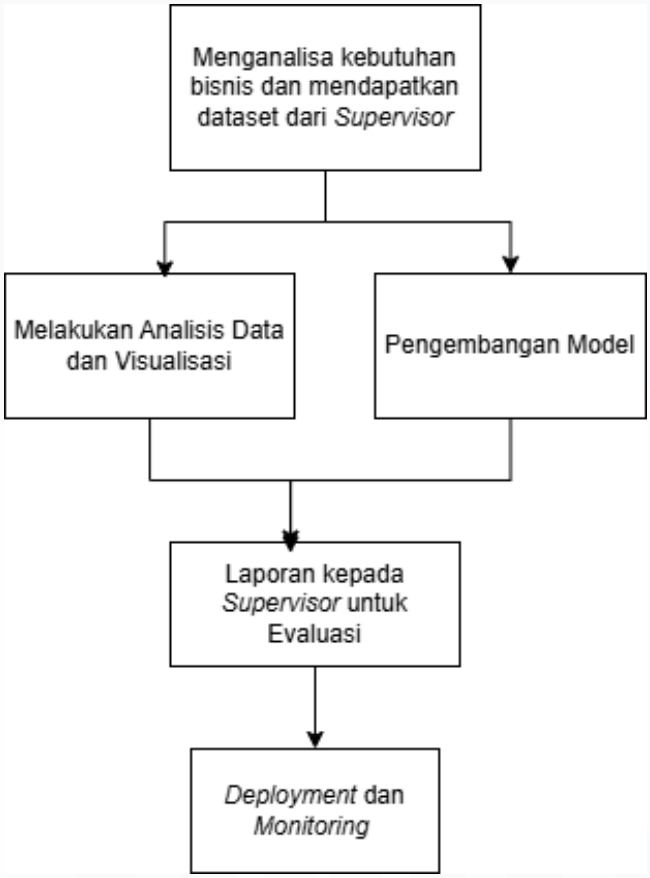
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi



Gambar 3.1 Alur Pekerjaan & Koordinasi Magang

Gambar 3.1 menjelaskan menggambarkan proses kolaboratif dan iteratif yang dilakukan selama masa magang, dengan tujuan menghasilkan solusi berbasis data yang bermanfaat bagi perusahaan. Alur ini dimulai dengan tahap Kedudukan dan Koordinasi, di mana peserta magang memulai pekerjaannya dengan memahami secara menyeluruh kebutuhan bisnis yang ada. Dalam tahap ini, peserta magang melakukan komunikasi dan diskusi intensif dengan Supervisor atau pembimbing lapangan untuk memperoleh arahan dan akses terhadap dataset yang relevan.

Koordinasi ini sangat penting karena menjadi landasan agar pekerjaan yang dilakukan benar-benar sesuai dengan kebutuhan perusahaan dan memiliki dampak yang nyata terhadap proses bisnis.

Setelah data tersedia, proses berlanjut ke tahap Analisis Data dan Visualisasi. Di sini, peserta magang mulai mengeksplorasi dataset yang telah diberikan menggunakan berbagai teknik analisis statistik atau eksploratif. Tujuannya adalah untuk menemukan pola, tren, anomali, atau insight penting dari data yang dapat dijadikan dasar pengambilan keputusan. Hasil dari eksplorasi ini tidak hanya disampaikan dalam bentuk tulisan, tetapi juga divisualisasikan menggunakan grafik, diagram, atau dashboard agar lebih mudah dipahami oleh pihak terkait, termasuk yang tidak memiliki latar belakang teknis.

Tahap lainnya adalah pengembangan model, yang dilakukan apabila hasil analisis menunjukkan bahwa dibutuhkan solusi lanjutan seperti model prediksi atau sistem rekomendasi. Pada tahap ini, diterapkan teknik *machine learning* atau model statistik untuk membangun solusi yang dapat memproyeksikan hasil di masa depan atau mengotomatisasi proses tertentu. Pengembangan model ini bersifat eksperimental, dan diharapkan melakukan iterasi dan pengujian untuk mencapai model dengan performa terbaik.

Model atau hasil analisis yang sudah dikembangkan kemudian dilaporkan kepada Supervisor pada tahap laporan kepada *Supervisor* untuk evaluasi. Dalam fase ini, dilakukan peninjauan menyeluruh terhadap hasil kerja intern, termasuk validasi terhadap asumsi, metode yang digunakan, dan akurasi dari model yang dibuat. *Feedback* dari *Supervisor* sangat krusial untuk melakukan penyempurnaan terhadap hasil kerja dan memastikan bahwa solusi tersebut siap digunakan dalam konteks bisnis nyata.

Terakhir, apabila solusi yang dibuat telah disetujui, proses masuk ke tahap deployment dan monitoring. Pada tahap ini, model atau sistem yang dikembangkan

diimplementasikan dalam lingkungan operasional perusahaan. Proses deployment ini bisa melibatkan integrasi ke dalam sistem IT perusahaan atau pengujian lebih lanjut di lingkungan pengguna. Setelah deployment, dilakukan monitoring untuk memastikan performa solusi tetap optimal dan sesuai harapan. Bila ditemukan kendala atau penyimpangan, maka proses sebelumnya dapat diulangi untuk melakukan penyesuaian.

Dalam melaksanakan segala tugas yang diberikan, bimbingan dilakukan oleh Arthur Pello, yang juga menjabat sebagai *Head of Engineering*. Selain itu, pengawasan umum dalam kegiatan magang ini berada di bawah koordinasi Garth Parlimbangan, selaku *General Manager Business & IT*.

3.2 Tugas dan Uraian Kerja Magang

3.2.1 Tugas

Selama masa magang, banyak tugas yang berfokus pada pengolahan data, pembuatan visualisasi, serta pengelolaan database. Dalam menyelesaikan setiap tugas, digunakan sejumlah tools yang saling terintegrasi, yaitu Google Colab, Metabase, PostgreSQL, *framework Langchain* dan Visual Studio Code. Berikut penjabaran rinci mengenai peran masing-masing tools dalam kegiatan magang:

1. Google Colab

Merupakan tools utama yang digunakan untuk eksplorasi dan analisis data. Platform ini memungkinkan untuk menjalankan kode Python secara interaktif melalui cloud. Dalam tahap awal proyek, dimanfaatkan *Google Colab* untuk melakukan *Exploratory Data Analysis* (EDA), membersihkan data, serta mengidentifikasi pola-pola penting yang mendasari proses analisis lanjutan. Selain itu, penugasan menggunakan pustaka Python seperti Pandas, NumPy, Matplotlib, Pytorch dan Seaborn untuk mendukung analisis statistik serta membuat visualisasi yang memudahkan pemahaman data. Dalam beberapa kasus, juga membangun model *machine learning dan deep learning* dasar

menggunakan scikit-learn untuk mendukung pengambilan keputusan berbasis data.

2. Metabase

Digunakan sebagai platform untuk visualisasi data dan pembuatan dashboard interaktif. Dalam kegiatan magang, penggunaan *Metabase* untuk menghubungkan dan menampilkan data dari PostgreSQL secara visual. Antarmuka yang intuitif, memungkinkan untuk membuat laporan dan grafik tanpa harus menulis query SQL secara manual. Fitur ini sangat membantu dalam menyampaikan hasil analisis kepada pihak non-teknis. Selain itu, dibuat dashboard yang menyajikan metrik utama dan indikator performa agar tim dapat melakukan monitoring data secara berkala dan *real-time*.

3. PostGresSQL

Berperan sebagai sistem manajemen basis data relasional yang digunakan oleh peserta magang untuk mengelola dan mengambil data yang dibutuhkan dalam proses analisis. Peserta magang melakukan berbagai kueri SQL, termasuk perintah SELECT, JOIN, GROUP BY, dan pengolahan data lainnya untuk menghasilkan informasi yang relevan dari database. Dalam beberapa kesempatan, peserta magang juga melakukan optimalisasi kueri untuk mempercepat waktu pemrosesan dan meningkatkan efisiensi dalam pengambilan data berskala besar.

4. Code Editor Visual Studio Code

Digunakan sebagai editor utama dalam menulis skrip dan dokumentasi. Peserta magang menggunakan Visual Studio Code untuk menyusun kode Python, Javascript, serta mencatat proses kerja dalam bentuk dokumentasi proyek. Editor ini juga dilengkapi dengan berbagai ekstensi yang mendukung efisiensi kerja, seperti *auto-completion*, pengecekan sintaks, serta integrasi dengan sistem version control seperti Git. Dokumentasi teknis yang ditulis

menggunakan Visual Studio Code membantu dalam menjaga keteraturan alur kerja dan memudahkan kolaborasi dalam tim.

5. *Vector Storage*

Basis data vektor, atau vector database, adalah sistem penyimpanan khusus yang dirancang untuk mengelola dan mencari data dalam bentuk representasi numerik berdimensi tinggi yang dikenal sebagai vector embeddings. Berbeda dengan database tradisional yang mengindeks data berdasarkan teks atau nilai pasti, basis data ini dioptimalkan untuk melakukan pencarian berdasarkan kedekatan atau kesamaan semantik. Prosesnya dimulai dengan mengubah data tidak terstruktur seperti teks, gambar, atau audio menjadi vektor menggunakan model machine learning. Ketika sebuah kueri masuk, kueri itu juga diubah menjadi vektor, dan sistem akan dengan cepat menemukan data dengan vektor yang paling mirip dalam ruang multidimensi. Teknologi ini menjadi tulang punggung bagi berbagai aplikasi kecerdasan buatan modern, seperti sistem pencarian semantik yang memahami konteks, mesin rekomendasi yang akurat, pengenalan gambar, serta deteksi anomali. *Vector storage* menawarkan solusi ini sebagai layanan yang dikelola sepenuhnya (*fully managed*), memungkinkan pengembang untuk fokus pada inovasi aplikasi tanpa harus mengurus kompleksitas infrastruktur di baliknya.

8. Langchain

LangChain adalah framework sumber terbuka yang dirancang untuk menyederhanakan pengembangan aplikasi yang didukung oleh Large Language Models (LLM). Tujuannya adalah untuk menjembatani kesenjangan antara kemampuan LLM yang kuat dengan kebutuhan aplikasi dunia nyata, memungkinkan pengembang untuk membangun aplikasi yang lebih cerdas, kontekstual, dan interaktif. Secara umum, proses kerja Langchain dimulai ketika pengguna mengajukan pertanyaan atau permintaan. LangChain kemudian dapat mengubah pertanyaan tersebut menjadi representasi vektor

(menggunakan *embeddings*), mencari informasi yang relevan dari sumber data eksternal (menggunakan indeks), dan kemudian memberikan informasi tersebut bersama dengan pertanyaan asli ke LLM. LLM kemudian menghasilkan respons berdasarkan konteks yang diberikan.

Tabel 3.1 menunjukan uraian langkah kerja yang dijabarkan secara rinci:

Tabel 3.1 Rangkaian Langkah Kerja Kegiatan Magang

No	Aktivitas	Tanggal Mulai Aktivitas	Tanggal Selesai Aktivitas
1	Memahami regulasi perusahaan	23 Desember 2025	24 Desember 2025
2	Pembuatan dashboard analitik interaktif	24 Desember 2025	3 Januari 2025
3	Pengembangan model ringkasan dan transkrip percakapan	3 Januari 2025	21 Februari 2025
4	Pengembangan model klasifikasi topik percakapan	22 Februari 2025	8 April 2025
5	Pengembangan real-time sales promotions Retreiveal Augmented Generation (RAG) untuk virtual asisten	9 April 2025	23 Juni 2025

3.2.2 Uraian Kerja Magang

3.2.2.1 Memahami Regulasi Perusahaan

Program magang ini diawali dengan sebuah proses onboarding yang terstruktur dalam bentuk masa orientasi selama dua hari. Proses ini memiliki

tujuan strategis untuk mengintegrasikan peserta magang ke dalam lingkungan kerja serta memberikan landasan yang kuat sebelum memulai tugas. Selama periode orientasi, perusahaan memfasilitasi sesi perkenalan dengan seluruh anggota tim, yang tidak hanya bertujuan untuk mengenal individu tetapi juga untuk memahami struktur organisasi dan alur komunikasi antar departemen. Visi dan misi perusahaan juga dijelaskan secara rinci agar setiap peserta magang dapat menyelaraskan kontribusinya dengan tujuan jangka panjang perusahaan. Lebih lanjut, diberikan pula pengantar mengenai proyek yang akan menjadi tanggung jawab utama, sehingga memberikan gambaran jelas mengenai ekspektasi dan hasil yang ingin dicapai. Sebagai langkah konkret dalam menunjang produktivitas, setiap peserta magang diberikan dukungan teknis berupa akses penuh terhadap tools kerja esensial dan sebuah akun email perusahaan, yang berfungsi sebagai untuk mengakses seluruh komunikasi dan sistem internal.

3.2.2.2 Dashboard Analitik Interaktif

Tujuan utama dari pembuatan proyek *dashboard* analitik ini adalah untuk memberikan wawasan *real-time* yang dapat digunakan oleh klien dalam mengambil keputusan bisnis secara lebih cepat, akurat, dan berbasis data dari produk *software* NXTX yang berupa *chatbo*t. *Dashboard* ini dirancang agar mampu menampilkan data percakapan pelanggan secara menyeluruh, mencakup riwayat interaksi, tren komunikasi, hingga performa konversi, yang semuanya disajikan secara visual dan interaktif.

