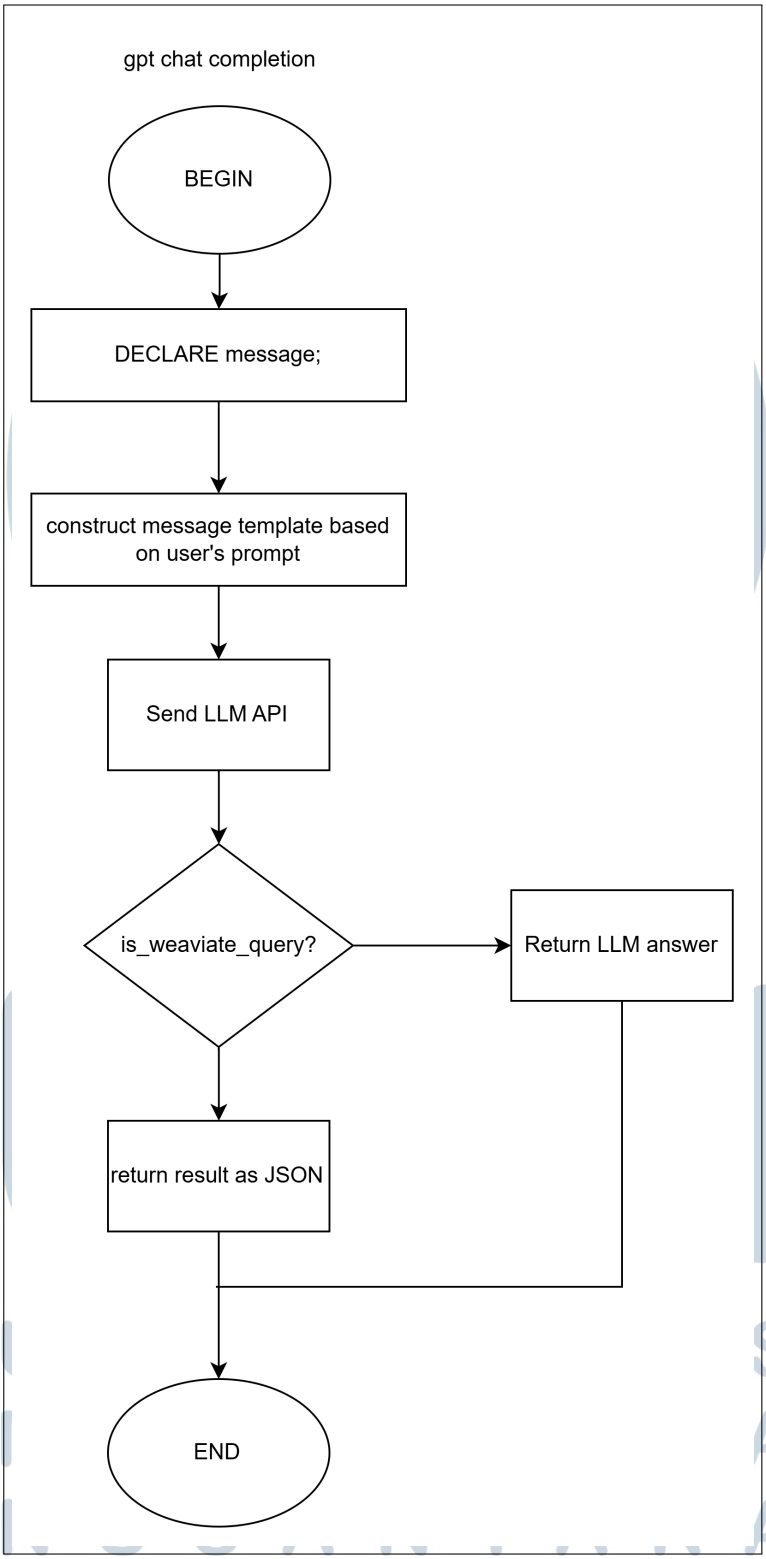


Gambar 3.1. End to end flowchart

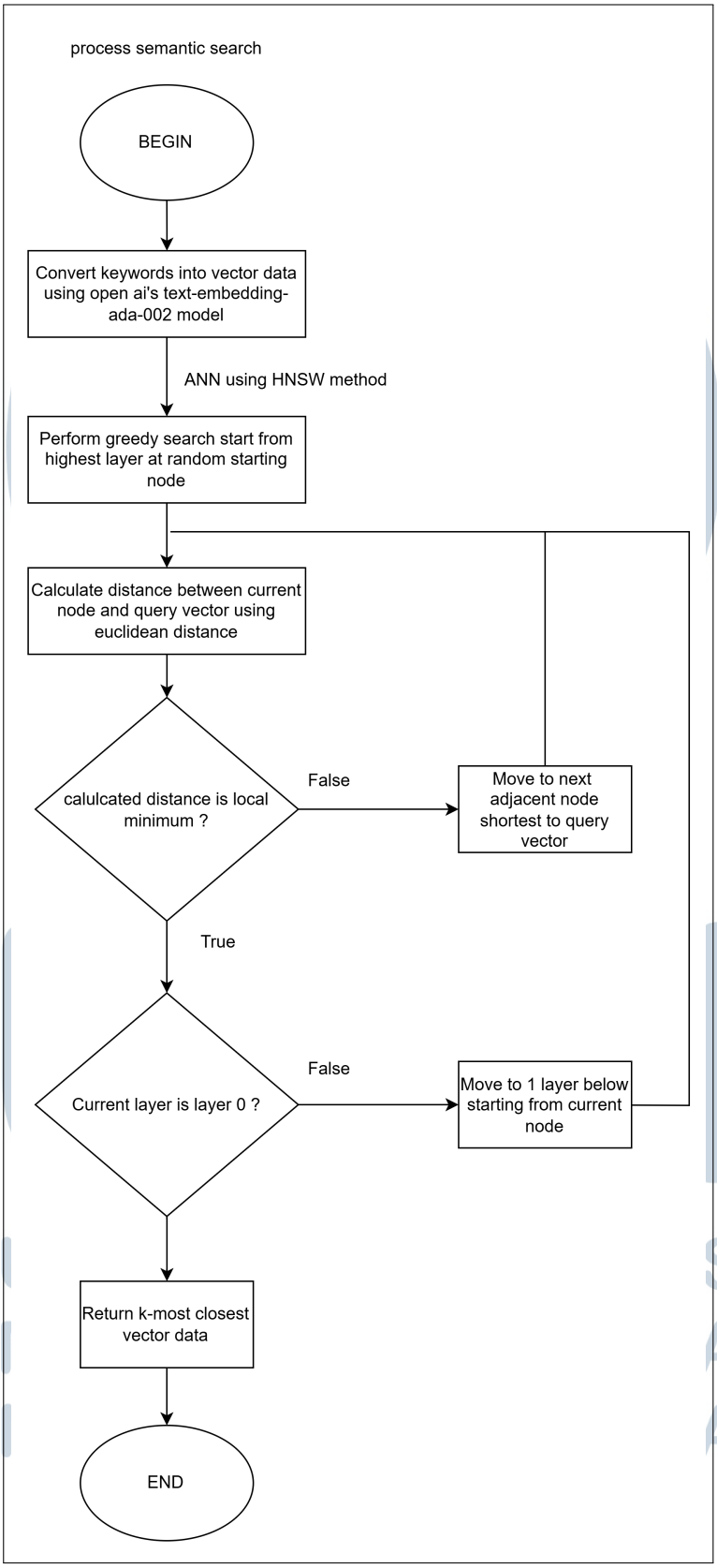
Dengan rancangan RAG yang diberikan, dapat dibuat flowchartnya sebagai berikut:



