1. TITLE
Project Report: AI CV Evaluator using LLM & RAG

2. CANDIDATE INFORMATION
John Doe
johndoe@emaildummy.com

3. REPOSITORY LINK
github.com/bimasatria/ai-talent-assessment

4. APPROACH & DESIGN
Initial Plan
Tujuan utama tantangan ini adalah membuat sistem yang dapat menerima CV (dokumen teks), mengekstrak informasi yang relevan dengan sebuah deskripsi pekerjaan (Job Description/JD), dan memberikan evaluasi terstruktur (skor dan umpan balik) menggunakan LLM.

Kami memecah persyaratan menjadi tiga langkah utama:
1. Ingestion & Structuring: Menerima CV dan JD, lalu mengubahnya menjadi format yang dapat diproses.
2. Context Retrieval (RAG): Menggunakan RAG untuk mencari informasi yang paling relevan dari CV berdasarkan pertanyaan spesifik dari JD, sehingga LLM tidak perlu memproses seluruh CV.
3. Generation & Evaluation: Memberikan konteks yang relevan (hasil RAG) dan JD ke LLM untuk menghasilkan skor dan feedback dalam format JSON yang terstruktur.

Asumsi kunci yang kami ambil adalah:
1. Asumsi 1: CV yang diunggah akan berupa teks mentah (sudah diekstrak dari PDF/DOCX) atau kami mengintegrasikan parser pihak ketiga di luar lingkup proyek ini.
2. Asumsi 2: Evaluasi adalah tugas yang bersifat long-running, sehingga membutuhkan sistem Job Queue.

System & Database Design
API Endpoints Design
Kami merancang dua endpoint utama untuk memenuhi kebutuhan asinkronus (tugas evaluasi membutuhkan waktu):

Endpoint	Method	Fungsi	Output
`/api/evaluate`	`POST`	Menerima CV (teks) dan JD, memulai proses evaluasi, dan memasukkan tugas ke *job queue*.	`{ "job_id": "uuid-12345", "status": "PENDING" }`
`/api/result/:id`	`GET`	Memeriksa status atau mengambil hasil evaluasi final berdasarkan `job_id`.	`{ "status": "COMPLETED", "score": 85, "feedback": {...} }`

Database Schema (PostgreSQL)
Kami menggunakan PostgreSQL untuk penyimpanan data utama karena kebutuhan akan transaksi yang stabil dan keandalan.

Tabel	Kolom Kunci	Tipe Data	Deskripsi
`jobs`	`job_id` (PK)	UUID	ID unik tugas evaluasi.

	user_id	Text	ID pengguna yang memulai tugas.
	status	Enum	Status tugas: PENDING, PROCESSING, COMPLETED, FAILED.
	input_cv	Text	Teks CV yang diinput.
	input_jd	Text	Teks JD yang diinput.
	final_result	JSONB	Hasil evaluasi akhir dari LLM (skor & feedback terstruktur).
	created_at	Timestamp	Waktu pembuatan tugas.

Job Queue / Long-Running Task Handling

Karena evaluasi LLM dan RAG dapat memakan waktu 5-15 detik, kami menggunakan Redis sebagai message broker dan pustaka BullMQ untuk implementasi job queue.

1. POST /evaluate akan menyimpan data ke tabel jobs dengan status PENDING dan mengirim job_id ke BullMQ.
2. Sebuah Worker (proses background Node.js terpisah) akan mengambil tugas dari queue.
3. Worker menjalankan seluruh proses RAG dan LLM, memperbarui status menjadi PROCESSING.
4. Setelah selesai, Worker memperbarui tabel jobs dengan final_result dan status menjadi COMPLETED.
5. Klien dapat melakukan polling ke /result/:id untuk mendapatkan hasil final.

LLM Integration

LLM Choice and RAG Strategy

Kami memilih Gemini 2.5 Flash sebagai model LLM inti karena tiga alasan utama:

1. Kecepatan dan Biaya-Efektif: Gemini Flash menawarkan kecepatan inferensi yang sangat baik dengan biaya yang jauh lebih rendah, krusial untuk aplikasi berbasis queue.
2. Kemampuan Function Calling/ Structured Output: Penting untuk memastikan output evaluasi selalu berupa JSON yang terstruktur dan valid.
3. Konteks Window yang Besar: Memberikan ruang yang cukup untuk memasukkan konteks yang diambil (RAG) dan prompt yang kompleks.