Dengan adanya *dashboard* ini, klien dapat memantau aktivitas pengguna secara langsung, menganalisis pola perilaku, serta mengidentifikasi kebutuhan pelanggan berdasarkan data historis dan tren terkini berdasarkan interaksi pelanggan dengan *chatbot* AI yang digunakan. Tujuan lainnya adalah untuk membantu klien dalam mengukur efektivitas strategi komunikasi dan pelayanan mereka, melalui metrik-metrik seperti jumlah percakapan, tingkat kepuasan pelanggan, serta konversi dari

interaksi ke tindakan bisnis. *Tools* yang digunakan berupa aplikasi metabase, sebuah aplikasi *business intelligence* (BI) yang menitikberatkan terhadap penggunaan SQL untuk pembuatan visualisasi data.

1. Memahami Kebutuhan Bisnis

Langkah pertama dalam pembuatan dashboard analitik adalah memahami kebutuhan bisnis dari klien. Proses ini dilakukan melalui diskusi dengan *stakeholder* untuk mengidentifikasi jenis data dan insight yang dibutuhkan oleh pengguna akhir. Dalam konteks proyek ini, klien menginginkan wawasan *real-time* yang dapat mendukung pengambilan keputusan cepat, akses terhadap riwayat percakapan yang lengkap, serta analisis mendalam mengenai metrik keterlibatan pengguna dan tingkat konversi. Hasil dari proses ini menjadi dasar perancangan struktur dashboard, pemilihan data yang relevan, serta jenis visualisasi yang akan digunakan.

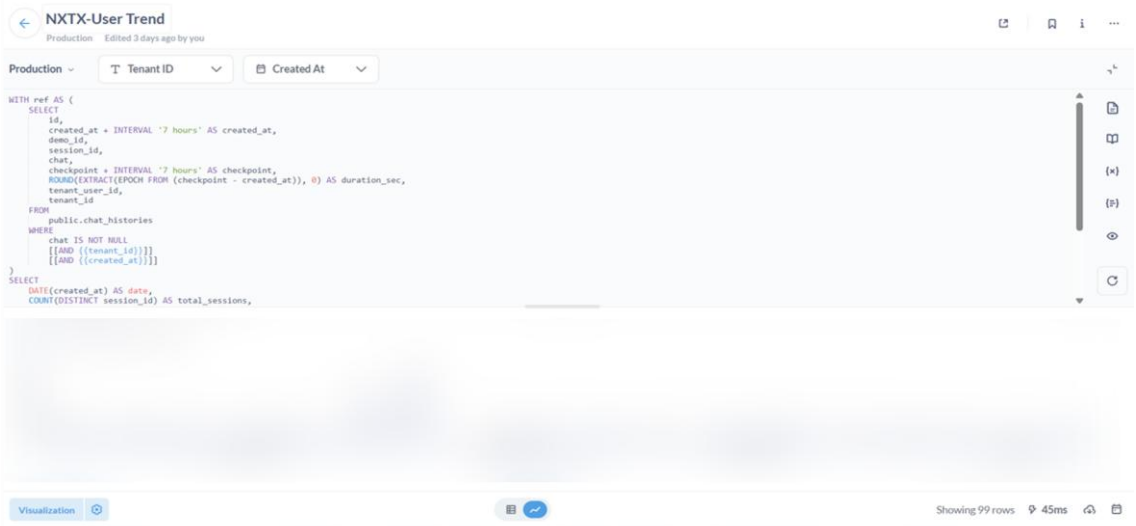
2. Penentuan Tata Letak dan Jenis Visualisasi yang Diperlukan

Setelah kebutuhan bisnis dipahami, langkah berikutnya adalah menentukan tata letak *dashboard* dan jenis visualisasi yang akan digunakan. Pada tahap ini, dibuat sketsa awal dashboard dengan membagi tampilan ke dalam beberapa bagian, seperti ringkasan metrik utama di bagian atas, grafik tren aktivitas pengguna di tengah, dan daftar riwayat percakapan di bagian bawah. Jenis visualisasi disesuaikan dengan karakteristik data, misalnya line chart untuk menunjukkan tren waktu, *pie chart* untuk distribusi *feedback*, dan tabel untuk menampilkan data percakapan secara rinci. Tujuannya adalah menciptakan tampilan *dashboard* yang informatif, mudah dipahami, dan nyaman digunakan oleh klien.

3. Pembuatan *Query* SQL untuk Setiap Visualisasi

Langkah selanjutnya adalah membuat query SQL yang digunakan untuk menampilkan data pada setiap visualisasi. *Query* dibuat menggunakan fitur

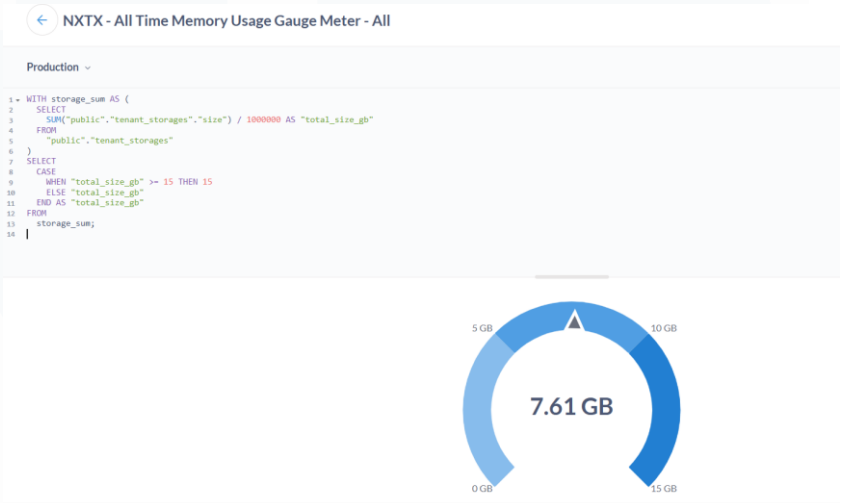
"*Native Query*" di Metabase, di mana setiap pertanyaan diformulasikan secara langsung dengan bahasa SQL untuk menghasilkan informasi yang sesuai kebutuhan. Beberapa contoh *query* yang dibuat meliputi: trend penggunaan AI *avatar chatbot* berdasarkan waktu dan jumlah session oleh user, persentase tingkat konversi pengguna, dan distribusi feedback pelanggan. Semua *query* yang telah dibuat kemudian disimpan sebagai "*Saved Questions*" agar dapat dengan mudah digunakan dan dihubungkan ke *dashboard*.



Gambar 3.2 Query SQL pembuatan visualisasi tren user

Gambar 3.2 menampilkan *query* yang bertujuan untuk menganalisis tren penggunaan pengguna (NXTX-User Trend), khususnya terkait dengan durasi sesi chat. Kode SQL ini diawali dengan *Common Table Expression* (CTE) bernama ref. Di dalam CTE ref, ia melakukan seleksi berbagai kolom penting dari tabel "public"."chat_histories". Kolom-kolom yang dipilih meliputi id, created_at (yang kemudian disesuaikan dengan menambahkan INTERVAL '7 hours' dan diberi alias created_at lagi, kemungkinan untuk penyesuaian zona waktu), demo_id, session_id, chat, checkpoint (juga disesuaikan dengan penambahan INTERVAL '7 hours'), tenant_user_id, dan tenant_id.

Bagian penting dari perhitungan adalah duration_sec, yang dihitung dengan membulatkan perbedaan waktu antara checkpoint dan created_at dalam epoch (detik). Ini mengindikasikan bahwa checkpoint mungkin menandai akhir sesi atau titik penting dalam interaksi chat, sehingga selisih waktu antara checkpoint dan created_at (awal sesi) merepresentasikan durasi chat tersebut.



Gambar 3.3 Script SQL kalkulasi ukuran knowledge base

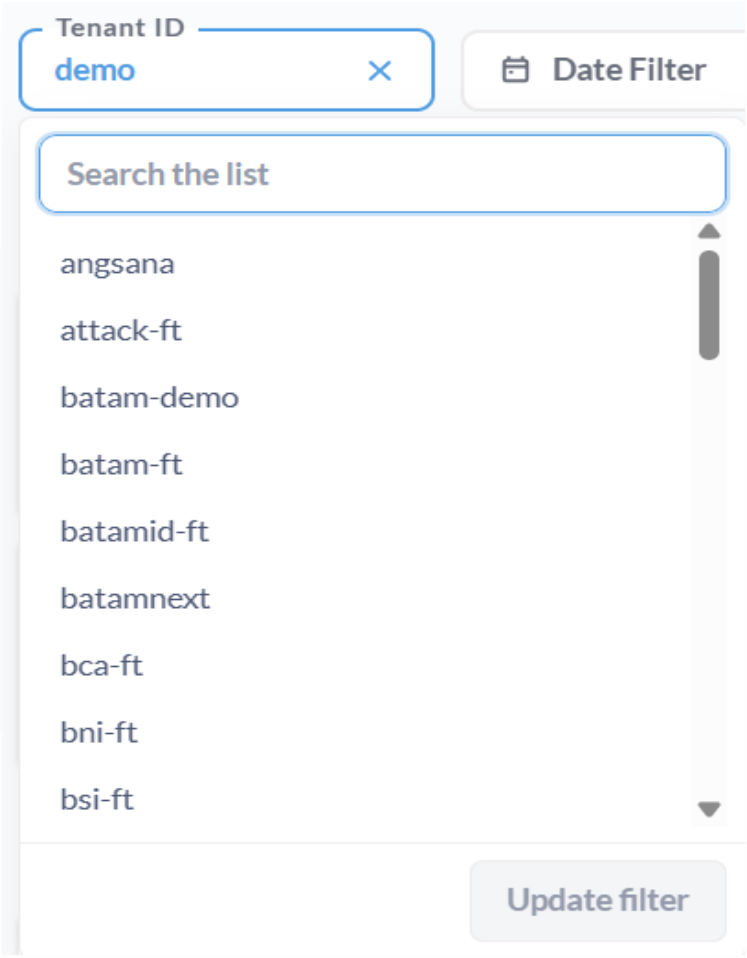
Gambar 3.3 yang ditampilkan bertujuan untuk menghitung total ukuran penyimpanan yang digunakan, yang dalam konteks Anda diasumsikan sebagai ukuran total dari *knowledge base* yang terdiri dari file PDF, Excel, dan teks website. Kode ini dimulai dengan *Common Table Expression (CTE)* bernama storage_sum. Di dalam CTE ini, ia melakukan agregasi dengan menjumlahkan kolom size dari tabel "public"."tenant_storages". Hasil penjumlahan ini kemudian dibagi dengan 1000000 (kemungkinan untuk mengonversi dari byte ke megabyte atau satuan lain yang lebih besar, dan kemudian diasumsikan menjadi gigabyte seperti yang tertera pada visualisasi) dan diberi alias total_size_gb.

Setelah menghitung total_size_gb, *query* utama memilih nilai ini, namun dengan penambahan logika CASE. Logika CASE ini berfungsi sebagai batas atas: jika total_size_gb lebih besar atau sama dengan 15, maka nilai yang ditampilkan akan dibatasi menjadi 15. Jika tidak, nilai total_size_gb yang sebenarnya akan ditampilkan. Ini menunjukkan bahwa visualisasi gauge meter yang menyertainya memiliki batas maksimum 15 GB, dan setiap nilai yang melebihi batas tersebut akan tetap ditampilkan sebagai 15 GB. Dengan demikian, kode ini secara efektif mengukur dan menampilkan penggunaan memori total *knowledge base*.

4. Merapikan Tata Letak Setiap Visualisasi Data pada Dashboard

Setelah *query* selesai dibuat, setiap hasil visualisasi dimasukkan ke dalam *dashboard*. Proses ini dilakukan dengan menambahkan masing-masing *"card"* ke dalam layout dashboard yang telah dirancang sebelumnya. Tata letak disusun agar visualisasi yang paling penting ditampilkan terlebih dahulu, seperti metrik utama di bagian atas, diikuti oleh grafik tren dan informasi tambahan lainnya. Masing-masing *car*d disesuaikan ukurannya, dan jika diperlukan ditambahkan teks deskriptif sebagai penanda atau judul bagian. Penataan ini bertujuan untuk meningkatkan pengalaman pengguna dalam membaca dan memahami data.