Gambar 3.2. Flowchart RAG



Gambar 3.3. GPT Chat Completion Flowchart



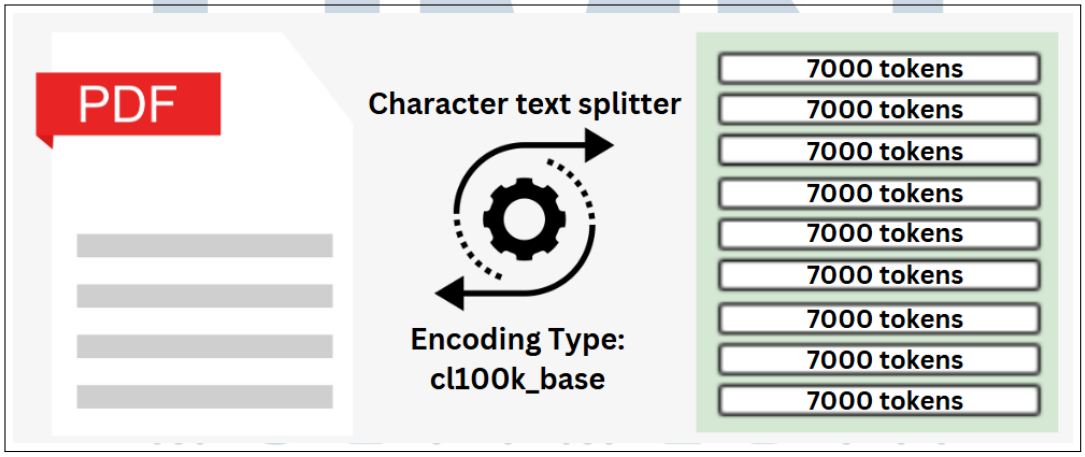
Gambar 3.4. GPT Chat Completion Flowchart

3.4.2 Teknik populasi data Vector Database

Dalam tahap ini, data yang disediakan dalam bentuk teks akan diolah menjadi vektor, kemudian vektor tersebut akan disisipkan ke dalam vector database. Proses penyisipan data ke dalam basis data vektor dimulai dengan melakukan proses chunking terlebih dahulu. Chunking adalah proses di mana teks dipotong menjadi bagian-bagian kecil. Setiap bagian ini dipisahkan menjadi bagian-bagian dengan ukuran sekitar 7000 token menggunakan *library LangChain Character Text Splitter*.

Teknik tokenisasi yang digunakan adalah metode *encoding cl100k_base*. Metode encoding ini sangat penting karena digunakan untuk proses *embedding* yang diimplementasikan pada model *embedding* milik OpenAI, yaitu *text-embedding-ada-002*. Model *embedding* ini mampu menampung hingga 8191 token untuk diproses. Hasil akhir dari proses *embedding* melalui model ini adalah data vektor yang memiliki 1536 dimensi.

Setelah proses chunking dan tokenisasi selesai, teks yang telah diubah menjadi vektor ini kemudian disisipkan ke dalam basis data vektor. Proses ini memungkinkan sistem untuk melakukan pencarian semantik dengan lebih efektif, karena data telah dioptimalkan dalam format vektor yang mendukung analisis dan pengambilan informasi secara cepat dan akurat. Dengan demikian, seluruh proses ini memastikan bahwa data yang diolah siap digunakan untuk keperluan analitik dan pencarian informasi yang lebih mendalam pada proses selanjutnya.



Gambar 3.5. Teknik text splitting

```python
def extract_text_from_pdf(pdf_path):
    text = ""
    reader = PdfReader(pdf_path)

    for page in reader.pages:
        text += page.extract_text() + "\n"

    return text.replace('\n', ' ')
```

Gambar 3.6. Source code proses read document

```python
text = extract_text_from_pdf(path)

text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    encoding_name="cl100k_base",
    chunk_size=7000,
    chunk_overlap=200
)
chunks = text_splitter.split_text(text)
print("Chunks divided into: " + str(len(chunks)) + " chunks.")
print(f"Prepare to populate vector db")
```

Gambar 3.7. Source code proses chunking document

Teks yang telah dipisahkan menjadi chunk tersebut selanjutnya akan menjalani proses konversi dari teks menjadi data yang dapat direpresentasikan dalam bentuk vektor dengan dimensi tinggi. Proses konversi chunk menjadi data vektor dilakukan menggunakan library OpenAI dengan model embedding yaitu text-embedding-ada-002. Setiap chunk teks yang diproses oleh model ini akan menghasilkan sebuah vektor yang mampu merepresentasikan arti semantik dari kalimat-kalimat yang diembed. Model embedding ini menghasilkan vektor dengan 1536 dimensi yang merepresentasikan arti semantik pada kalimat tersebut.

Setelah proses konversi selesai dan setiap chunk telah diubah menjadi vektor berdimensi 1536, langkah selanjutnya adalah menyisipkan data tersebut ke dalam basis data vektor. Penyisipan ini penting untuk memastikan bahwa data yang telah diembed dapat diakses dan dianalisis secara efisien dalam basis data. Basis data vektor memungkinkan sistem untuk melakukan pencarian dan pengambilan

informasi berdasarkan makna semantik dari teks yang telah diubah menjadi vektor. Dalam penelitian ini, model yang digunakan adalah model OpenAI text-embedding-ada-002 dengan metode perhitungan distance metric memakai metode *manhattan*.

```
weaviate_auth_api_key = os.getenv("WEAVIATE_AUTH_API_KEY")
auth_config = weaviate.AuthApiKey(api_key=weaviate_auth_api_key)
weaviate_client = weaviate.Client(
    url="https://my-sandbox-8k8cirqc.weaviate.network",
    auth_client_secret=auth_config,
    additional_headers = {
        "X-OpenAI-Api-Key": chat_gpt_api_key  # Replace with your inference API key
    }
)
```

Gambar 3.8. Inisiasi koneksi weaviate

```
class_obj = {
    "class": weaviate_schema_name,
    "vectorizer": "text2vec-openai",
    "vectorIndexConfig": {
        "distance": "manhattan"
    },
    "moduleConfig": {
        "text2vec-openai": {
            "model": "ada",
            "modelVersion": "002",
            "type": "text"
        },
        "generative-openai": {}
    }
}
```

Gambar 3.9. Inisiasi schema menggunakan distance metric manhattan

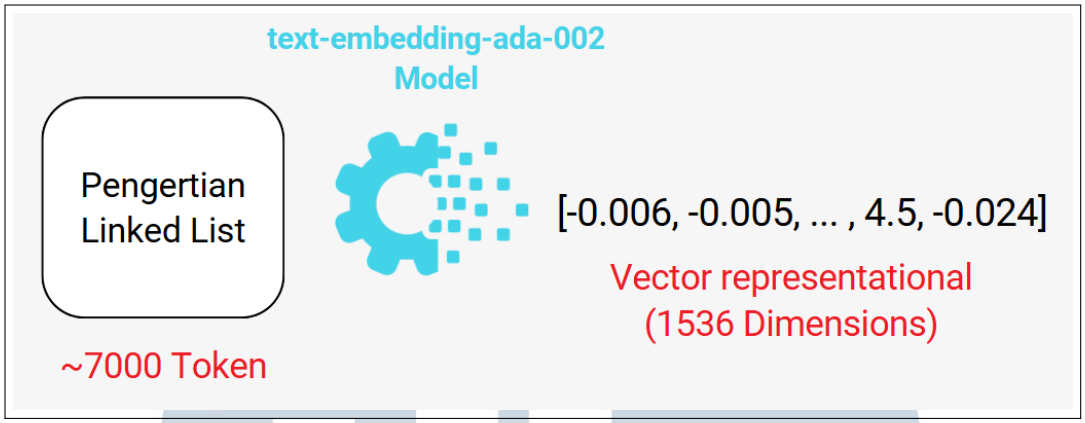
```python
weaviate_client.batch.configure(batch_size=150)  # Configure batch
with weaviate_client.batch as batch:  # Initialize a batch process
    for i, d in enumerate(chunks):  # Batch import data
        print(f"importing chunk: {i+1}")
        properties = {
            "content": d
        }
        batch.add_data_object(
            data_object=properties,
            class_name=weaviate_schema_name
        )
```

Gambar 3.10. Source code menyisipkan data kedalam vector store

```
Document 1: [0.013143454, 0.010444536, ... , -0.036710788] (1536 dimensions)
Document 2: [0.013186826, 0.021123031, ... , -0.034155823] (1536 dimensions)
Document 3: [0.0052324655, 0.027004959, ... , -0.040364735] (1536 dimensions)
Document 4: [0.0028723225, 0.017321898, ... , -0.02599638] (1536 dimensions)
Document 5: [0.010603967, 0.023564372, ... , -0.03285232] (1536 dimensions)
Document 6: [0.0042530852, 0.011127258, ... , -0.03035092] (1536 dimensions)
Document 7: [0.019063631, 0.007049142, ... , -0.049350873] (1536 dimensions)
Document 8: [-0.013595947, 0.015319377, ... , -0.022965387] (1536 dimensions)
Document 9: [-0.012029281, 0.017868135, ... , -0.04196806] (1536 dimensions)
Document 10: [0.00738346, 0.015552836, ... , -0.03995755] (1536 dimensions)
Document 11: [-0.0076415227, 0.015893824, ... , -0.03197767] (1536 dimensions)
```

Gambar 3.11. Document vector data

Dengan menyisipkan data vektor yang telah dikonversi ke dalam basis data vektor, sistem memperoleh kemampuan untuk mengakses dan memanfaatkan informasi yang telah dioptimalkan dalam format vektor berdimensi tinggi. Proses ini meningkatkan efisiensi pencarian semantik dan berguna untuk memastikan setiap informasi yang direpresentasikan dalam basis data memiliki arti semantik yang tinggi. Ini merupakan langkah krusial dalam implementasi teknik retrieval yang efektif, memastikan bahwa data yang diolah siap digunakan untuk proses selanjutnya.