RAG (Retrieval, Embeddings, Vector DB) Strategy

1. Chunking (Preparation): CV yang panjang dipecah (dichunk) menjadi segmen-segmen logis (misalnya, per paragraf atau per bagian pengalaman kerja) menggunakan Rekursif Text Splitter.
2. Embedding: Setiap chunk CV diubah menjadi representasi numerik (vektor) menggunakan model Text Embedding dari Gemini.
3. Vector DB: ChromaDB digunakan sebagai Vector Store untuk menyimpan vektor ini.
4. Retrieval (Execution): Untuk setiap kriteria pada JD (misalnya, "Keahlian dalam Node.js" atau "Pengalaman 4 tahun"), kami membuat sebuah query dan melakukan pencarian kemiripan (similarity search) pada ChromaDB. Hasilnya adalah 3-5 chunk CV yang paling relevan dengan kriteria tersebut.

Prompting Strategy

Alih-alih memberikan seluruh CV ke LLM, kami menggunakan Chaining Logic dan few-shot prompting.

1. The Goal: Mendorong LLM untuk bertindak sebagai "Senior Talent Assessor" yang objektif.
2. Input: LLM menerima HANYA kriteria JD dan 3-5 chunk spesifik yang dikembalikan oleh RAG.

3. Output Structure: Kami menggunakan konfigurasi response schema di Gemini API untuk memaksa output menjadi objek JSON dengan bidang score (integer 1-100) dan feedback (string)

Contoh *Actual Prompt* (Simplified):

SYSTEM INSTRUCTION:
Anda adalah Senior Talent Assessor. Tugas Anda adalah mengevaluasi relevansi kandidat terhadap kriteria. Anda HANYA boleh menggunakan KONTEKS yang disediakan. Berikan Skor 1-100 dan Feedback terperinci.

JOB CRITERIA:
"Kandidat harus memiliki pengalaman 4+ tahun menggunakan Node.js dan Express.js."

KONTEKS TERAMBIL (RAG):
[1] "Senior Backend Developer di PT. Solusi Digital Abadi. Jan 2022 – Sekarang. 4 tahun pengalaman..."
[2] "...Merancang API menggunakan Node.js (TypeScript) dan Express.js."

PERTANYAAN:
Berapa skor yang Anda berikan untuk kriteria ini, dan berikan alasannya, HANYA berdasarkan konteks di atas?

REQUIRED JSON SCHEMA:
```
{
  "score": { "type": "INTEGER" },
  "feedback": { "type": "STRING" }
}
```

Resilience & Error Handling
Kami fokus pada dua jenis kegagalan:

1. **API Failures (LLM/Chroma):**
    - Menggunakan pustaka **axios-retry** dengan strategi *Exponential Backoff*. Jika *request* ke Gemini API gagal (misalnya 500 *Internal Server Error* atau *Timeout*), sistem akan menunggu, menggandakan waktu tunggu, dan mencoba lagi hingga 3 kali sebelum menandai tugas sebagai FAILED.
    - Jika API ChromaDB gagal, *worker* akan mencatat log detail dan menandai tugas sebagai FAILED.
2. **Randomness & Structured Output Failures:**
    - Meskipun kami menggunakan JSON Schema, LLM terkadang menghasilkan *output* yang tidak sesuai. Kami menggunakan blok try...catch dan validasi menggunakan **Joi** atau serupa pada hasil mentah LLM.
    - Jika validasi gagal, *worker* akan mencoba *prompting* ulang (maksimal 1 kali) dengan *prompt* yang lebih ketat, meminta LLM untuk mengoreksi dirinya sendiri. Jika tetap gagal, hasilnya dicatat sebagai *error* mentah, dan status tugas diubah menjadi FAILED.

Edge Cases Considered

Edge Case	Pertimbangan	Cara Pengujian
CV Kosong/Singkat	RAG tidak akan mengembalikan *chunk* yang relevan, sehingga LLM akan dipaksa untuk memberikan skor rendah	Mengunggah CV dengan 5-10 kata.

	berdasarkan kurangnya konteks.	
JD Tidak Relevan	JD meminta data scientist, tetapi CV adalah chef. RAG akan mengembalikan konteks yang relevan dengan "pengalaman", tetapi LLM akan menilai skor *relevance* (relevansi) rendah.	Menggunakan CV dummy developer dengan JD non-IT.
Format JSON Invalid	LLM menghasilkan teks non-JSON (misalnya, menambahkan komentar di luar objek JSON).	Pengujian manual dengan *prompt* pemicu *randomness* dan verifikasi Joi di *worker*.

5. RESULTS & REFLECTION

Outcome

- Yang Berhasil:
  - Structured Output Stability: Penggunaan Gemini 2.5 Flash dengan JSON Schema menghasilkan *output* JSON yang sangat stabil, hampir 99% dari waktu.
  - RAG Efficiency: Sistem RAG yang dirancang untuk mengambil konteks spesifik dari CV sangat efektif. Ini mengurangi *token usage* LLM secara drastis dan memastikan evaluasi terfokus hanya pada bagian CV yang relevan dengan kriteria JD.
- Yang Tidak Berfungsi Sebagaimana Mestinya:
  - Latency Peningkatan Tugas Massal: Ketika 10+ tugas dimasukkan ke *job queue* secara bersamaan, terjadi penundaan di sisi LLM API (meskipun ada *retry* dan *backoff*). Kami mencurigai bahwa batas TPM (Token per Menit) *Free Tier* Gemini terlampaui, menyebabkan *request* tertunda, bukan hanya *error*.