5. Menambahkan *Filter* Interaktif Berupa Waktu dan Tenant

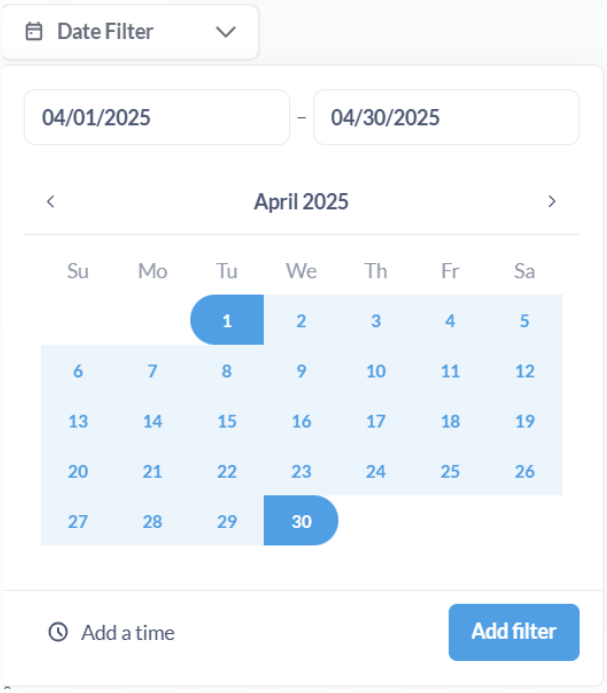


Gambar 3.4 Tampilan filter berdasarkan tenant

Gambar 3.44 merupakan tampilan antarmuka dari fitur *filter* yang digunakan pada dashboard analitik. Fitur ini memungkinkan pengguna untuk menyesuaikan dan menyaring informasi yang ditampilkan berdasarkan pilihan tenant tertentu. Dengan kata lain, setiap data dan visualisasi yang muncul di dashboard akan berubah secara dinamis mengikuti *tenant* yang dipilih melalui filter ini. Fitur ini sangat berguna dalam sistem yang memiliki banyak tenant atau pengguna, seperti pada platform yang memiliki sifat *multi-tenancy*, karena memungkinkan pengguna untuk melakukan analisis secara lebih spesifik dan terfokus.

Sebagai contoh, jika pengguna memilih *tenant* "batam-ft" dari daftar *filter*, maka seluruh informasi yang ditampilkan pada dashboard akan secara

otomatis diperbarui untuk menampilkan data yang relevan hanya dengan aktivitas, penggunaan AI, serta performa tenant batam-ft saja. Hal ini mencakup metrik seperti jumlah interaksi, volume penggunaan layanan AI, waktu aktif, jenis avatar yang digunakan, dan respons pengguna terhadap konten yang dihasilkan. Dengan pendekatan ini, pengguna atau *administrator* dapat melakukan pemantauan performa dan membuat keputusan berbasis data yang lebih tepat sasaran tanpa terganggu oleh informasi dari *tenant* lainnya.

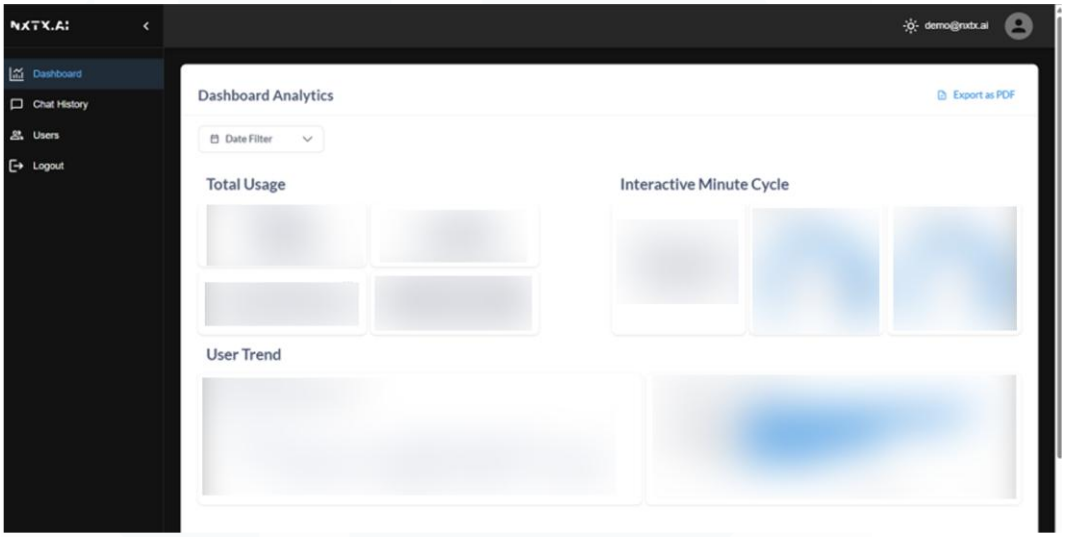


Gambar 3.5 Tampilan filter berdasarkan tanggal

Gambar 3.5 mempilan fitur untuk memperkuat fleksibilitas dalam analisis, yaitu filter interaktif untuk pemilihan tanggal ditambahkan ke dalam dashboard. *Filter* ini memungkinkan pengguna untuk memilih rentang waktu tertentu atau menyaring data berdasarkan tenant atau klien tertentu. Jenis filter yang digunakan antara lain filter tanggal dan kategori. Setiap *filter* dihubungkan ke pertanyaan yang relevan agar visualisasi dapat secara otomatis menyesuaikan dengan parameter yang dipilih pengguna.

Dengan adanya fitur ini, dashboard menjadi lebih dinamis dan adaptif terhadap kebutuhan analisis yang berbeda-beda.

6. Memasang *Embedding* Link Dashboard ke Website Khusus untuk Klien



Gambar 3.6 Tampilan akhir setelah memasang embedded link ke website

Tahap akhir dari proses pembuatan dashboard adalah melakukan embed dashboard ke dalam website khusus milik klien. Gambar 3.6 menunjukan hasil *embedding* dilakukan melalui fitur "*Embed this dashboard*" pada Metabase, yang menyediakan tautan dalam bentuk iframe atau secure link. Link tersebut kemudian disisipkan ke halaman web klien menggunakan kode HTML, sehingga dashboard dapat diakses langsung oleh pengguna akhir tanpa harus masuk ke platform Metabase. Dengan ini, klien dapat mengakses visualisasi data mereka secara langsung dan real-time melalui situs web internal yang telah mereka sediakan.

3.2.2.3 Model Pembuatan Ringkasan dari Transkrip Percakapan

1. Memahami Kebutuhan Bisnis

Pada tahap awal, dilakukan identifikasi kebutuhan bisnis terkait pentingnya merangkum percakapan antara pengguna dan asisten (*chatbot*)

secara otomatis. Ringkasan ini diharapkan dapat membantu tim ser dalam memahami inti pembicaraan tanpa harus membaca seluruh isi percakapan. Lingkup proyek difokuskan pada pengambilan data dari database, melakukan pemrosesan teks, serta menghasilkan ringkasan percakapan yang efektif dan mudah dipahami. Persyaratan teknis mencakup penggunaan model bahasa besar (large language model), evaluasi menggunakan metrik ROUGE, dan kemampuan untuk memproses data dalam jumlah besar secara efisien.

2. Pengumpulan Data Transkrip

session_id varchar	start_chat timestamptz	end_chat timestamptz	chat json
639cbf40-aa4b-11ef-b6d0-868f8a24a438	2024-11-24 10:03:47+00	NULL	[{"user":["Halo ini testinguser transkrip"],"}
dc293b31-b082-11ef-8acb-5eb5fbd8875e	2024-12-02 07:56:00+00	NULL	[{"user":["Halo"],"avatar":["Halo","","," saya
b6666d02-b083-11ef-a8ac-469419b14cc0	2024-12-02 08:02:06+00	NULL	[{"user":["Halo!"],"avatar":["Halo","","," saya
95b2c637-b088-11ef-a8ac-469419b14cc0	2024-12-02 08:37:00+00	NULL	[{"user":["Ya"],"avatar":["Halo","","," saya","
3a7de8b5-b08a-11ef-a425-02f07865ee09	2024-12-02 08:48:55+00	NULL	[{"user":["Halotest, test, halo test"],"avata
d7667632-b08b-11ef-93d4-0eb4d364c31c	2024-12-02 09:00:16+00	NULL	[{"user":["Halo halo halo halo halo halo ha
a9a64398-b08d-11ef-93d4-0eb4d364c31c	2024-12-02 09:13:22+00	NULL	[]
ec1e075b-b08d-11ef-a425-02f07865ee09	2024-12-02 09:15:11+00	NULL	[]

Gambar 3.7 Tampilan dataset yang akan diambil dari dari database

Gambar 3.7 menunjukan pengumpulan data yang dilakukan dengan mengakses database, tempat penyimpanan historis chat antara pengguna dan chatbot. Data percakapan diambil dari tabel bernama chat_histories, khususnya pada kolom chat yang berisi format JSON. Transkrip percakapan yang ada berada dalam bahasa indonesia dan inggris. Data tersebut kemudian dikonversi ke dalam bentuk DataFrame menggunakan library pandas untuk mempermudah manipulasi dan pemrosesan selanjutnya. Pada gambar diatas terlihat fitur-fitur yang terdapat di tabel chat_histories. akan

3. Pra-Pemrosesan Data Transkrip Menggunakan Teknik NLP

```python
# Optimized data extraction
def extract_conversation(chat_json):
    if isinstance(chat_json, str):
        try:
            chat_data = json.loads(chat_json)
        except:
            return ""
    elif isinstance(chat_json, (list, dict)):
        chat_data = chat_json
    else:
        return ""

    messages = []
    for turn in chat_data if isinstance(chat_data, list) else [chat_data]:
        if isinstance(turn, dict):
            user_msg = ' '.join(turn['user']) if 'user' in turn and isinstance(turn['user'], list) else ''
            avatar_msg = ' '.join(turn['avatar']) if 'avatar' in turn and isinstance(turn['avatar'], list) else ''
            if user_msg:
                messages.append(f"User: {user_msg}")
            if avatar_msg:
                messages.append(f"Assistant: {avatar_msg}")
    return ' '.join(messages)

# Preprocessing
df['cleaned_chat'] = df['chat'].apply(extract_conversation)
df = df[df['cleaned_chat'].str.len() > 0]

# Create reference summaries
def create_reference(text):
    try:
        sentences = sent_tokenize(text)
        return ' '.join(sentences[:3]) if len(sentences) > 3 else text
    except:
        return text

df['reference'] = df['cleaned_chat'].apply(create_reference)
```

Gambar 3.8 Script Python untuk pemrosesan data

Setelah data terkumpul, dilakukan proses pra-pemrosesan teks. Gambar 3.8 melibatkan tahapan ekstraksi struktur percakapan menggunakan fungsi *extract_conversation*, yang memisahkan dialog antara pengguna dan asisten menjadi format teks terstruktur. Selanjutnya, dibuat ringkasan referensi (reference summary) menggunakan fungsi *create_reference*, dengan mengambil hingga tiga kalimat pertama dari percakapan sebagai acuan penilaian hasil ringkasan. Pra-pemrosesan ini penting untuk memastikan bahwa data bersih, konsisten, dan siap digunakan dalam proses pelatihan atau inferensi model.

```python
def normalize_word(word, target_word="ukrida", threshold=4):
    cleaned_word = word.replace(" ", "").lower()
    distance = Levenshtein.distance(cleaned_word, target_word.lower())
    return target_word if distance <= threshold else word
```

Gambar 3.9 Script python untuk normalisasi data

Gambar 3.9 menunjukan tahapan normalisasi data juga dilakukan dengan menggunakan teknik *levenshtein distance*. Teknik ini digunakan untuk menormalkan kata-kata yang mirip tetapi tidak identik (misalnya karena kesalahan ketik, variasi penulisan, dll.)

4. Data Modelling

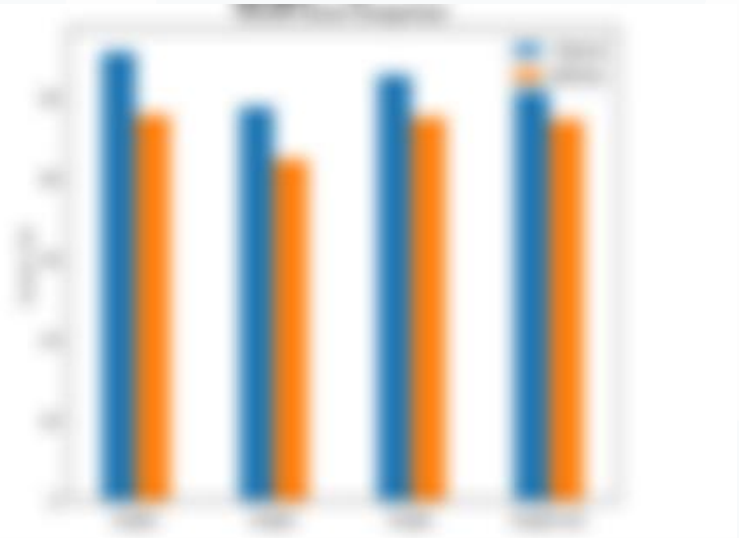
```python
def generate_summaries(texts, batch_size=4):
    summaries = []
    for i in tqdm(range(0, len(texts), batch_size), desc="Generating summaries"):
        batch = texts[i:i+batch_size]
        inputs = tokenizer(batch, return_tensors="pt", padding=True, truncation=True,
        summary_ids = model.generate(
            inputs.input_ids,
            max_length=150,
            min_length=30,
            num_beams=4,
            early_stopping=True
        )
        summaries.extend(tokenizer.batch_decode(summary_ids, skip_special_tokens=True)
    return summaries
```

Gambar 3.10 Script python untuk fine-tuning model

Gambar 3.10 menunjukan tahapan proses pemodelan dilakukan dengan melakukan fine tuning model pre-trained dari *Hugging Face*, yaitu

facebook/bart-large-cnn, yang dikenal handal dalam tugas ringkasan teks (text summarization). Model ini diload menggunakan PyTorch dan dioptimalkan untuk berjalan di GPU jika tersedia. Fungsi *generate_summaries* dibuat untuk menghasilkan ringkasan secara batch, sehingga proses dapat berjalan efisien untuk banyak teks sekaligus. Setelah itu, hasil ringkasan awal diperhalus menggunakan sebuah model tambahan yang lebih ringan. Model ini digunakan untuk meningkatkan keluwesan dan keterbacaan ringkasan dengan pendekatan prompt yang meminta perbaikan dalam bahasa Indonesia agar terdengar lebih alami.