Gambar 3.12. Proses konversi teks menjadi vector data

Setelah data melalui proses konversi menjadi representasi vector berdimensi 1536, data tersebut akan disisipkan ke dalam vector database. Proses ini dimulai dengan indexing data oleh vector database Weaviate. Dalam Weaviate, data yang disimpan dalam bentuk graph *Hierarchical Navigable Small World (HNSW)* akan direkonstruksi ulang. Proses rekonstruksi ini bertujuan untuk menentukan tetangga terdekat dari vektor baru yang dimasukkan dimana proses penempatan data baru pada algoritma *HNSW* akan melalui beberapa langkah yaitu:

1. Penentuan Layer Optimal

   Setiap vektor akan ditempatkan pada layer tertentu dalam graf HNSW. Layer ini ditentukan berdasarkan tingkat kedekatan vektor baru dengan vektor-vektor yang sudah ada dalam database.

2. Rekonstruksi Graph

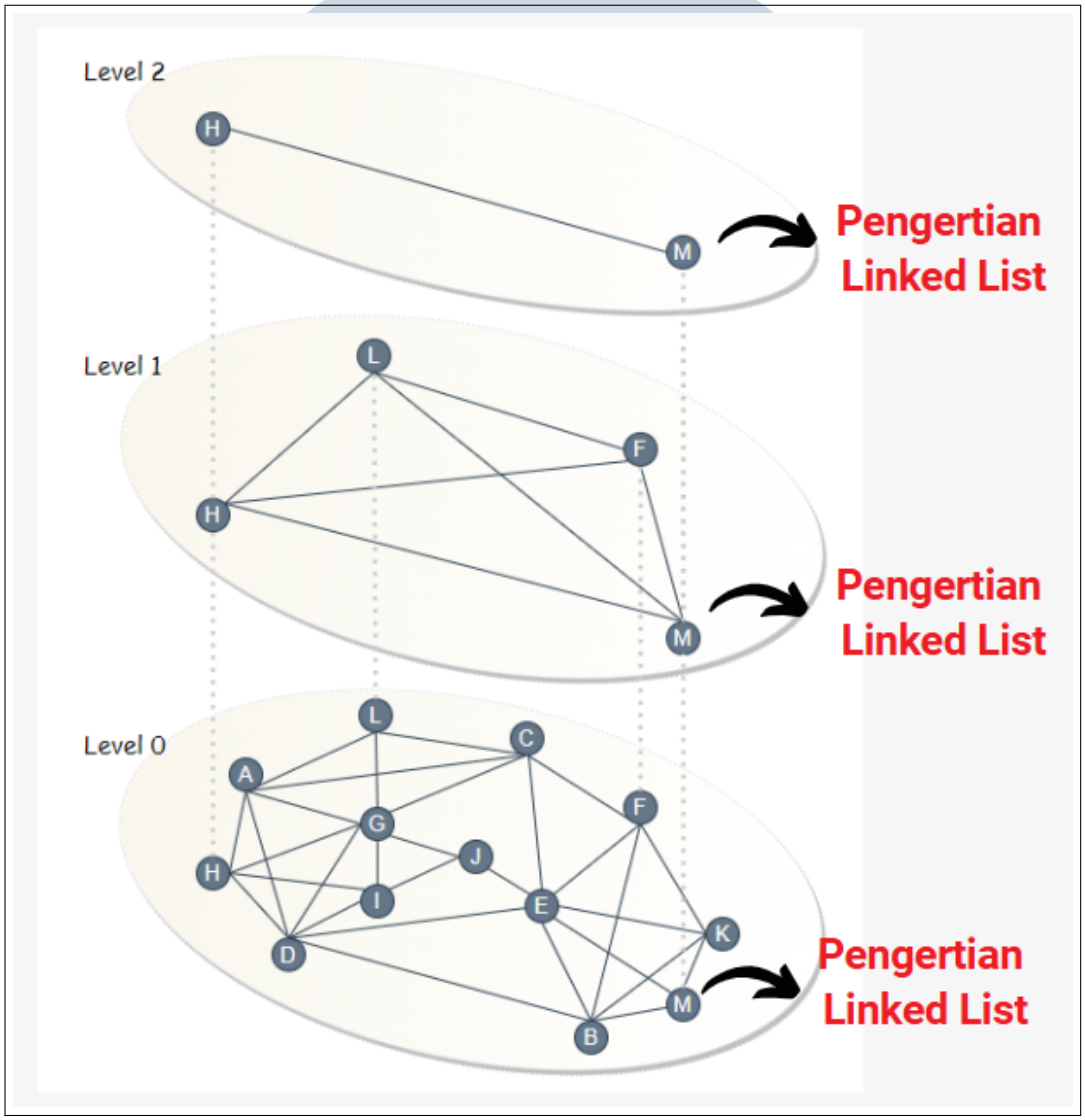
   Graph HNSW akan direkonstruksi dengan memperbarui koneksi antara vektor-vektor. Vektor baru akan terhubung dengan tetangga terdekat yang sudah ada dalam graf. Hal ini dilakukan untuk memastikan bahwa pencarian tetangga terdekat dilakukan dengan cepat dan efisien.

3. Optimalisasi Penempatan Data

   Dengan menentukan tetangga terdekat dan melakukan rekonstruksi graf, data baru ditempatkan pada posisi yang optimal. Penempatan optimal ini memastikan bahwa proses retrieval atau penarikan data akan lebih cepat karena struktur graf telah dioptimalkan untuk pencarian cepat.

Dengan proses ini, data yang telah dikonversi dan disimpan dalam vector database Weaviate akan memiliki performa pencarian yang tinggi. Hal ini sangat

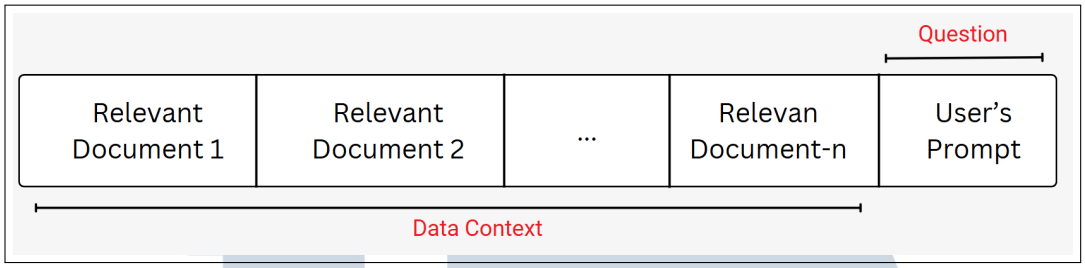
penting untuk aplikasi seperti retrieval-augmented generation, di mana kecepatan dan akurasi dalam penarikan data sangat krusial untuk mengurangi halusinasi pada *LLM*.