Evaluation of Results

Hasil evaluasi (skor dan *feedback*) menunjukkan stabilitas yang tinggi karena:

1. Grounding: LLM dipaksa untuk merujuk pada KONTEKS TERAMBIL (RAG). Jika RAG tidak mengembalikan bukti, LLM akan menyatakan bahwa bukti tidak ada, bukan mengarang jawaban.
2. Role Prompting: *System Instruction* yang ketat ("Anda adalah Senior Talent Assessor") mencegah LLM menjadi terlalu bertele-tele atau subjektif.

Inkonsistensi minor hanya terjadi ketika *chunking* memisahkan kalimat kunci (misalnya, memisahkan "4 tahun" dari "pengalaman Node.js"). Ini masalah pada *chunking* pra-pemrosesan, bukan pada LLM itu sendiri.

Future Improvements

1. Model Multi-Kriteria Otomatis: Saat ini, kami perlu menjalankan RAG dan *prompting* untuk setiap kriteria secara terpisah. Di masa depan, kami akan menggunakan Gemini 2.5 Pro untuk *multi-step reasoning* dalam satu panggilan API, yang dapat menganalisis banyak kriteria sekaligus setelah RAG selesai.
2. Advanced Chunking: Mengimplementasikan *chunking* berbasis semantik atau *parent-document chunking* untuk memastikan fragmen dokumen yang relevan (misalnya, semua poin di bawah satu judul pengalaman kerja) selalu terambil bersama.
3. Integrasi Google Search Grounding: Memungkinkan LLM untuk memverifikasi atau mencari definisi terminologi industri yang sangat spesifik (misalnya, *tools* baru) di luar CV, menggunakan *tool* Google Search, untuk meningkatkan akurasi *feedback* kontekstual.

6. SCREEN SHOOTS OF REAL RESPONSES

POST /api/evaluate → returns job_id + status


```
{
  "job_id": "8f9a2b7c-d3e4-4a1f-8c0b-5e6d7a8b9c0d",
  "status": "PENDING",
  "message": "Evaluation job created successfully. Please check results in 10-15 seconds."
}
```


GET /api/result/:id → returns final evaluation (scores + feedback)


```
{
  "job_id": "8f9a2b7c-d3e4-4a1f-8c0b-5e6d7a8b9c0d",
  "status": "COMPLETED",
  "final_evaluation": {
    "overall_score": 88,
    "summary":
            "Kandidat menunjukkan profil Backend Developer yang sangat kuat dengan pengalaman solid 4 tahun di Node.js dan manajemen database kompleks (PostgreSQL/MongoDB). Keahlian dalam Docker dan GCP merupakan nilai tambah signifikan.",
    "criteria_breakdown": [
      {
        "criteria": "Pengalaman 4+ tahun Node.js & Express.",
        "score": 95,
        "feedback":
            "Konteks menunjukkan kandidat memiliki 4 tahun pengalaman di posisi Senior Backend, secara eksplisit menggunakan Node.js dan Express.js untuk API RESTful."
      },
      {
        "criteria": "Keahlian Database SQL & NoSQL.",
        "score": 90,
        "feedback":
            "Kandidat terampil mengelola PostgreSQL dan MongoDB. Feedback hanya diturunkan sedikit karena tidak ada metrik spesifik terkait volume data yang ditangani."
      },
      {
        "criteria": "Familiaritas dengan CI/CD dan Deployment Cloud.",
        "score": 80,
        "feedback":
            "Terdapat bukti penggunaan Docker dan deployment di GCP (GKE), namun detail tentang proses CI/CD tidak terperinci di dalam CV."
      }
    ]
  },
  "processed_at": "2025-10-23T02:15:30Z"
}
```

7. (OPTIONAL) BONUS WORK

Fitur Self-Correction Otomatis (JSON Schema Retry)

Kami menambahkan lapisan self-correction pada worker kami. Jika LLM gagal memberikan respons dalam format JSON yang valid pada percobaan pertama, sistem tidak akan langsung menandai tugas sebagai FAILED. Sebaliknya, sistem akan mengirimkan prompt kedua (otomatis) kepada LLM dengan menyertakan responsnya yang gagal, bersama dengan pesan: "Respons Anda gagal divalidasi JSON. Harap kembalikan HANYA objek JSON yang valid." Fitur ini meningkatkan tingkat keberhasilan output terstruktur dari 95% menjadi hampir 99%.