5. Evaluasi Model



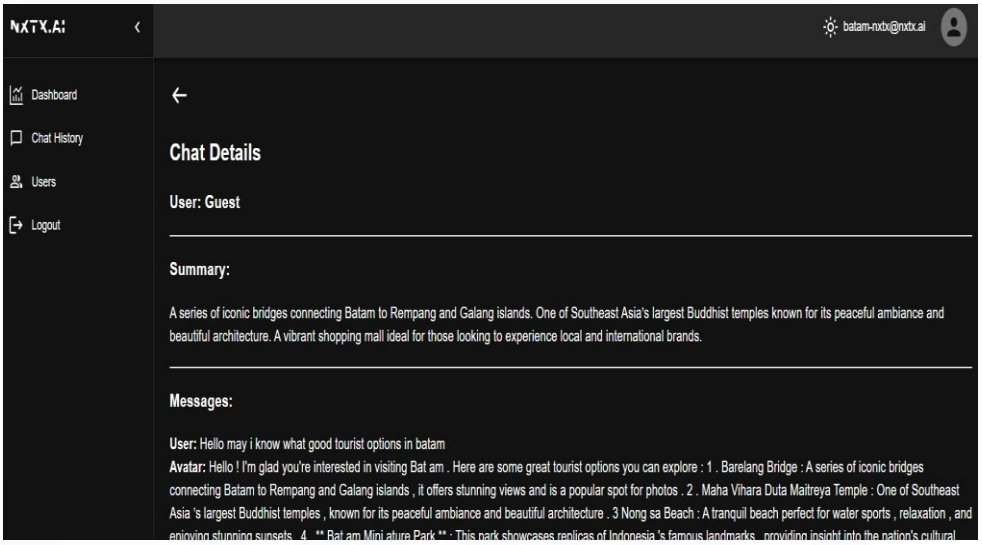
Gambar 3.11 Tampilan hasil evaluasi model

Gambar 3.11 menunjukan evaluasi hasil ringkasan dilakukan dengan menggunakan metrik ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*), yang terdiri dari ROUGE-1, ROUGE-2, dan ROUGE-L. Metrik ini membandingkan kemiripan antara ringkasan yang dihasilkan model dengan ringkasan referensi yang dibuat sebelumnya. Evaluasi dilakukan dua kali: sebelum dan sesudah proses perbaikan (*refinemen*t). Hasil evaluasi kemudian divisualisasikan dalam bentuk grafik batang

menggunakan matplotlib dan seaborn untuk mempermudah pemahaman terhadap peningkatan performa model setelah *refinement* dilakukan.

6. Integrasi API menggunakan FastAPI

Tahap selanjutnya yang adalah membungkus pipeline model ke dalam sebuah layanan API menggunakan FastAPI untuk bisa selalu terhubung ke database serta *website* analisis data perusahaan secara real-time. Hal ini bertujuan agar sistem atau aplikasi lain dapat mengakses fungsi ringkasan ini secara *real-time*, sehingga mempermudah integrasi dengan platform lain yang menggunakan layanan chatbot atau dokumentasi percakapan.



Gambar 3.12 Tampilan akhir output model pada website khusus klien

Gambar 3.12 merupakan hasil dari integrasi model yang dilakukan dengan baik. hasil output dari model ringkasan transkripsi muncul pada bagian "*Summary*".

7. Monitoring

Untuk menjamin kualitas model yang sudah diimplementasikan di lingkungan produksi, tahap monitoring menjadi sangat penting. Proses monitoring mencakup pelacakan performa model, waktu respon, serta deteksi kesalahan melalui sistem logging.

3.2.2.4 Model Klasifikasi Topik Percakapan Dinamis

Tujuan utama skrip ini adalah untuk menganalisis secara otomatis percakapan antara pengguna dan avatar guna mengidentifikasi topik utama atau niat pengguna. Informasi ini dapat digunakan untuk berbagai tujuan, seperti memahami pertanyaan umum, meningkatkan respons *chatbot*, mengarahkan percakapan ke departemen yang sesuai, dan mengumpulkan analisis tentang minat pengguna. Sistem ini dirancang untuk multi-tenant, artinya dapat melayani universitas atau departemen yang berbeda, masing-masing dengan kumpulan kategori yang unik.

1. Inisialisasi

Fase awal proyek melibatkan proses pengaturan dan inisialisasi yang komprehensif untuk memastikan aplikasi berjalan dengan lancar dan aman. Pertama, semua perpustakaan Python yang diperlukan diimpor untuk menyediakan fungsionalitas yang diperlukan bagi skrip. Informasi sensitif seperti kunci API dan URL database dimuat dari berkas .env menggunakan fungsi load_dotenv(), yang sangat penting untuk menjaga keamanan dan menyederhanakan manajemen konfigurasi. LLM kemudian di inisialisasi dengan konfigurasi tersebut. Setalah itu, aplikasi Celery bernama ‘tasks’ diinisialisasi. Celery dikonfigurasi untuk menggunakan Redis sebagai broker pesan, dengan logika *retry* bawaan untuk menangani masalah koneksi broker dengan lancar. Selain itu, kamus global bernama GLOBAL_CATEGORY_CACHE dibuat untuk berfungsi sebagai cache dalam memori untuk data kategori, diindeks berdasarkan tenant_id. Mekanisme caching ini diimplementasikan untuk meminimalkan kueri database yang berulang dan meningkatkan efisiensi pengambilan data kategori di seluruh aplikasi.

2. Fungsi Bantu untuk Normalisasi Teks dan Pembuatan Slug

Langkah kedua dalam proyek ini berfokus pada implementasi fungsi bantu yang mempermudah normalisasi teks dan pembangkitan *slug*, yang sangat penting untuk menjaga konsistensi dan kegunaan data.

```python
# Global cache for categories to reduce API calls
GLOBAL_CATEGORY_CACHE = {}

def normalize_word(word, target_word="ukrida", threshold=4):
    """Normalize a word to a target word using Levenshtein edit distance."""
    cleaned_word = word.replace(" ", "").lower()
    distance = Levenshtein.distance(cleaned_word, target_word.lower())
    return target_word if distance <= threshold else word

def clean_label(label):
    """Normalize category labels to avoid inconsistencies."""
    label = " ".join(label.split()).strip()
    label = re.sub(r'\*\*', '', label)
    label = re.sub(r'[^\w\s-]', '', label)
    label = label.lower()
    label = re.sub(r'^\d+', '', label)
    label = re.sub(r'-', '', label)
    label = re.sub(r'\b(di|dari|tegalwar)\b', '', label)
    label = " ".join(label.split())

    words = label.split()
    normalized_words = [normalize_word(word) if word != "ukrida" else word for word in words]
```

Gambar 3.13 Script Normalisasi Data Teks Percakapan

Gambar 3.13 menunjukan fungsi normalize_word dirancang untuk menstandarkan variasi kata kunci tertentu—seperti “ukrida”—dengan membersihkan input (menghapus spasi dan mengubah ke huruf kecil) serta menghitung jarak *Levenshtein* ke kata target. Jika perbedaan berada dalam batas ambang yang ditentukan, fungsi ini mengembalikan bentuk kanonik, sehingga dapat menangani kesalahan ketik atau variasi ejaan minor. Fungsi clean_label lebih lanjut meningkatkan keseragaman data dengan memproses label kategori: menghapus spasi ekstra, simbol *markdown*, karakter non-alfanumerik, digit awal, dan kata-kata stop umum dalam bahasa Indonesia. Fungsi ini juga memisahkan label menjadi kata-kata individu, menormalisasi istilah tertentu menggunakan normalize_word, dan memastikan keunikan sambil mempertahankan urutan kata. Proses ini

memastikan bahwa label serupa seperti “Biaya Kuliah” dan “biaya kuliah!!” diperlakukan secara identik. Terakhir, fungsi generate_slug membuat slug yang ramah URL dengan mengubah label menjadi huruf kecil, mengganti spasi dengan tanda hubung, dan menghapus karakter yang tidak diinginkan yang tersisa. Fungsi-fungsi bantu ini secara kolektif memastikan bahwa data kategori bersih, konsisten, dan mudah diakses untuk pemrosesan lebih lanjut atau penggunaan dalam URL.

3. Pengembangan Kelas Agen (Logika Inti)

Logika inti aplikasi dikapsulkan dalam kelas Agent, yang bertanggung jawab untuk memproses riwayat obrolan dan mengkategorikan percakapan untuk setiap tenant spesifik. Saat diinisialisasi, kelas Agent memuat dan menyimpan semua kategori yang ada untuk tenant yang diberikan, baik dari cache memori global maupun dengan mengakses database, memastikan akses efisien ke informasi kategori.

```
class Agent:
        return closest_label if highest_score >= threshold else None

    def get_embedding(self, text, max_tokens=[redacted], max_retries=3):
        [redacted]
        tokens = encoding.encode(text)
        if len(tokens) > max_tokens:
            tokens = tokens[:max_tokens]
        truncated_text = encoding.decode(tokens)
```

Gambar 3.14 Fungsi Pengambilan Embedding

```
class Agent:

    def cosine_similarity(self, vec1, vec2):
        dot_product = np.dot(vec1, vec2)
        norm_vec1 = np.linalg.norm(vec1)
        norm_vec2 = np.linalg.norm(vec2)
        return dot_product / (norm_vec1 * norm_vec2) if norm_vec1 and norm_vec2 else 0
```

Gambar 3.15 Fungsi Perbandingan Cosinus

Kelas ini mencakup metode untuk mengorganisir data obrolan mentah menjadi blok percakapan yang dapat dibaca, yang kemudian disiapkan sebagai input untuk LLM. Untuk menjaga konsistensi dalam pengelompokan, Agent menggunakan pencocokan string *fuzzy* untuk memetakan label yang dihasilkan LLM ke kategori yang sudah ada, memungkinkan variasi kecil dalam penulisan label. Gambar 3.14 dan Gambar 3.15 menampilan bahwa *class* ini dapat menghasilkan vektor embedding untuk teks dan menghitung kesamaan kosinusnya, mendukung perbandingan semantik lanjutan jika diperlukan.

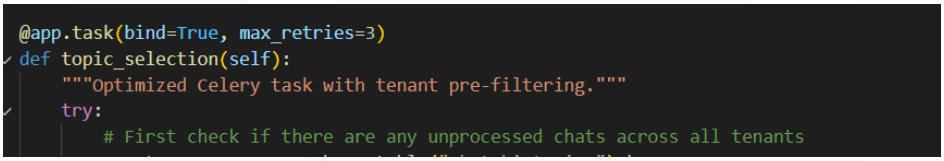
Fungsi kunci memanfaatkan LLM untuk menghasilkan label kategori yang ringkas dan sadar konteks dari percakapan obrolan, dipandu oleh prompt yang dirancang dengan cermat yang mencerminkan persyaratan spesifik domain dan memastikan konsistensi output. Berdasarkan Kelas Agen juga mengandung mekanisme untuk menyaring label yang tidak relevan atau generik, serta menentukan kategori paling sesuai untuk setiap percakapan dengan mencocokkan label yang dihasilkan dengan kategori yang disimpan menggunakan pencocokan tepat dan *fuzzy*.

```
class Agent:

    def process_chat_history(self):
        """Optimized version of process_chat_history with batch processing."""
        log.info(f"🚀 Agent: Categorizing chats for Tenant ID: {self.tenant_id}...")

        # Check if there are any unprocessed chats first
        count_response = [redacted] \
            .select("count", count="exact") \
            .filter("tenant_id", "eq", self.tenant_id) \
            .execute()

        if count_response.data[0]["count"] == 0:
            log.info(f"ℹ️ No unprocessed chats for Tenant ID: {self.tenant_id}")
            return

        # Process in batches of 100
        batch_size = 100
        start = 0
        total_processed = 0
```

Gambar 3.16 Fungsi pemrosesan riwayat percakapan

Gambar 3.16 menampilkan bahwa untuk pemrosesan batch, Agen mengiterasi melalui catatan percakapan yang belum diproses, menetapkan kategori, dan memperbarui basis data secara efisien seperti pada. Jika percakapan terlalu singkat atau tidak memiliki konten yang berarti, percakapan tersebut akan diberi kategori dan ringkasan default. Selama proses ini, Agent memastikan bahwa hanya penugasan kategori baru yang dimasukkan ke dalam database, menghindari duplikasi dan menjaga integritas data. Pendekatan terstruktur dan modular ini memungkinkan kategorisasi yang skalabel dan akurat untuk volume besar data obrolan di seluruh tenant, sambil memanfaatkan kemampuan LLM secara aman untuk pemahaman bahasa alami dan pembangkitan label.