Gambar 3.13. Proses indexing vector database HNSW graph

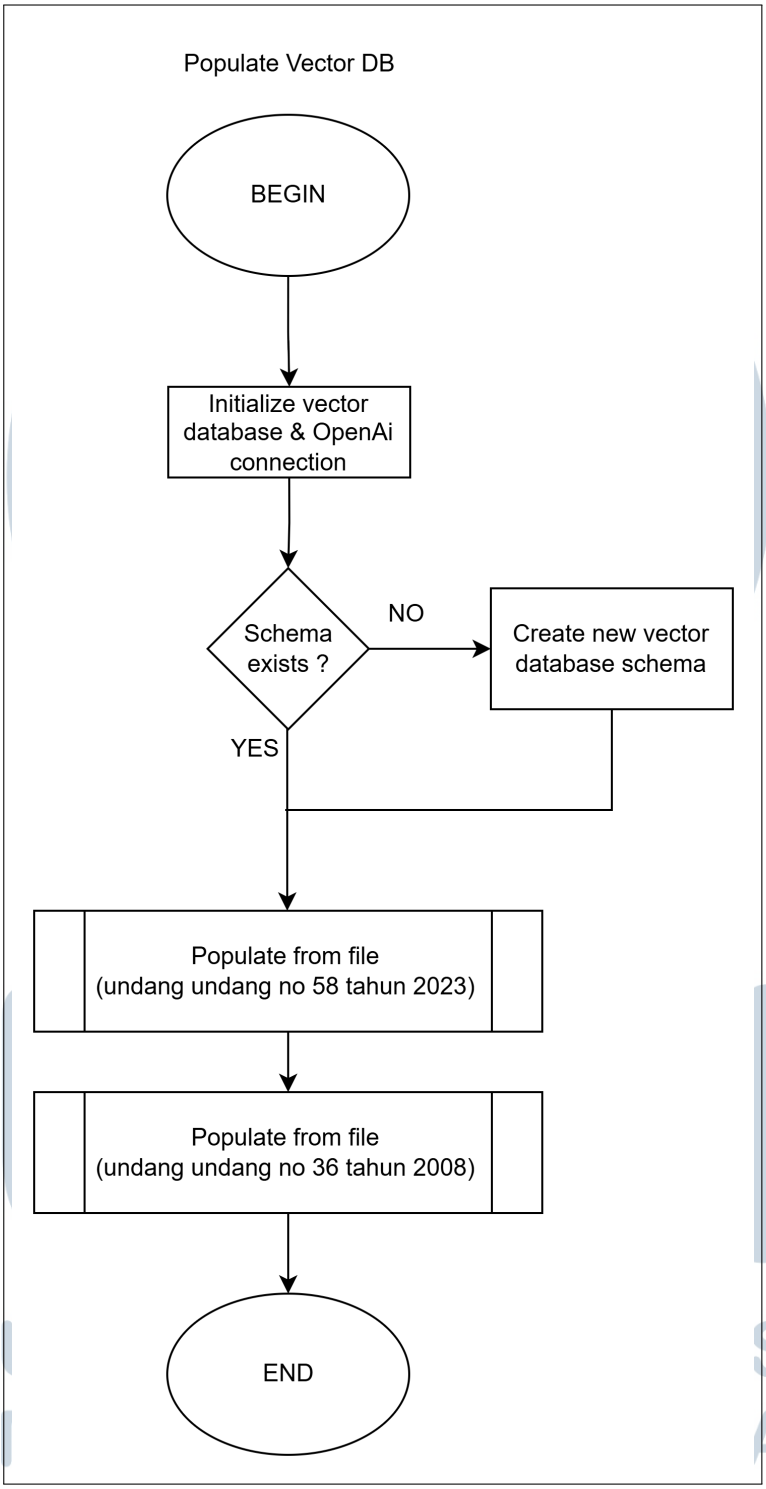
Pemilihan 7000 token dalam setiap chunk didasarkan kepada kemampuan dari model LLM OpenAI yang hanya dapat melakukan pengiriman 16000 token dalam sekali request, yang mana nantinya file-file relevan yang akan diambil dari vector database akan dicari 2 file terdekat berdasarkan representasi vector mereka yang menyisakan 2000 token untuk prompt user. Chunk-chunk tersebut nantinya akan diambil dalam proses *semantic search* dengan algoritma *ANN (Approximate Nearest Neighbor)* menggunakan metode *HNSW (Hierarchical Navigable Small World)* untuk digabungkan dengan *prompt user* untuk menghilangkan sifat

halusinasi dari LLM

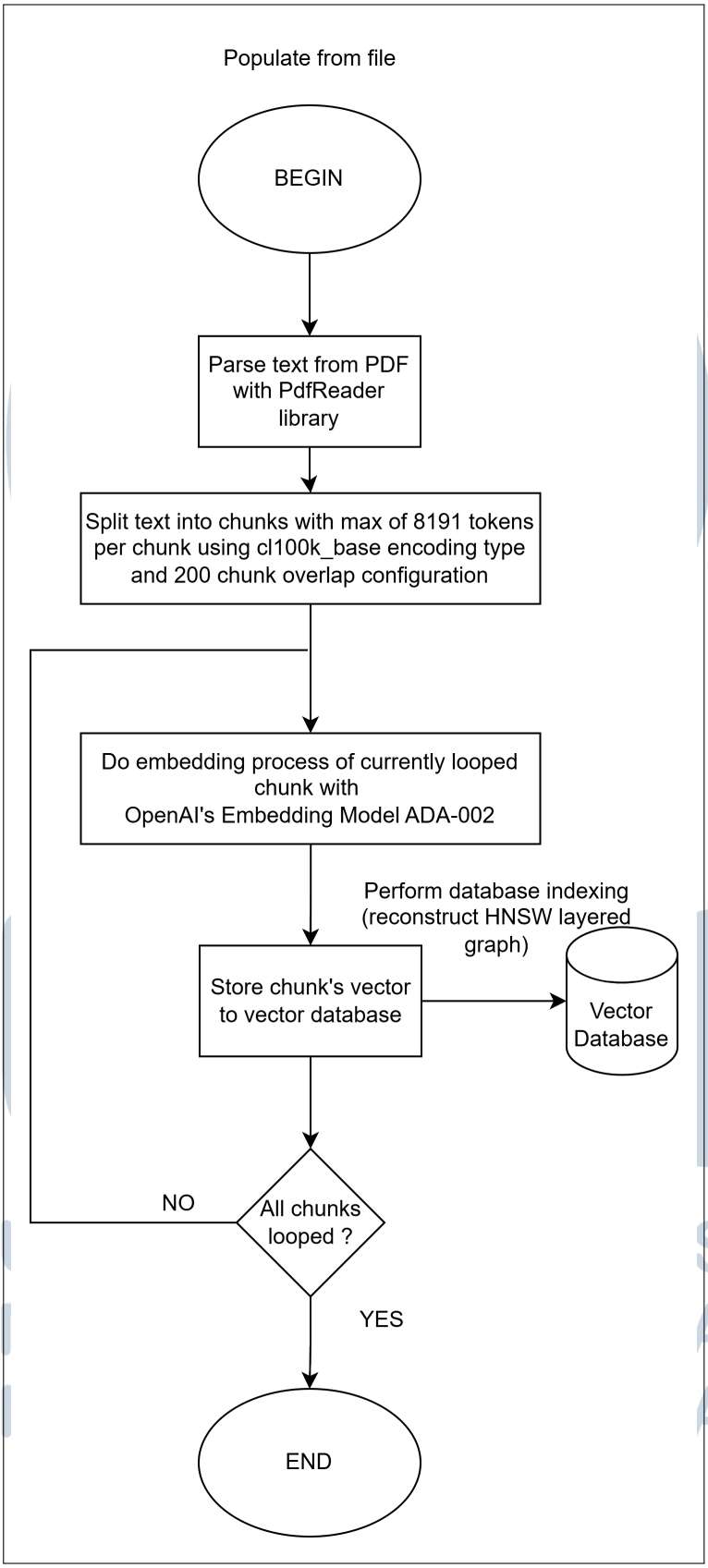


Gambar 3.14. Expectation Data Distribution

Dari rancangan metode untuk melakukan populasi data pada vector database weaviate, dapat diilustrasikan dengan flowchart berikut ini:



Gambar 3.15. Populate vector DB flowchart



Gambar 3.16. Read data from file flowchart

3.5 Implementasi Teknik RAG Untuk LLM

Implementasi melibatkan pengembangan teknik *RAG* yang menggunakan *Large Language Model (LLM)* dari *GPT 3.5*. Implementasi ini bertujuan untuk menghasilkan sebuah aplikasi yang dapat mengintegrasikan informasi yang relevan ke dalam jawaban yang dihasilkan oleh *LLM* secara akurat.

A Mencari keyword dari prompt user

Tahap awal dalam proses Retrieval-Augmented Generation (RAG) adalah mencari keyword dari prompt yang disediakan oleh user. Langkah-langkah rinci dari tahap ini adalah sebagai berikut:

1. Analisa keywords oleh LLM GPT-3.5

   Prompt yang diberikan oleh user akan dibaca oleh model language model besar (LLM) GPT-3.5. Model ini akan menganalisis teks prompt untuk mengidentifikasi kata-kata kunci (keywords) yang paling relevan.

   Berikut prompt yang digunakan untuk mencari keyword user:

```
WEAVIATE_ROLE = "You are a Vector Database Specialist. Your primary responsibility is to develop and optimize algorithms for parsing user
prompts and generating efficient queries for vector databases. Leveraging your expertise in natural language processing (NLP) and
database management, you will play a crucial role in enabling seamless interaction between users and the vector database, ensuring
accurate and relevant responses to user queries."
```

Gambar 3.17. Weaviate Role

```
WEAVIATE_PROMPT = f"""
        I have this prompt which has the information that needs to be queried in the weaviate vector database.
        Transform the prompt into an optimized query vector database keywords returned in the form of JSON. The format should follow this
        rules:

        returned answer must be a JSON object with 1 property.
        first property is `query` where the value holds an optimized query keywords to be queried to the vector database in english
        language and the data type is string.

        answer example:
        {{
            "query": "Binary tree insertion methods"
        }}

        query property is the optimized query text in english that will be queried to vector database to retrieve relevant documents.

        The data in my weaviate vector database contains data of algorithm and data structure written in english language.

        Here is the prompt:
        <user_prompt>
    """
```

Gambar 3.18. Weaviate Prompt

```
print("=== Start processing prompt: " + str(user_prompt))
weaviate_prompt = WEAVIATE_PROMPT.replace("<user_prompt>", user_prompt)
response = gpt_chat_completion(weaviate_prompt, WEAVIATE_ROLE, True)

response_obj = json.loads(response)
keywords = response_obj["query"]
print("=== Prompt keywords: " + keywords)

print("=== keyword's vector: ")
embed_text_info(keywords)
```

Gambar 3.19. Get Prompt Keywords

2. Tujuan Ekstraksi Keyword

   Ekstraksi keywords bertujuan untuk menemukan lokasi vektor yang optimal dalam vector database selama proses pencarian berikutnya. Keywords yang relevan memastikan bahwa pencarian dilakukan secara efisien dan akurat, sehingga sistem dapat mengakses data yang paling sesuai dengan permintaan user.

3. Proses Embeddings Keywords

   Setelah keywords berhasil diidentifikasi, tahap berikutnya adalah melakukan embeddings pada keywords tersebut. Proses embeddings ini mengubah keywords menjadi representasi vektor berdimensi tinggi yang dapat diproses oleh vector database. Embeddings adalah langkah penting karena memungkinkan keywords untuk dibandingkan dan dicocokkan dengan vektor lain dalam database.

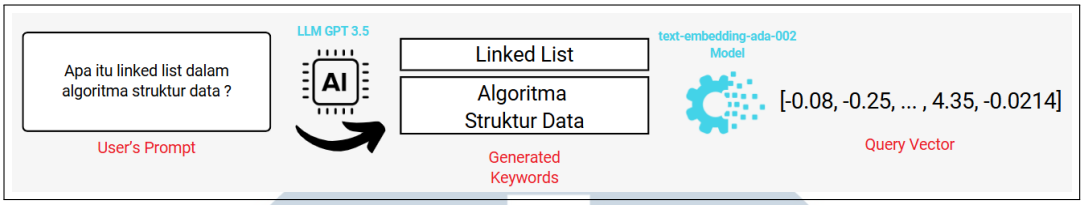
```
def embed_text_info(text, model="text-embedding-ada-002"):
    text = str(text).replace("\n", " ")
    vector = gpt_client.embeddings.create(input = [text], model=model).data[0].embedding
    print_vector_info(vector)

def print_vector_info(vector, index):
    print("Document " + str(index) + ": [" + str(vector[0]) +", "+ str(vector[1]) + ", ... , " + str(vector[len(vector) - 1]) + "] (1536
    dimensions)")
```

Gambar 3.20. Source code print vector information

4. Penentuan Lokasi Query Vector

   Vektor hasil embeddings dari keywords kemudian digunakan untuk menentukan lokasi query vector dalam vector database. Lokasi ini merupakan titik awal untuk melakukan pencarian vektor yang relevan dalam database.



Gambar 3.21. Prompt Keyword Generation

```
=== Start processing prompt: apakah analisis performa dari suatu algoritma tergantung kepada resource yang diperlukannya ?
=== Prompt keywords: Performance analysis algorithm resource dependency
=== keyword's vector:
[0.0003823210718110204, -0.004086864180862904, ... , -0.003754124976694584] (1536 dimensions)
```

Gambar 3.22. Keyword vector representation

B Semantic search retrieval menggunakan ANN dengan metode Hierarchical Navigable Small World (HNSW)

Semantic search yang digunakan untuk melakukan pencarian data melalui vector database Weaviate telah mengalami abstraksi sehingga dalam proses koding pembuatan aplikasinya hanya diperlukan untuk menentukan distance metric yang ingin dipakai. Berikut source code untuk melakukan proses semantic search pada weaviate database menggunakan bahasa pemrograman python:

```python
def semantic_search(keywords):
    response = (
        weaviate_client.query
        .get(weaviate_schema_name, ["content"])
        .with_near_text({"concepts": [keywords]})
        .with_limit(2)
        .with_additional(["distance", "vector"])
        .do()
    )

    response_data = response["data"]['Get'][weaviate_schema_name]

    print("=== Retrieved chunk 1's vector: ")
    print_vector_info(response_data[0]["_additional"]["vector"])
    print("=== Retrieved chunk 2's vector: ")
    print_vector_info(response_data[1]["_additional"]["vector"])
    print("=== Distance to retrieved chunk 1 with manhattan calculation metric: " + str(response_data[0]["_additional"]["distance"]))
    print("=== Distance to retrieved chunk 2 with manhattan calculation metric: " + str(response_data[1]["_additional"]["distance"]))

    response_data = map(lambda x: x['content'].replace('\n', ''), response_data)
    response_data = list(response_data)

    return response_data
```

Gambar 3.23. Weaviate Semantic Search

```
=== Start processing semantic search with limit: 2
=== Retrieved chunk 1's vector:
[0.019063631, 0.007049142, ... , -0.049350873] (1536 dimensions)
=== Retrieved chunk 2's vector:
[0.0042530852, 0.011127258, ... , -0.03035092] (1536 dimensions)
=== Distance to retrieved chunk 1 with manhattan calculation metric: 22.649996
=== Distance to retrieved chunk 2 with manhattan calculation metric: 22.65222
```

Gambar 3.24. Hasil Semantic Search

Hierarchical Navigable Small World (HNSW) adalah salah satu metode yang digunakan dalam algoritma Approximate Nearest Neighbor (ANN) yang diterapkan dalam vector database Weaviate. Metode ini menggunakan struktur berbasis graph untuk perhitungannya. HNSW dimanfaatkan untuk melakukan pencarian data vektor dengan menemukan data vektor lain yang terdekat dari lokasi vektor pencarian.

Proses pencarian dalam HNSW terdiri dari navigasi melalui beberapa lapisan (layers) graph untuk menemukan elemen yang paling mendekati query. Berikut adalah penjelasan detail mengenai proses pencarian dalam HNSW:

1. Lapisan tertinggi layer graph (Topmost layer)

   Proses pencarian dimulai dari lapisan tertinggi dalam graph HNSW. Pada lapisan ini, jumlah elemen vektor relatif sedikit sehingga pencarian dapat dilakukan lebih cepat karena lebih sedikit elemen yang perlu dihitung.

   Titik awal pencarian biasanya adalah node yang dipilih secara acak atau node yang terakhir ditambahkan ke dalam struktur graph HNSW.