4. Pemrosesan Tugas Asinkron dengan Celery



Gambar 3.17 fungsi pemilihan chat yang akan diproses

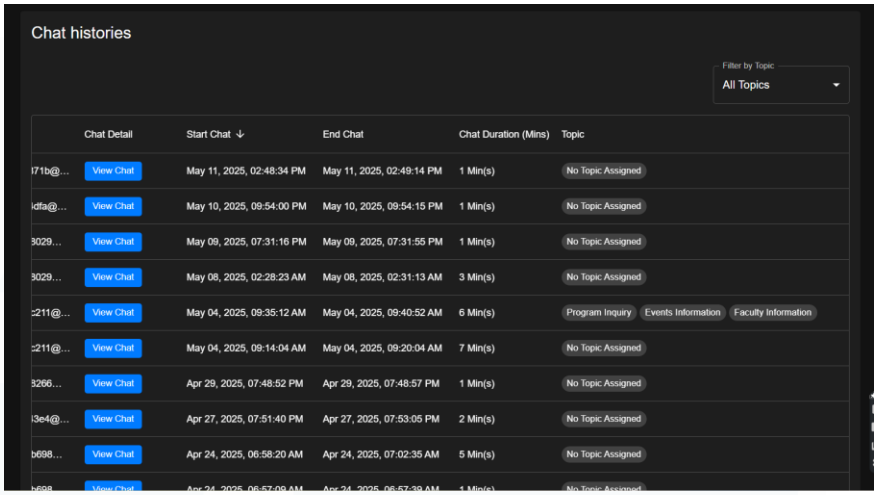
Pada langkah keempat, pemrosesan tugas asinkron diimplementasikan menggunakan Celery untuk mengelola pemrosesan obrolan secara efisien di seluruh tenant. Gambar 3.17 menunjukan sebuah tugas Celery bernama topic_selection didefinisikan dengan kemampuan retry, memungkinkan tugas tersebut secara otomatis mencoba ulang hingga tiga kali jika terjadi kegagalan. Tugas ini dimulai dengan memeriksa secara global apakah ada obrolan yang belum diproses (di mana ringkasan bernilai NULL) di semua *tenant*; jika tidak ada yang ditemukan, tugas ini mencatat status ini dan keluar lebih awal untuk menghemat sumber daya.

Selanjutnya, tugas ini memanggil panggilan prosedur jarak jauh (RPC) Postgresql bernama get_tenants_with_unprocessed_chats untuk mengambil daftar tenant yang memiliki obrolan yang belum diproses dan memerlukan

pemrosesan. Jika tidak ada tenant yang dikembalikan, tugas ini mencatat hal ini dan berakhir. Untuk setiap tenant yang teridentifikasi, tugas ini mengambil ID tenant. Untuk setiap tenant yang memenuhi syarat, *instance* kelas Agent dibuat untuk menangani riwayat obrolan tenant, memanggil metode pemrosesannya untuk mengkategorikan dan merangkum obrolan yang belum diproses. Penanganan kesalahan yang andal diimplementasikan menggunakan blok try-except; jika terjadi pengecualian selama pemrosesan, kesalahan dicatat, dan tugas dijadwalkan untuk dicoba ulang setelah penundaan 60 detik. Desain asinkron ini memastikan pemrosesan data obrolan yang skalabel dan tahan kesalahan, memungkinkan pembaruan tepat waktu sambil meminimalkan *downtime* atau intervensi manual.

5. Eksekusi Script

Pada langkah terakhir, skrip mencakup blok eksekusi bersyarat yang hanya dijalankan ketika skrip dieksekusi secara langsung, bukan sebagai modul yang diimpor. Di dalam blok ini, tugas topic_selection dikirimkan secara asinkron ke antrian tugas Celery menggunakan metode .delay(). Hal ini mengantrekan tugas untuk diproses oleh pekerja Celery terpisah, yang harus berjalan secara independen untuk mengambil dan menjalankan tugas tersebut. Desain ini memungkinkan skrip utama memicu pemrosesan latar belakang tanpa memblokir, sehingga memungkinkan penanganan tugas kategorisasi obrolan yang efisien dan skalabel.



Gambar 3.18 Hasil Akhir Klasifikasi Percakapan

Gambar 3.18 merupakan tampilan akhir dari tugas ini. Teks percakapan yang sudah diproses akan mendapatkan label intent percakapan berdasarkan topik apa saja yang dilakukan user saat berinteraksi dengan AI. Label-label ini ditampilkan pada dashboard khusus klien pada bagian kolum “Topic”. Hal ini memungkinkan klien untuk menganalisa setiap percakapan yang ada dengan mudah dan cepat untuk kepentingan bisnis mereka.

3.2.2.5 Pengembangan *Sales Promotion* RAG untuk Asisten Virtual

Proyek ini bertujuan untuk mengembangkan sistem promosi penjualan real-time menggunakan *Retrieval-Augmented Generation* (RAG) yang terintegrasi dengan database produk dan voucher. Sistem ini memungkinkan pelanggan untuk mendapatkan informasi produk dan promosi yang relevan secara real-time melalui interaksi dengan asisten virtual.

Retrieval-Augmented Generation (RAG) merupakan pendekatan yang menggabungkan dua kekuatan utama dalam AI modern:

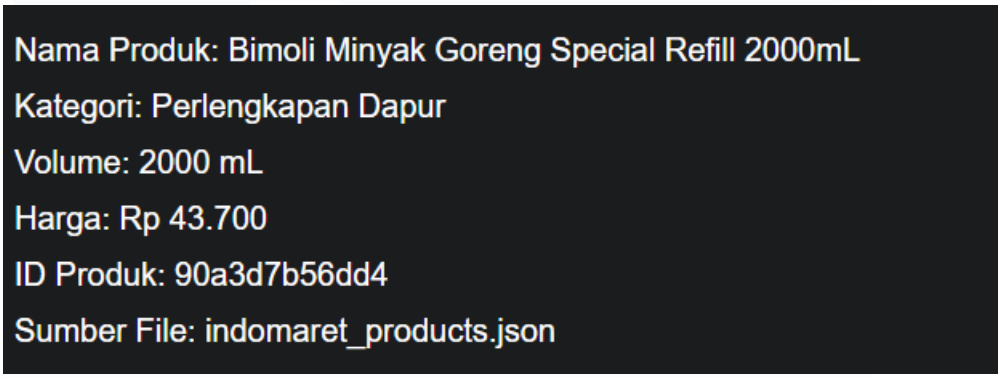
1. *Information Retrieval (IR)* : mengambil informasi yang relevan dari basis data (structured/unstructured)

2. *Natural Language Generation* (NLG) : membentuk respons bahasa alami dari informasi tersebut menggunakan model bahasa besar (LLM).

Dengan pendekatan ini, sistem tidak hanya mengandalkan model bahasa untuk menjawab pertanyaan, tetapi juga menarik informasi yang akurat dan terkini dari basis data internal, menjadikannya ideal untuk kebutuhan sales promotion yang cepat berubah dan kontekstual.

Terdapat dua metadata utama yang diperlukan untuk pembuatan RAG ini:

1. Metadata Produk

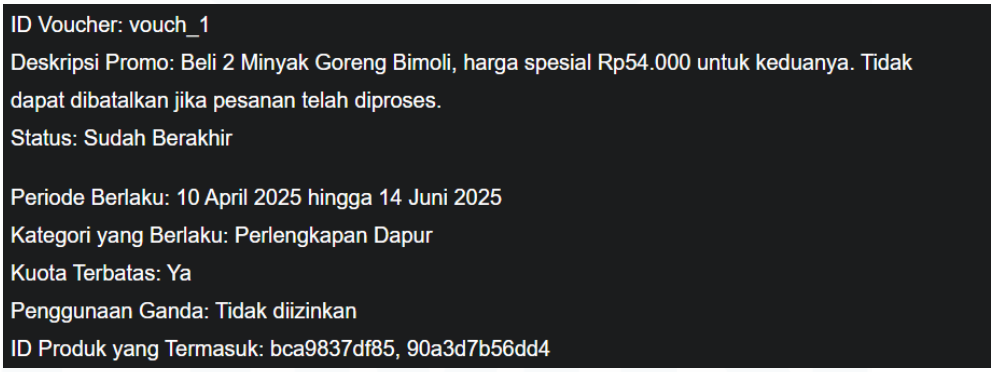


Gambar 3.19 Rencana Metadata Produk

Gambar 3.19 menampilkan rencana dimana *field* ID pada metadata produk, seperti "ibca9837df85", berfungsi sebagai pengenal unik universal untuk setiap item produk dalam sistem. Peranannya dalam sistem RAG sangat penting: setelah proses pencarian semantik di *vector storage* mengidentifikasi vektor produk yang paling relevan dengan kueri pengguna, ID inilah yang digunakan sebagai kunci untuk mengambil keseluruhan informasi detail produk dari database relasional utama. Selain itu, ID produk ini menjadi jembatan vital untuk menghubungkan produk secara presisi dengan promosi tertentu, yaitu melalui referensi dalam field products_included pada metadata voucher, memastikan bahwa diskon hanya diterapkan pada item yang benar-benar dituju.

Field content, yang dalam contoh berisi "Bimoli Minyak Goreng 2000mL, volume: 2000mL, price: Rp 41,400.", merupakan representasi tekstual yang paling kaya dan deskriptif mengenai suatu produk, seringkali merupakan gabungan dari nama, atribut kunci, dan deskripsi singkat. Dalam arsitektur RAG, *field content* ini memegang peranan sentral karena teks inilah yang umumnya diubah menjadi embedding vektor untuk disimpan dan diindeks dalam database vektor guna mendukung pencarian semantik. Kualitas dan kelengkapan informasi dalam content secara langsung mempengaruhi akurasi pencarian produk yang relevan dengan kueri pengguna. Selanjutnya, ketika produk berhasil diambil, sebagian atau seluruh isi content ini akan dimasukkan ke dalam prompt yang diberikan kepada LLM, menyediakan dasar faktual yang kaya bagi LLM untuk merumuskan jawaban atau deskripsi produk.

2. Metadata *Voucher*



Gambar 3.20 Rencana Metadata Voucher

Gambar 3.20 menampilkan rencana dimana *field* ID pada metadata voucher, seperti "vouch_1", berfungsi sebagai pengenal unik untuk setiap promosi atau voucher yang ada dalam sistem. Keunikan ID ini memungkinkan sistem untuk melacak, mengelola, dan merujuk pada setiap *voucher* secara individual tanpa ambiguitas. Dalam konteks RAG, meskipun mungkin tidak selalu ditampilkan langsung kepada pengguna akhir, ID voucher ini penting untuk proses internal, seperti saat LLM perlu

merujuk pada promosi spesifik jika terdapat beberapa opsi, atau untuk keperluan logging dan analisis efektivitas promosi.

Field end_date, seperti "2025-06-14T23:59:59+07:00", secara presisi menetapkan tanggal dan waktu berakhirnya masa berlaku sebuah voucher. Penggunaan format standar ISO 8601, lengkap dengan informasi zona waktu (dalam contoh +07:00 menandakan Waktu Indonesia Barat), sangat krusial untuk akurasi. Dalam sistem RAG, end_date memainkan peran vital dalam proses validasi: sistem harus secara otomatis memeriksa apakah tanggal dan waktu saat ini masih berada sebelum end_date yang tertera. Jika sudah melewati batas waktu ini, voucher dianggap kadaluwarsa dan tidak akan diambil atau ditawarkan kepada pengguna, memastikan hanya promosi yang valid yang disampaikan.

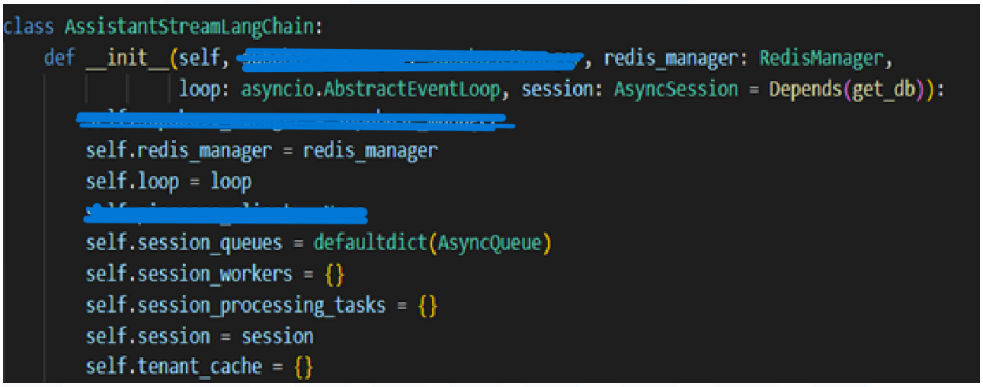
Field products_included, yang dalam contoh berisi daftar ID produk seperti ["ibca9837df85", "90a3d7b56dd4"], memungkinkan penargetan promosi voucher ke produk-produk yang sangat spesifik. Ini adalah mekanisme pencocokan voucher yang paling presisi dalam sistem RAG. Ketika pengguna menunjukkan minat pada suatu produk, sistem akan memeriksa apakah ID produk tersebut (dari metadata produk) terdapat dalam daftar products_included pada metadata voucher. Jika ada kecocokan, ini menandakan bahwa voucher tersebut secara eksplisit dirancang untuk produk tersebut, sehingga sangat relevan untuk ditawarkan.