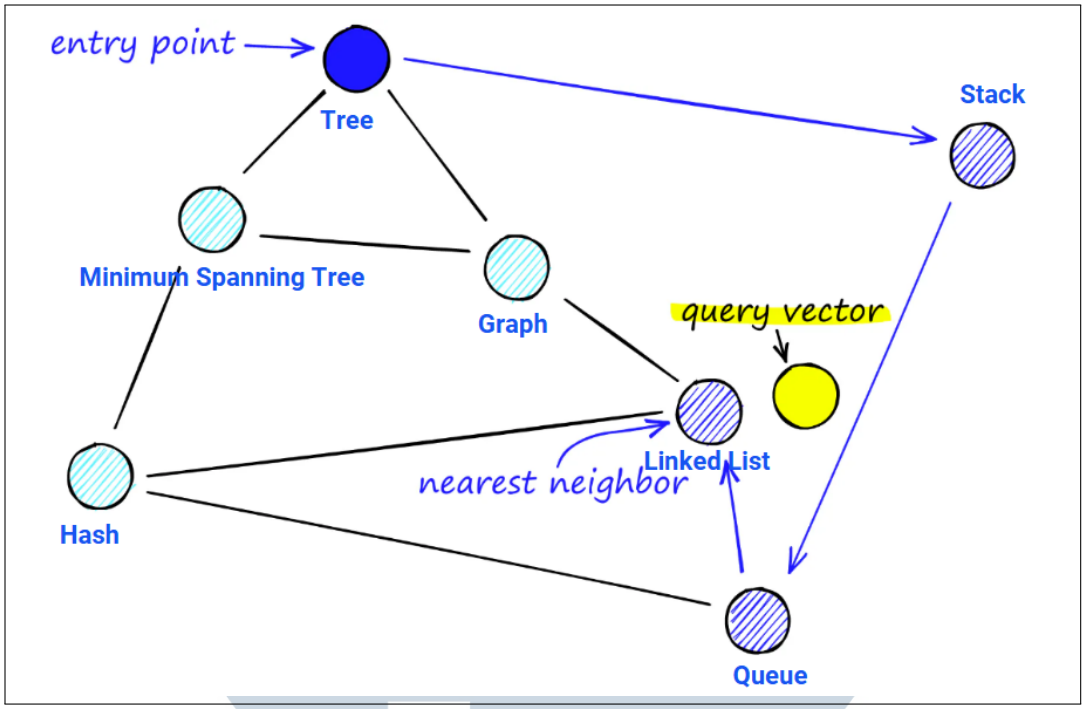
2. Greedy Search

   Setelah menemukan titik awal pada graph, proses dilanjutkan dengan pencarian greedy pada layer tersebut. Greedy search bertujuan untuk menemukan titik vektor dalam layer yang paling dekat dengan lokasi vektor pencarian.

   Setiap langkah greedy search melibatkan perhitungan jarak antar vektor menggunakan distance metric manhattan. Rumus perhitungan distance menggunakan metode manhattan adalah sebagai berikut:

$$d(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} |p_i - q_i| \tag{3.1}$$

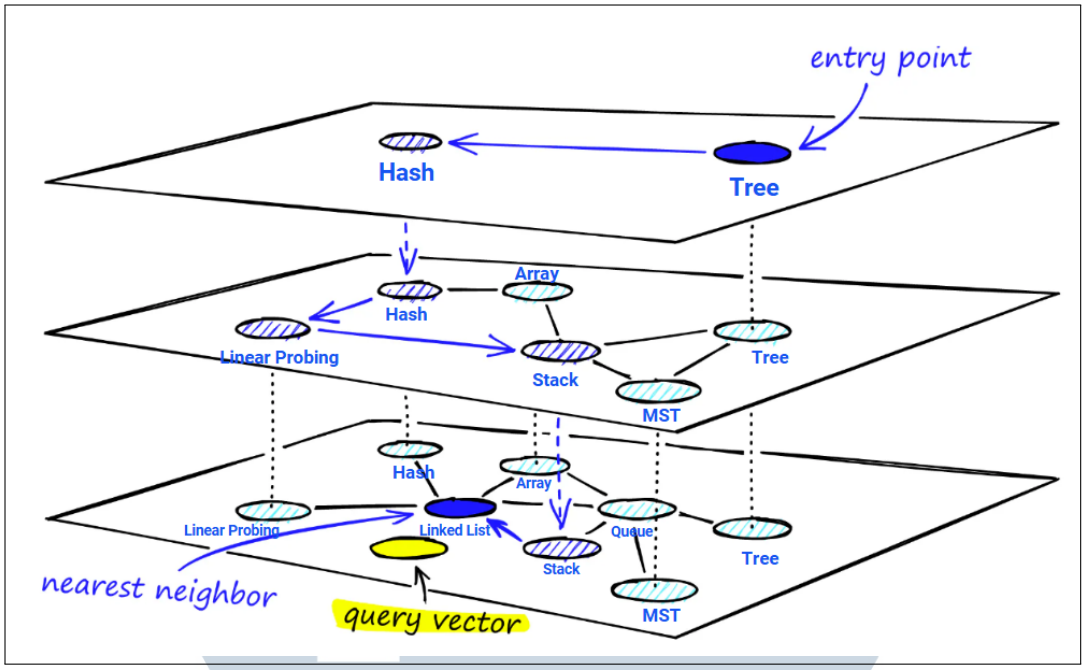
   Proses greedy search terus dilakukan sampai mencapai local minimum, yaitu kondisi di mana tidak ada vektor dalam layer tersebut yang lebih dekat dengan vektor pencarian dibandingkan dengan vektor yang sedang diperiksa.



Gambar 3.25. Greedy Search in Navigable Small World

3. Local minimum

   Jika pada suatu layer telah mencapai local minimum, maka proses pencarian akan dilanjutkan pada titik tersebut namun pindah ke satu layer dibawahnya. Proses ini akan terus dilakukan dan diulang menggunakan algoritma greedy search sampai berada di layer level dasar dan menemukan titik vector terdekat dengan lokasi vector pencarian.



Gambar 3.26. Hierarchical Navigable Small World Graph

C Augmentation dan Generation

Setelah berhasil mendapatkan dokumen yang relevan dari prompt pengguna, tahap selanjutnya adalah melakukan proses augmentasi. Proses ini melibatkan penggabungan antara prompt pengguna dengan data relevan yang telah diperoleh dari basis data vektor selama pencarian semantik. Proses augmentasi sangat penting karena memastikan bahwa informasi yang diberikan tidak hanya akurat, tetapi juga komprehensif dan kontekstual.

Setelah prompt pengguna dan dokumen relevan berhasil digabungkan menjadi satu prompt yang koheren, langkah berikutnya adalah mengirimkan prompt yang telah diperkuat dengan data relevan tersebut ke *LLM GPT-3.5*. Model GPT-3.5 akan memproses prompt ini dan memberikan jawaban yang akurat berdasarkan data yang disediakan. Dengan adanya data yang relevan dan kontekstual yang telah diintegrasikan sebelumnya, LLM mampu menghasilkan respons yang lebih tepat dan informatif.

Berikut prompt dan kode yang digunakan untuk melakukan pemanggilan API ChatGPT untuk mendapatkan jawaban relevan sesuai dengan dokumen yang dibutuhkan oleh user.

```
CHATBOT_ROLE = "You are an Algorithm and Data Structure expert, your primary responsibility is to provide accurate, reliable, and
up-to-date information on Algorithm and Data Structures knowledge where the relevant information will be provided along with the prompt."
```

Gambar 3.27. Chatbot Role Template

```
CHATBOT_PROMPT = """
    I have this prompt which you must answer based on the given contexts.

    your knowledge will be limited to these relevant context below:
    <relevant_information>.

    Jangan mencoba untuk menjawab jika jawaban tidak ada dalam informasi yang disediakan. cukup jawab dengan jawaban ketidaktahuan.
    jangan dijawab dengan pengetahuan anda.

    Here is the prompt:
    <user_prompt>
"""
```

Gambar 3.28. Chatbot Prompt Template

```
relevant_data = semantic_search(keywords)
relevant_data = ".".join(relevant_data)

# print("=== Retrieved relevant data: " + relevant_data)
chatbot_prompt = CHATBOT_PROMPT.replace("<relevant_information>", relevant_data).replace("<user_prompt>", user_prompt)

print("=== Start asking GPT for: " + user_prompt)
response = gpt_chat_completion(chatbot_prompt, CHATBOT_ROLE, False)

print("=== Response from gpt: " + response)
```

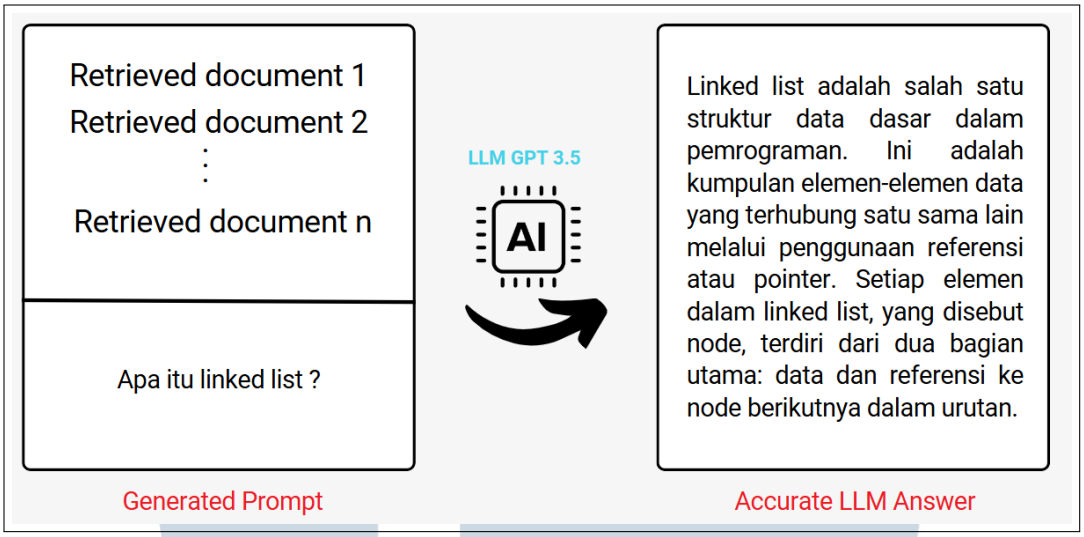
Gambar 3.29. Source Code proses augmentasi dan generasi

```
def gpt_chat_completion(prompt, role, is_weaviate_query, model = "gpt-3.5-turbo"):
    messages = [
        {
            "role": "system",
            "content": role
        },
        {
            "role": "user",
            "content": prompt
        },
    ]

    if(is_weaviate_query):
        return gpt_client.chat.completions.create(
            model = model,
            messages=messages,
            response_format={ "type": "json_object" }
        ).choices[0].message.content

    return gpt_client.chat.completions.create(
        model = model,
        messages=messages,
    ).choices[0].message.content
```

Gambar 3.30. Source Code Pengiriman API menggunakan API OpenAI



Gambar 3.31. Hasil augmentasi dan generasi LLM

3.6 Integrasi dengan Aplikasi

Setelah model RAG berhasil diimplementasikan, langkah selanjutnya adalah mengintegrasikannya dengan aplikasi yang diinginkan. Aplikasi yang akan dibuat nantinya berupa aplikasi tanya jawab dimana AI akan menjawab pertanyaan yang diberikan oleh pengguna dan akan berusaha untuk menjawabnya dengan teknik Retrieval Augmented Generation yang sudah dikembangkan sebelumnya. Tahap ini melibatkan pengembangan User Interface untuk pengguna dengan backend untuk proses data dan memberikan respon kepada pengguna.

3.6.1 React JS untuk pengembangan frontend

Frontend aplikasi dikembangkan menggunakan React.js, sebuah library JavaScript yang populer untuk membangun antarmuka pengguna (UI) yang dinamis dan responsif. React.js memungkinkan layout-layout yang dibuat dalam *User Interface* menjadi bagian-bagian kecil yang dapat digunakan kembali sehingga memudahkan pengembangan *UI*.

3.6.2 Http Request

Axios digunakan sebagai *library* untuk menangani *HTTP request* dari *frontend React* ke *backend Flask*. *Axios* memungkinkan aplikasi untuk mengirimkan dan menerima data dari server dengan mudah dan fleksibel.

3.6.3 Flask sebagai Backend API

Backend aplikasi dibangun menggunakan *Flask*, sebuah *microframework Python* yang ringan dan fleksibel. *Flask* digunakan untuk membuat *API* yang menerima permintaan dari *frontend*, memproses data, dan mengembalikan hasilnya.

1. Endpoint API

   Endpoint yang disediakan oleh backend di *Flask* yaitu endpoint yang dapat menangani penerimaan prompt dari pengguna, memproses data menggunakan *vector database* dengan algoritma *ANN (Approximate Nearest Neighbor)* dengan *HNSW (Hierarchical Navigable Small World)*, dan mengembalikan hasilnya.

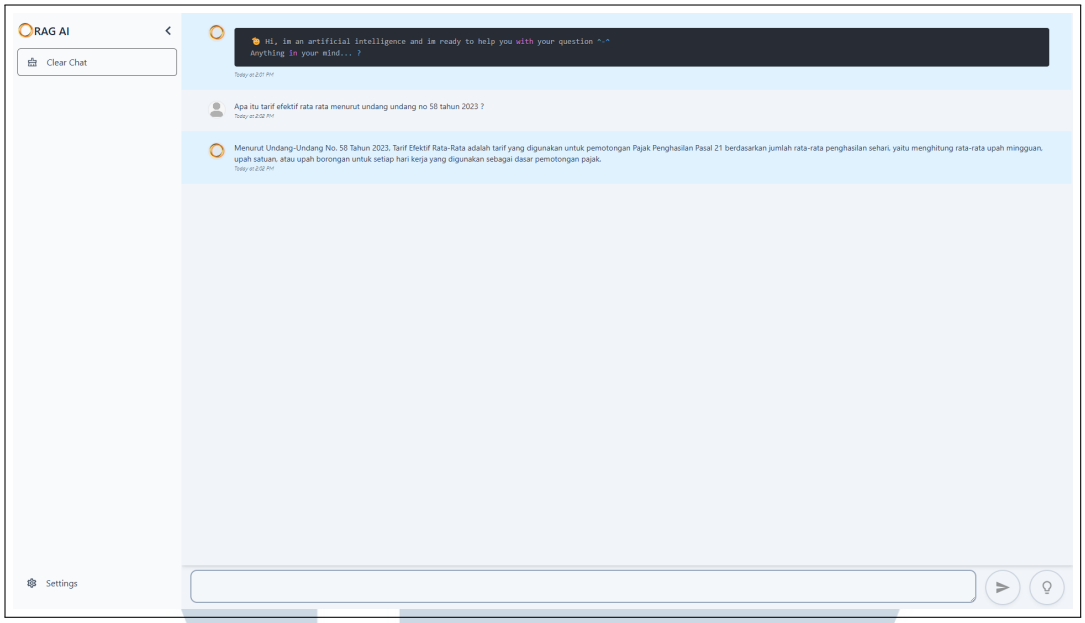
2. Flask-CORS

   Untuk mengizinkan (cross-origin requests) dari frontend React ke backend Flask, library *Flask-CORS* digunakan. Library ini memastikan bahwa frontend dan backend dapat berkomunikasi meskipun berjalan di domain yang berbeda.

3.6.4 Desain mockup frontend

Mockup yang dibangun memberikan gambaran visual tentang bagaimana antarmuka pengguna akan terlihat dan berfungsi. Mockup ini membantu proses *development* menjadi lebih teratur serta dapat memvisualisasikan alur interaksi pengguna dengan aplikasi, dari memasukkan prompt hingga melihat hasil. Hal ini membantu dalam merancang pengalaman pengguna (UX) yang intuitif dan efisien.

Berikut mockup yang akan dibangun dalam aplikasi tanya jawab *Retrieval Augmented Generation* ini:



Gambar 3.32. Mockup Aplikasi

3.7 Pengujian dan Evaluasi

Pengujian dan evaluasi dilakukan untuk memastikan sistem bekerja sesuai dengan kebutuhan dan spesifikasi yang telah ditetapkan. Proses ini mencakup pengujian fungsionalitas, performa, dan keakuratan model dalam menghasilkan teks. Evaluasi dilakukan melalui metode kuantitatif dengan mengukur tingkat akurasi sistem RAG (Retrieval Augmented Generation) menggunakan *Large Language Model* lainnya.