Field start_date, misalnya "2025-04-10T00:00:00+07:00", menentukan tanggal dan waktu kapan sebuah voucher mulai berlaku. Sama seperti end_date, akurasi tanggal, waktu, dan zona waktu sangat penting. Dalam sistem RAG, start_date digunakan bersama end_date untuk melakukan validasi keaktifan voucher. Sistem akan memastikan bahwa tanggal dan waktu saat ini sudah melewati atau sama dengan start_date sebelum sebuah voucher dianggap aktif dan dapat ditawarkan kepada pengguna. Ini mencegah promosi ditawarkan sebelum periode yang telah ditetapkan.

Field text, yang dalam contoh berisi "Bell 2 Minyak Goreng Bimoli, harga spesial Rp54.000 untuk keduanya...", adalah deskripsi atau narasi promosi aktual yang dirancang untuk dilihat atau didengar oleh pengguna. Ini adalah aspek marketing dari penawaran voucher tersebut. Dalam sistem RAG, setelah sistem internal menentukan bahwa sebuah voucher valid dan relevan untuk produk yang diminati pengguna, field text inilah yang menjadi salah satu input utama bagi *Large Language Model* (LLM). LLM akan menggunakan text ini, bersama dengan informasi produk dan konteks percakapan lainnya, untuk merangkai penjelasan mengenai promosi tersebut dalam bahasa yang alami, menarik, dan mudah dipahami oleh pengguna.

Berikut merupakan langkah-langkah proses pengembangan RAG tersebut :

1. Inisialisasi Sistem



Gambar 3.21 Class AssistantStreamLangchain

Gambar 3.21 merupakan langkah fundamental dalam pengembangan sistem interaksi pelanggan cerdas ini adalah pembangunan fondasi melalui kelas AssistantStreamLangChain, yang berfungsi sebagai pengelola utama seluruh alur kerja (*pipeline*) sistem. Kelas ini bertanggung jawab menginisialisasi serangkaian koneksi krusial yang menjadi penopang operasional sistem. Koneksi pertama adalah ke Postgresql, sebuah basis data relasional yang digunakan untuk menyimpan data inti terstruktur seperti

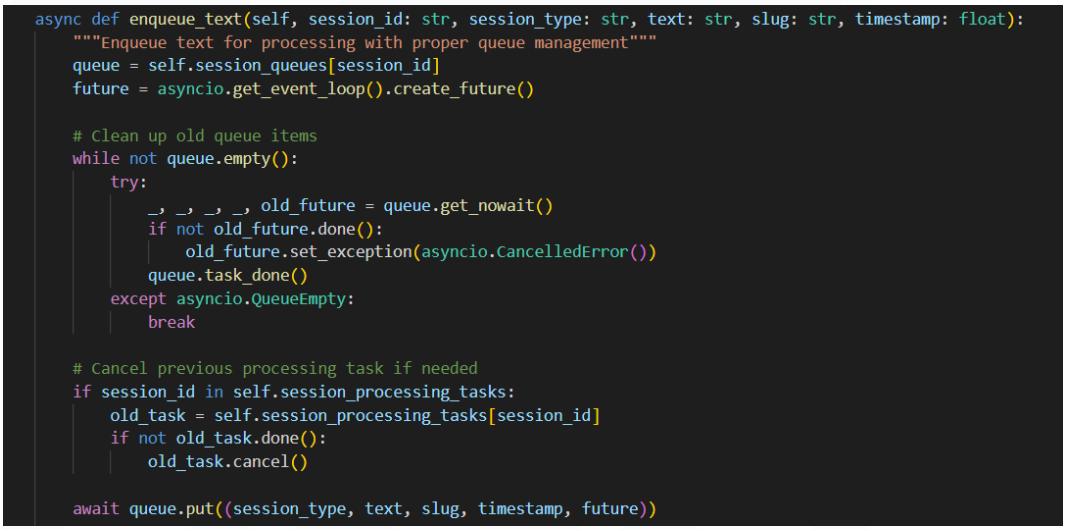
informasi detail produk, berbagai skema voucher promosi, serta konfigurasi spesifik untuk masing-masing tenant atau klien, yang esensial untuk personalisasi layanan.

Selanjutnya, sistem terhubung ke Redis, sebuah in-memory data store yang berperan vital dalam *caching* data yang sering diakses dan pengelolaan state sementara seperti sesi pengguna, yang secara signifikan mempercepat waktu respons sistem dan menjaga konteks percakapan. Koneksi penting lainnya adalah ke sebuah database vektor yang menjadi kunci untuk fitur pencarian semantik. Database vektor mendapatkan *embedding* teks dari deskripsi produk, memungkinkan sistem melakukan pencarian berbasis makna kontekstual terhadap pertanyaan pengguna, bukan sekadar pencocokan kata kunci. Untuk menangani interaksi pengguna secara efisien dan responsif, sistem mengimplementasikan Async Queue dan Session Worker, di mana setiap interaksi pengguna dikelola secara asinkron melalui antrian pesan dan diproses oleh *worker* khusus per sesi, memungkinkan pemrosesan yang terpisah dan paralel sehingga sistem dapat menangani banyak pengguna secara bersamaan. Seluruh arsitektur ini juga dirancang untuk mendukung *multi-tenancy*, memungkinkan sistem melayani lebih dari satu brand atau toko secara simultan dari satu instans aplikasi, dengan konfigurasi dan data yang disesuaikan untuk setiap *tenant*, yang pengelolaannya juga didukung oleh data konfigurasi tenant di Postgresql.

2. Proses *Enqueue* Pesan

Ketika seorang pengguna berinteraksi dengan sistem dengan mengirimkan sebuah pertanyaan, misalnya, “Ada diskon apa untuk minyak goreng?”, sistem segera menginisiasi sebuah alur kerja yang terstruktur untuk menangani pesan tersebut, yang dikenal sebagai proses enqueue pesan. Tahap pertama dalam proses ini adalah tindakan preventif yang krusial: sistem akan membersihkan pesan-pesan sebelumnya yang mungkin masih ada dalam antrian khusus untuk sesi pengguna tersebut, terutama jika

pesan-pesan lama itu belum memasuki tahap pemrosesan aktif. Pembersihan ini dilakukan untuk memitigasi risiko terjadinya konflik pemrosesan atau duplikasi respons. Kondisi ini bisa terjadi jika pengguna mengirimkan beberapa pertanyaan secara berurutan dalam waktu singkat, atau jika pengguna mengoreksi pertanyaan sebelumnya. Dengan memprioritaskan dan hanya mempertahankan pesan terbaru yang belum diproses, sistem memastikan bahwa sumber daya komputasi akan difokuskan untuk menjawab intensi terkini dari pengguna, sehingga meningkatkan relevansi dan efisiensi respons.



```python
async def enqueue_text(self, session_id: str, session_type: str, text: str, slug: str, timestamp: float):
    """Enqueue text for processing with proper queue management"""
    queue = self.session_queues[session_id]
    future = asyncio.get_event_loop().create_future()

    # Clean up old queue items
    while not queue.empty():
        try:
            _, _, _, _, old_future = queue.get_nowait()
            if not old_future.done():
                old_future.set_exception(asyncio.CancelledError())
            queue.task_done()
        except asyncio.QueueEmpty:
            break

    # Cancel previous processing task if needed
    if session_id in self.session_processing_tasks:
        old_task = self.session_processing_tasks[session_id]
        if not old_task.done():
            old_task.cancel()

    await queue.put((session_type, text, slug, timestamp, future))
```

Gambar 3.22 Script *Enqueue* Text

Gambar 3.22 memastikan bahwa antrian sesi pengguna bersih dari pesan-pesan yang usang atau sudah tidak relevan, pesan baru yang berisi pertanyaan aktual pengguna tersebut kemudian ditambahkan (di-*enqueue*) ke dalam antrian yang didedikasikan spesifik untuk sesi pengguna yang bersangkutan. Antrian ini berfungsi sebagai *buffer* yang terorganisir, memastikan bahwa pesan diproses sesuai urutan kedatangan yang logis dalam konteks sesi tersebut. Setiap item yang dimasukkan ke dalam antrian ini tidak hanya berisi teks pertanyaan mentah, tetapi juga dapat disertai

metadata penting seperti timestamp pengiriman, ID sesi unik, dan informasi kontekstual lainnya yang mungkin diperlukan oleh worker pemroses.

Langkah penting berikutnya adalah menghubungkan pesan yang baru saja di-enqueue tersebut dengan sebuah objek Future. Dalam konteks pemrograman asinkron, seperti yang digunakan dalam sistem ini, objek Future bertindak sebagai placeholder atau penampung untuk hasil komputasi yang belum tersedia pada saat itu tetapi diharapkan akan ada di masa mendatang. Bagian sistem yang bertanggung jawab atas komunikasi dengan pengguna (seperti antarmuka *chatbot*) akan 'menunggu' (await) pada objek Future ini. Ketika *worker* sesi telah selesai memproses pesan dan menghasilkan sebuah jawaban atau serangkaian data, hasil tersebut akan "diselesaikan" atau "disematkan" ke dalam objek Future yang terkait. Mekanisme ini memungkinkan antarmuka pengguna tetap responsif dan tidak terblokir, serta memungkinkan sistem untuk mulai mengirimkan potongan-potongan respons kepada pengguna secara *real-time* atau bertahap segera setelah tersedia, tanpa harus menunggu keseluruhan proses komputasi internal selesai.

Keseluruhan arsitektur proses enqueue pesan ini dirancang secara cermat dengan tujuan utama untuk menjaga agar dalam satu sesi interaksi pengguna, sistem hanya memproses satu pesan pada satu waktu. Dengan memberlakukan pemrosesan tunggal per sesi ini, sistem secara fundamental dapat menghindari berbagai masalah yang dapat timbul dari pemrosesan paralel yang tidak terkontrol dalam konteks yang sama. Masalah tersebut mencakup tumpang tindih (*overlapping*) pemrosesan yang dapat menyebabkan inkonsistensi data internal, kondisi balapan (*race conditions*) di mana hasil akhir tidak dapat diprediksi, atau potensi error lainnya yang mungkin muncul jika beberapa pesan dari pengguna yang sama diolah secara bersamaan dan berpotensi memodifikasi state sesi secara konflik.

Dengan demikian, integritas data sesi, urutan logis percakapan, dan akurasi respons dapat terjaga dengan baik.

3. Pemrosesan Pesan oleh Worker

```python
async def _session_worker(self, session_id: str):
    """Worker that processes messages for a session"""
    queue = self.session_queues[session_id]
```

Gambar 3.23 Script Session Worker Function

Setelah pesan pengguna berhasil ditambahkan ke antrian sesi spesifiknya melalui proses *enqueue*, tanggung jawab selanjutnya untuk pengolahan pesan tersebut beralih ke entitas yang disebut worker sesi. Gambar 3.23 menampilkan bahwa setiap sesi pengguna aktif didukung oleh sebuah worker yang berdedikasi atau dialokasikan secara dinamis dari sebuah pool pekerja, yang secara berkelanjutan memantau dan siap mengambil pesan dari antrian sesi yang menjadi tanggung jawabnya untuk diproses lebih lanjut. Keberadaan worker per sesi atau yang dialokasikan untuk sesi ini memastikan isolasi pemrosesan antar pengguna, sehingga aktivitas satu pengguna tidak secara langsung mempengaruhi pengguna lain. Desain *worker* ini secara fundamental memanfaatkan kemampuan pemrograman asinkron Python, di mana setiap worker diimplementasikan sebagai sebuah *coroutine*, yang didefinisikan dengan sintaks async def. Pilihan penggunaan coroutines ini merupakan kunci untuk mencapai performa tinggi dan skalabilitas sistem, terutama dalam kemampuannya menangani sejumlah besar pengguna secara konkuren. *Coroutines* memungkinkan worker untuk melakukan operasi yang bersifat I/O-bound (seperti menunggu respons dari query database, panggilan API eksternal ke layanan embedding, atau interaksi dengan LLM) tanpa memblokir thread eksekusi utama. Sebaliknya, sistem dapat secara efisien beralih ke coroutine (atau worker) lain yang siap berjalan, sehingga memaksimalkan utilisasi sumber daya dan menjaga responsivitas sistem meskipun di bawah beban kerja yang tinggi.