3.7.1 Metode Evaluasi

Hasil pengujian akan menunjukkan tingkat keakuratan yang dapat dicapai oleh sistem tanya jawab LLM dengan teknik *Retrieval Augmented Generation* yang menggunakan *vector database*. Pengujian dilakukan dengan menggunakan end-to-end testing dengan *framework RAG evaluation tools - DeepEval*.

Metode evaluasi yang digunakan dalam penelitian ini adalah metode pengukuran skor *correctness* dengan *G-Eval* menggunakan framework *DeepEval*.

Dataset algoritma dan struktur data yang sebelumnya sudah disimpan kedalam vector database sebesar 7000 chunk akan dilakukan analisa pertanyaan yang potensial dapat digunakan untuk mengukur tingkat *correctness* dalam aplikasi *RAG* ini secara otomatis. Berikut daftar pertanyaan yang ter-generate oleh *DeepEval* dapat dilihat pada tabel 3.1:

```python
def generate_golden_csv_file():
    print("Start generating golden and store it to csv file.")
    print("Process semantic search")
    list_text = semantic_search()

    print("Generating goldens")
    synthesizer = generate_goldens(list_text)

    print("Saving result as csv file")
    saved_path = synthesizer.save_as(file_type="json", directory="./synthetic_data")
    print("Saved path: " + saved_path)
```

Gambar 3.33. Generate Question source code 1

```python
def semantic_search():
    response = (
        weaviate_client.query
        .get(weaviate_schema_name, ["content"])
        .do()
    )

    response_data = response["data"]['Get'][weaviate_schema_name]
    response_data = map(lambda x: x['content'].replace('\n', ' '), response_data)
    response_data = list(response_data)

    return response_data
```

Gambar 3.34. Generate Question source code 2

```python
def generate_goldens(list_text):
    iterable_text = map(lambda x: [x], list_text)
    list_text = list(iterable_text)
    synthesizer = Synthesizer(model="gpt-3.5-turbo")
    synthesizer.generate_goldens(
        contexts=list_text,
        include_expected_output=True,
        max_goldens_per_context=1
    )

    return synthesizer
```

Gambar 3.35. Generate Question source code 3

Tabel 3.1. Daftar Pertanyaan

No	Pertanyaan
1	Imagine a scenario where you need to find the next number in the Fibonacci sequence. How would you apply the recursive definition to determine the value?
2	Contrast the main focuses and impacts of the concept of a minimal spanning tree.
3	How does Dijkstra's algorithm's time complexity relate to its performance in graph traversal?
4	What key data structures and algorithms are discussed in the lecture notes?
5	How are arrays typically utilized for data storage in computer science algorithms?
6	How do complexity considerations, stability, and partitioning strategies differ among Selection Sort, Bubble Sort, and Insertion Sort?
7	Explain the working mechanism of Treesort in sorting algorithms.
8	How does sorting algorithm stability influence the reliability of sorting outcomes in practice?
9	What is the average-case time complexity of Quicksort in terms of comparisons?
10	Compare different strategies for choosing the pivot in Quicksort to enhance its efficiency.
11	List the types of trees in computer science and their defining characteristics.
12	Compare the definitions of quad-trees and binary trees, focusing on their structure and rules of construction.
13	How does a quad-tree partition a two-dimensional space and store grey-value pictures efficiently?
14	How might understanding the concept of shortcuts influence the efficiency of Dijkstra's algorithm?
15	How is a graph implemented in computer science using an array-based approach?
16	Compare the array-based representations of graphs for unweighted and weighted graphs.

No	Pertanyaan
17	How do directed and undirected graphs differ in terms of connectivity and edges?
18	What is the representation of a graph using an adjacency matrix in an array-based implementation?
19	How can the Travelling Salesman Problem be defined and what are its implications in graph theory?
20	How does the decision tree in Bubble Sort determine the number of comparisons needed?
21	Imagine the maximum number of comparisons Selection Sort makes in the worst case scenario.
22	What is the time complexity of Selection Sort compared to Bubble Sort and Insertion Sort?
23	What is the average height of a binary tree, and how can it be calculated efficiently?
24	Compare the relationship between the height and size of a binary tree to determine the maximum number of nodes.
25	How does the binary search tree algorithm ensure efficient search operations for locating values?
26	How can the validity of a binary tree as a binary search tree be verified?
27	Examine the process of sorting items in ascending order using binary search trees.
28	Describe the recursive structure of a binomial tree and its node ordering properties.
21	Compare the merge operation in binomial heaps with other main operations in the data structure.
30	What is the total number of nodes in a binomial heap with a specific order?

3.7.2 Penilaian

Penilaian akurasi aplikasi RAG dilakukan oleh LLM lain, yaitu model GPT-3.5 Turbo. Metode penilaian menggunakan ”LLM as Judge” (LLM sebagai penilai) ini umum digunakan dalam evaluasi sistem Retrieval Augmented Generation [16]. Metode yang digunakan dalam pengetesan ini adalah metode penilaian dengan *G-Eval* dengan *source code* sebagai berikut:

```
def generate_test_case_actual_output(input_file_name, output_file_name):
    dataset = EvaluationDataset()
    dataset.add_test_cases_from_json_file(
        file_path=input_file_name,
        input_key_name="input",
        actual_output_key_name="actual_output",
        expected_output_key_name = "expected_output",
        context_key_name = "context"
    )

    json_data = list()
    for index, item in enumerate(dataset.test_cases):
        response = test_eval_gpt(item.input)
        item.actual_output = response["response"].replace("\n", " ")
        item.retrieval_context = response["retrieval_context"]

        json_data.append({
            "input": item.input,
            "actual_output": item.actual_output,
            "expected_output": item.expected_output,
            "context": item.context,
            "retrieval_context": item.retrieval_context,
            "source_file": None
        })

    import json
    with open(output_file_name, 'w', encoding='utf-8') as f:
        json.dump(json_data, f, ensure_ascii=False, indent=4)
```

Gambar 3.36. Source code proses generate actual output aplikasi RAG

```python
def do_test_case(input_file_name: str, iteration: int, distance_metric: str, chunk_size: str):
    dataset = EvaluationDataset()
    dataset.add_test_cases_from_json_file(
        file_path=input_file_name,
        input_key_name="input",
        actual_output_key_name="actual_output",
        expected_output_key_name = "expected_output",
        context_key_name = "context"
    )

    correctness_metric = GEval(
        name="Correctness",
        criteria="Determine whether the actual output is factually correct based on the expected output.",
        evaluation_params=[LLMTestCaseParams.INPUT, LLMTestCaseParams.ACTUAL_OUTPUT],
        model="gpt-3.5-turbo"
    )

    evaluate(dataset, [correctness_metric], hyperparameters={
        "distance_metric": distance_metric,
        "chunk_size": chunk_size,
        "chunk_overlap": 200,
        "embedding_model": "text-embedding-3-small",
        "iteration": iteration,
        "model": "gpt-3.5-turbo",
        "prompt template": """...
    })
```

Gambar 3.37. Source Code pengetesan G-Eval menggunakan DeepEval

Dengan pendekatan ini, diharapkan dapat diperoleh gambaran tingkat akurasi jika parameter tertentu digunakan.

3.8 Dokumentasi (Penulisan Laporan)

Langkah terakhir dalam penelitian ini adalah penulisan laporan. Laporan akan disusun berdasarkan hasil yang didapatkan melalui rangkaian proses yang sudah dilakukan sebelum-sebelumnya sehingga penelitian ini dapat disajikan dengan jelas dan sistematis. laporan juga akan mencakup uraian tentang metodologi penelitian yang digunakan. Semua informasi ini akan dirangkum dengan sebaik mungkin dalam laporan untuk memberikan pemahaman yang komprehensif tentang penelitian ini kepada pembaca di kemudian hari.

3.9 Spesifikasi Sistem

3.9.1 Software

1. Linux Operating System
2. Visual Studio Code
3. Python

4. Open-AI API
5. React JS + Vite

3.9.2 Hardware

1. Processor Intel i3 Gen 10
2. Nvidia GTX 1660 Super Graphic Card
3. 16 GB RAM