Ketika sebuah *worker* sesi mendeteksi adanya pesan baru yang siap diambil dari antriannya—yang menandakan adanya input baru dari pengguna yang perlu ditangani—serangkaian tindakan terkoordinasi akan dieksekusi oleh worker tersebut. Langkah pertama yang sangat penting sebelum worker memulai pemrosesan inti pesan adalah melakukan validasi internal, yaitu memastikan bahwa tidak ada task atau sub-proses dari pemrosesan pesan sebelumnya untuk sesi yang sama yang masih aktif berjalan atau belum selesai sepenuhnya. Pengecekan internal ini berfungsi sebagai mekanisme pengaman tambahan untuk menjaga integritas dan konsistensi data dalam satu sesi pengguna. Ini mencegah skenario di mana *worker* mungkin secara prematur memulai pemrosesan pesan baru sementara hasil atau efek samping dari pesan sebelumnya belum tuntas, yang berpotensi menyebabkan kondisi tumpang tindih (overlap) atau race condition di dalam konteks operasional worker itu sendiri. Dengan demikian, dijamin bahwa setiap pesan dari pengguna diproses secara atomik dan berurutan dari perspektif worker tersebut.

Setelah kondisi internal worker dipastikan siap dan aman untuk melanjutkan, *worker* akan menjalankan fungsi inti yang bertanggung jawab atas pemrosesan pesan tunggal, yang dalam implementasi sistem ini kemungkinan besar direpresentasikan oleh sebuah metode atau fungsi spesifik seperti _process_single_text. Fungsi ini bukanlah sekadar operasi sederhana, melainkan merangkum dan mengorkestrasi keseluruhan pipeline pengolahan informasi yang komprehensif dan multi-tahap. Di dalam eksekusi fungsi _process_single_text ini, tercakup serangkaian langkah penting yang saling terkait dan berurutan. Selanjutnya, dilakukan pengambilan data konfigurasi tenant yang relevan dengan sesi pengguna tersebut untuk memungkinkan personalisasi dan penyesuaian alur pemrosesan. Kemudian, *pipeline* berlanjut ke tahap krusial seperti pencarian produk yang relevan berdasarkan interpretasi semantik dari input pengguna (seringkali melibatkan pembuatan *embedding* dari *query* dan pencarian di

database vektor), hingga akhirnya melakukan pencocokan promosi dengan cara memeriksa *voucher* aktif dan mencocokkannya dengan produk yang berhasil ditemukan. Eksekusi menyeluruh dari fungsi _process_single_text ini pada akhirnya akan menghasilkan kumpulan informasi, data, atau konteks yang kaya dan terstruktur, yang menjadi fondasi bagi tahap generasi respons oleh *Large Language Model (LLM)* pada langkah berikutnya.

4. Pemrosesan Utama

Tahap "Pemrosesan Utama" merupakan inti dari logika operasional sistem, di mana serangkaian langkah krusial dieksekusi untuk mempersiapkan data dan konteks yang relevan sebelum interaksi lebih lanjut dengan komponen pencarian dan model bahasa.

```
# Load or initialize chat session
chat_messages, chat_histories = [], []
top_k = 5
vector_storage, vector_storage_type = None, None

if session_id != 'test-session':
    chat_messages, chat_histories, vector_storage, vector_storage_type = await self._load_existing_session(session_id,

if not chat_messages:
    chat_messages, vector_storage, vector_storage_type = await self._initialize_from_tenant(session_id, slug)

# Check if this is a promotion query
is_promo_query = any(
    kw in text.lower()
    for kw in ["promo", "voucher", "diskon", "potongan"]
```

Gambar 3.24 Script Process Text Function

Gambar 3.24 menunjukan bahwa proses ini dimulai dengan langkah fundamental, yaitu memverifikasi konektivitas dan validitas kredensial ke layanan-layanan eksternal yang esensial, seperti database vektor untuk pencarian vektor. Verifikasi ini adalah tindakan preventif untuk memastikan bahwa semua jalur komunikasi terbuka dan sistem memiliki otorisasi yang diperlukan sebelum melakukan operasi pengambilan atau pengiriman data. Kegagalan pada tahap ini akan menghentikan proses lebih lanjut dan mungkin memicu mekanisme penanganan kesalahan, seperti memberi notifikasi kepada administrator atau memberikan respons standar kepada pengguna bahwa sistem sedang mengalami kendala teknis.

Setelah konektivitas dipastikan aman, langkah berikutnya adalah mengambil system prompt yang sesuai dari konfigurasi tenant. System prompt ini adalah serangkaian instruksi atau templat dasar yang telah ditentukan sebelumnya dan spesifik untuk setiap tenant atau klien yang dilayani oleh sistem. Instruksi ini sangat vital karena akan digunakan untuk memandu perilaku, gaya bahasa, batasan, dan fokus respons yang akan dihasilkan oleh *Large Language Model* (LLM) di tahap selanjutnya. Misalnya, satu *tenant* mungkin menginginkan respons yang formal dan informatif, sementara tenant lain mungkin preferensi respons yang lebih kasual dan ramah. Pengambilan system prompt yang tepat memastikan bahwa interaksi AI selaras dengan identitas merek dan kebutuhan spesifik tenant tersebut, yang datanya tersimpan dan dikelola dalam basis data konfigurasi tenant.

Selanjutnya, sistem melakukan analisis awal terhadap input pengguna untuk mengidentifikasi apakah pertanyaan pengguna tersebut secara eksplisit atau implisit terkait dengan pencarian promosi. Identifikasi ini umumnya dilakukan dengan memindai kata kunci tertentu dalam pertanyaan pengguna, seperti "promo", "diskon", "*sale*", "potongan harga", "penawaran khusus", atau frasa sejenis lainnya. Deteksi intensi promosi ini merupakan langkah penting karena akan mempengaruhi alur pemrosesan berikutnya, terutama dalam menentukan apakah sistem perlu melakukan pencarian khusus untuk voucher atau penawaran yang relevan, di samping pencarian produk reguler.

Berdasarkan hasil identifikasi intensi dan konteks pertanyaan secara keseluruhan, sistem kemudian akan melakukan proses pencarian produk dan, jika terindikasi adanya kebutuhan atau permintaan promosi, juga melakukan pencarian promosi secara simultan atau berurutan.

Pencarian produk akan melibatkan interaksi dengan penyimpanan berbasis vektor untuk pencarian semantik (*semantic search*), sementara

pencarian promosi akan melibatkan query ke daftar *voucher* atau penawaran aktif yang tersimpan. Keputusan untuk melakukan kedua jenis pencarian ini, atau hanya salah satunya, sangat bergantung pada interpretasi sistem terhadap kebutuhan pengguna pada saat itu. Berikut merupakan cuplikan struktur metadata yang digunakan untuk sistem.

Keseluruhan rangkaian langkah dalam "Pemrosesan Utama" ini secara kolektif mencerminkan logika cerdas yang tertanam dalam sistem, yang memungkinkannya untuk tidak hanya memproses permintaan secara literal, tetapi juga untuk memahami konteks pertanyaan pengguna secara lebih mendalam dan, berdasarkan pemahaman tersebut, menentukan serangkaian tindakan atau alur kerja yang paling tepat dan efisien untuk menghasilkan respons yang paling relevan dan bermanfaat.

5. Pencarian Produk

Salah satu diferensiator utama dan kekuatan inti dari sistem yang dikembangkan ini terletak pada kemampuannya untuk melakukan pencarian produk secara semantik, bukan sekadar pencocokan kata kunci literal. Pendekatan semantik ini memberikan kemudahan signifikan bagi pengguna, karena mereka tidak diharuskan untuk mengingat atau menyebutkan nama produk secara eksak dan lengkap. Pengguna dapat menggunakan bahasa sehari-hari, sinonim, deskripsi fungsional, atau bahkan pertanyaan yang mengandung intensi pembelian tanpa menyebutkan nama produk spesifik, dan sistem tetap mampu menginterpretasikan serta menemukan produk yang relevan. Hal ini menciptakan pengalaman pengguna yang lebih intuitif dan mengurangi friksi dalam proses pencarian informasi produk.

Proses pencarian produk semantik ini melibatkan serangkaian langkah teknis yang terkoordinasi. Langkah pertama adalah transformasi pertanyaan pengguna menjadi representasi numerik yang kaya makna, yaitu embedding vektor telah dilatih pada korpus teks yang sangat besar, sehingga mampu

menangkap nuansa semantik, hubungan kontekstual antar kata, dan inti makna dari teks input. Ketika pengguna mengirimkan pertanyaan, misalnya "minyak untuk masak yang lagi promo", teks ini diumpankan ke model transformer yang kemudian menghasilkan sebuah vektor numerik multidimensi. Vektor ini secara efektif "memetakan" makna pertanyaan pengguna ke dalam sebuah ruang vektor berdimensi tinggi, di mana konsep-konsep yang serupa akan memiliki posisi yang berdekatan.

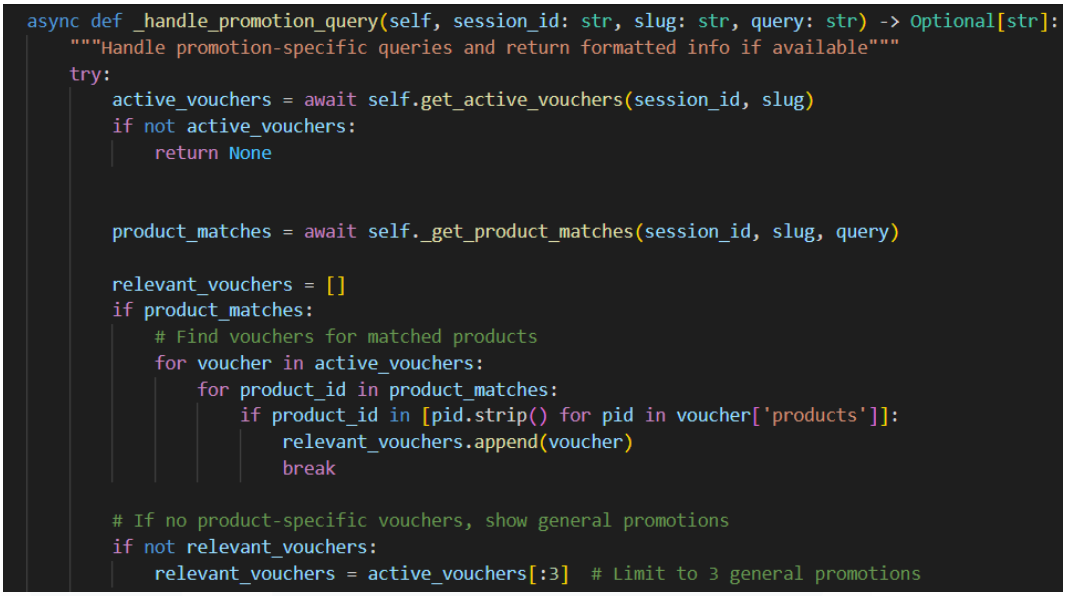
Setelah *embedding* vektor dari pertanyaan pengguna berhasil dibuat, langkah berikutnya adalah menggunakan vektor tersebut untuk melakukan pencarian kemiripan (similarity search) di dalam penyimpanan berbasis vektor, yang berfungsi sebagai database vektor. Sebelumnya, seluruh katalog produk—termasuk nama produk, deskripsi, kategori, dan atribut relevan lainnya—juga telah diproses menjadi embedding vektor menggunakan model transformer yang sama atau kompatibel, dan kemudian disimpan serta diindeks di penyimpanan berbasis vektor. Kemudian, *embedding* pertanyaan pengguna akan dibandingkan dengan miliaran atau jutaan *embedding* produk yang tersimpan, dan mengidentifikasi "dokumen" (dalam hal ini, produk-produk) yang memiliki embedding paling relevan atau paling dekat (misalnya, berdasarkan metrik cosine similarity atau jarak *Euclidean*) dengan embedding pertanyaan. Hasil berupa daftar produk yang diurutkan berdasarkan skor kemiripan secara semantik.

Meskipun pencarian semantik sangat kuat dalam menemukan produk yang relevan secara konseptual, untuk memastikan informasi yang ditampilkan benar-benar tepat sasaran dan sesuai dengan harapan pengguna, hasil awal dari *vector storage* ini kemudian disaring lebih lanjut menggunakan kriteria metadata yang terstruktur. Metadata ini, yang juga tersimpan bersama embedding produk di *vector storage* atau di database relasional seperti *Postgresq*l, mencakup atribut-atribut spesifik seperti

category (misalnya, "Sembako", "Perlengkapan Dapur"), product_name (nama resmi produk), brand, price_range, stock_status, atau bahkan fragmen content penting lainnya dari deskripsi produk. Proses penyaringan ini memungkinkan sistem untuk mempersempit hasil pencarian. Sebagai contoh konkret, jika pengguna mengajukan pertanyaan “Ada diskon minyak goreng?”, sistem akan:

a) Membuat *embedding* dari "diskon minyak goreng" (fokus utama pada "minyak goreng" untuk pencarian produk).
b) Menggunakan *embedding* ini untuk mencari di *vector storage*, yang akan mengembalikan berbagai produk minyak goreng dari berbagai merek dan ukuran karena secara semantik semuanya adalah "minyak goreng".
c) Jika ada informasi tambahan atau preferensi (misalnya, jika sistem mendeteksi merek tertentu dari histori atau jika pertanyaan lebih spesifik), metadata dapat digunakan untuk menyaring. Dalam kasus ini, bahkan tanpa penyaringan metadata yang kompleks, produk seperti "Bimoli Minyak Goreng 2000mL" akan secara otomatis tercocokkan dan muncul sebagai hasil yang relevan karena makna inti "minyak goreng" dalam nama dan deskripsinya sangat sesuai dengan makna pertanyaan pengguna, meskipun pengguna tidak menyebutkan merek "Bimoli" atau ukuran "2000mL" secara eksplisit. Kata "diskon" akan ditangani oleh modul promosi, namun produk relevan ditemukan melalui pencarian semantik ini. Kombinasi pencarian semantik dengan penyaringan metadata ini memastikan bahwa pengguna tidak hanya mendapatkan produk yang relevan secara makna, tetapi juga yang paling sesuai dengan konteks dan batasan spesifik yang mungkin ada.

6. Penanganan Promosi



```python
async def _handle_promotion_query(self, session_id: str, slug: str, query: str) -> Optional[str]:
    """Handle promotion-specific queries and return formatted info if available"""
    try:
        active_vouchers = await self.get_active_vouchers(session_id, slug)
        if not active_vouchers:
            return None


        product_matches = await self._get_product_matches(session_id, slug, query)

        relevant_vouchers = []
        if product_matches:
            # Find vouchers for matched products
            for voucher in active_vouchers:
                for product_id in product_matches:
                    if product_id in [pid.strip() for pid in voucher['products']]:
                        relevant_vouchers.append(voucher)
                        break

        # If no product-specific vouchers, show general promotions
        if not relevant_vouchers:
            relevant_vouchers = active_vouchers[:3]  # Limit to 3 general promotions
```

Gambar 3.25 Script Handle Promotion Query Function

Apabila sistem, melalui mekanisme deteksi intensi pada tahap "Pemrosesan Utama", berhasil mengidentifikasi bahwa pertanyaan atau input pengguna mengindikasikan ketertarikan terhadap promosi, diskon, atau penawaran khusus—misalnya melalui penggunaan kata kunci seperti "diskon", "promo", "voucher", "murah", atau konteks percakapan yang mengarah ke sana, maka sebuah alur kerja spesifik untuk "Penanganan Promosi" akan diaktifkan. Gambar 3.25 menampilkan langkah pertama dalam alur kerja ini adalah mengambil daftar lengkap voucher yang sedang aktif dari Postgresql. Proses pengambilan ini tidak sembarangan; sistem akan menjalankan query yang cermat untuk memastikan hanya voucher yang memenuhi kriteria "aktif" yang disertakan. Kriteria ini umumnya mencakup pemeriksaan validitas tanggal (di mana tanggal saat ini berada dalam rentang start_date dan end_date voucher), status voucher yang memang diatur sebagai 'aktif' (bukan 'nonaktif', 'kadaluwarsa', atau 'arsip'), dan mungkin juga kriteria lain seperti batas penggunaan yang belum tercapai atau ketersediaan stok promosi jika ada. Informasi yang diambil dari Postgresql untuk setiap voucher aktif ini sangat detail, meliputi ID unik

voucher, teks deskriptif promosi yang akan ditampilkan kepada pengguna, jenis diskon (misalnya, persentase, potongan harga tetap), nilai diskon, syarat dan ketentuan berlaku (seperti minimum pembelian), serta yang paling penting, daftar produk spesifik (berdasarkan ID produk) atau kategori produk yang tercakup dalam promosi tersebut.

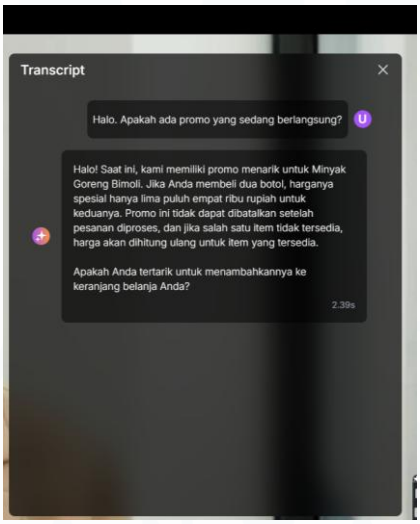
Setelah daftar voucher aktif berhasil diperoleh, sistem melanjutkan ke tahap pencocokan yang krusial. Pada tahap ini, dilakukan pengecekan silang untuk menentukan apakah produk-produk yang telah diidentifikasi sebagai relevan bagi pengguna dari hasil proses "Pencarian Produk" sebelumnya (langkah 5), termasuk dalam cakupan salah satu atau lebih voucher aktif tersebut. Ini melibatkan iterasi melalui setiap voucher aktif dan membandingkan kriteria kelayakan produknya (baik itu daftar ID produk spesifik, kategori produk, atau bahkan merek tertentu yang tercantum dalam detail voucher) dengan daftar produk yang relevan bagi pengguna. Sebagai contoh, jika hasil pencarian produk sebelumnya adalah "Bimoli Minyak Goreng 2000mL" dan "Sania Minyak Goreng 1000mL", dan terdapat voucher aktif "Diskon Rp 5.000 untuk semua produk Bimoli", maka sistem akan mendeteksi kecocokan antara voucher tersebut dengan produk "Bimoli Minyak Goreng 2000mL". Logika pencocokan ini bisa sederhana (berdasarkan kesamaan ID produk) atau lebih kompleks (melibatkan pencocokan kategori atau atribut produk lainnya).

Apabila dari proses pengecekan tersebut ditemukan satu atau lebih kecocokan, artinya ada voucher aktif yang relevan dengan produk yang sedang dipertimbangkan atau ditanyakan oleh pengguna, maka informasi detail mengenai voucher yang cocok tersebut akan dikompilasi secara sistematis untuk dijadikan bagian dari konteks yang akan diberikan kepada Large Language Model (LLM) untuk menghasilkan respons akhir. Informasi yang dikompilasi ini mencakup elemen-elemen penting seperti teks promosi yang menarik dan mudah dipahami (misalnya, "Dapatkan

potongan harga spesial Rp 10.000 untuk Bimoli Minyak Goreng 2000mL!"), periode tanggal berlaku voucher ("Berlaku mulai 1 Juni 2025 hingga 15 Juni 2025") untuk memberikan informasi urgensi dan validitas, serta penjelasan mengenai produk terkait yang spesifik dari hasil pencarian pengguna yang memenuhi syarat untuk promosi tersebut. Selain itu, informasi lain seperti syarat dan ketentuan utama atau cara klaim voucher juga dapat disertakan.

Pendekatan proaktif dalam penanganan promosi ini memastikan bahwa pengguna tidak hanya menerima informasi produk yang mereka cari, tetapi juga secara kontekstual disuguhkan dengan promosi atau diskon relevan yang dapat langsung mereka manfaatkan. Hal ini tidak hanya meningkatkan nilai interaksi bagi pengguna dengan memberikan potensi penghematan, tetapi juga berpotensi meningkatkan angka konversi penjualan bagi tenant atau brand, karena penawaran yang tepat waktu dan relevan cenderung lebih efektif dalam mendorong keputusan pembelian. Dengan demikian, sistem bertindak lebih dari sekadar penjawab pertanyaan, melainkan sebagai asisten belanja yang cerdas dan membantu.

7. Generasi Respons



Gambar 3.26 Tampilan akhir dari hasil pengembangan RAG

Semua elemen seperti system prompt, konten pengguna, data produk, dan detail promosi kemudian digabungkan menjadi satu kesatuan input atau prompt majemuk. Penggabungan ini seringkali mengikuti sebuah templat yang telah dirancang untuk memandu LLM tentang bagaimana cara menggunakan berbagai potongan informasi tersebut. Misalnya, templat tersebut mungkin menginstruksikan LLM untuk pertama-tama merujuk pada pertanyaan pengguna, kemudian menggunakan informasi produk dan promosi yang disediakan untuk merumuskan jawaban, sambil tetap mematuhi arahan dari *system prompt*.

Setelah prompt gabungan ini dikirimkan ke *Large Language Model*, model bahasa tersebut akan memproses input yang kompleks ini dan menghasilkan respons dalam bentuk kalimat alami yang koheren dan kontekstual. Keunggulan LLM terletak pada kemampuannya untuk melakukan sintesis informasi—tidak hanya mengulang data mentah, tetapi merangkai kata-kata menjadi penjelasan, rekomendasi, atau jawaban yang mengalir secara natural. Gambar 3.26 menampilkan respons yang dihasilkan dirancang agar dapat langsung disampaikan kepada pengguna melalui berbagai saluran interaksi, seperti avatar digital yang berbicara atau *chatbot* berbasis teks. Sifat respons yang alami ini membuat interaksi terasa jauh lebih manusiawi, intuitif, dan mudah dipahami oleh pengguna, berbeda dengan interaksi kaku yang mungkin dihasilkan oleh sistem berbasis aturan atau template sederhana. Hal ini secara signifikan meningkatkan kualitas pengalaman pengguna dan efektivitas komunikasi.

3.3 Kendala yang Ditemukan

Selama proses kerja magang di PT NXTX Artificial Intelligence, terdapat beberapa kendala yang ditemukan. Kendala-kendala tersebut mengcakup kompleksitas dalam memproses data yang tidak terstruktur, tuntutan teknis pengembangan model untuk tugas khusus, tantangan dalam integrasi lingkungan

produksi real-time menggunakan API (seperti FastAPI) dan sistem *Retrieval-Augmented Generation* (RAG). Hal ini membutuhkan fokus yang kuat pada integrasi sistem, latensi rendah, dan skalabilitas untuk melayani dasbor analitik yang berorientasi pada klien dan asisten virtual secara efektif.

Berikut merupakan rincian kendala yang dialami secara rinci :

1. Penentuan Tata Letak dan Jenis Visualisasi Dashboard

Salah satu kendala yang dialami adalah pemilihan jenis visualisasi yang tepat. Hal ini dikarenakan penampilan informasi yang terlalu banyak dalam satu layar membuat dashboard sulit dibaca, sehingga kehilangan fungsi utamnaya untuk memberikan insight yang cepat.

2. Penyesuaian dan Evaluasi Model yang Kompleks

Tantangan utama proyek ini adalah kedalaman teknis yang diperlukan untuk melampaui penggunaan API standar. Diperlukan melakukan penyesuaian halus pada model yang telah dilatih sebelumnya (BART), yang merupakan proses komputasi intensif yang memerlukan persiapan data yang cermat dan penyesuaian hiperparameter. Kesulitan utama adalah menciptakan ringkasan referensi berkualitas tinggi dan tidak kaku.

3. Kinerja Real-Time dan Integrasi Sistem pada RAG

Kendala utama pada tugas ini adalah sistem harus bisa menjalankan proses concurrent secara real-time, memahami pertanyaan pengguna, mencari database produk/voucher, mengambil konteks yang relevan dan memberikannya ke LLM untuk menghasilkan respons yang baik. Hal ini membutuhkan fokus dan akurasi yang baik pada pencarian vector storage. start_date dan end_date sangat rentan terhadap kesalahan zona waktu (timezone). Kesalahan kecil dapat menyebabkan promosi ditawarkan terlalu dini atau terlalu lama.

4. Konsistensi Data dari Transrip yang Bervariasi

Teks percakapan dan label kategori seringkali tidak konsisten. Pengguna atau LLM bisa saja membuat kesalahan ketik (misalnya, "ukridaa" vs "ukrida"), menggunakan format berbeda ("Biaya Kuliah" vs. "biaya kuliah!!"), atau menambahkan kata-kata yang tidak relevan.

3.4 Solusi atas Kendala yang Ditemukan

Bagian ini berisi solusi atas kendala yang ditemukan selama proses kerja magang:

1. Penerapan Hirarkis dan Berbasis *Story-Telling* pada Perancangan Dashboard

Penerapan prinsip desain visual dilakukan dimana informasi yang informasi paling penting (ringkasan KPI) diletakkan di bagian paling atas dan paling menonjol. Tata letak dibuat mengalir secara logis.

2. Pendekatan Hibrida untuk Model Ringkasan

Menggunakan bart-large-cnn untuk tugas berat (membuat draf ringkasan) lalu menggunakan model yang lebih ringan dan cerdas dalam mengikuti instruksi untuk "memoles" ringkasan tersebut adalah cara yang sangat cerdas. Prompt seperti "perbaiki ringkasan ini agar lebih alami" secara efektif memanfaatkan kekuatan model instruksi (*instruction-tuned models*) untuk lokalisasi dan peningkatan gaya bahasa

3. Akurasi Validasi Waktu dengan metode Standardisasi

start_date dan end_date pada metadata di *vector storage* digunakan sebagai indikasi apakah voucher tersebut masih berlaku atau tidak. Praktik terbaik yang telah diterapkan dalam proyek ini adalah menyimpan semua tanggal dan waktu di *vector storage* dalam format ISO 8601 lengkap dengan offset zona waktu (contoh: 2025-06-14T23:59:59+07:00).

4. Normalisasi Teks Secara Spesifik

Fungsi normalize_word ini menggunakan *Levenshtein Distance* untuk menangani kesalahan ketik atau variasi ejaan minor. Jika perbedaan kata masih dalam ambang batas toleransi, kata tersebut akan diubah ke bentuk standarnya.