BAB II

KAJIAN LITERATUR

2.1 Pencatatan Keuangan

Pencatatan keuangan merupakan kegiatan merekam seluruh aktivitas transaksi keuangan secara terstruktur dan berkelanjutan. Pencatatan keuangan bertujuan untuk menyediakan informasi yang akurat mengenai arus kas, posisi keuangan, serta pola pemasukan dan pengeluaran dalam suatu periode tertentu. Informasi ini menjadi dasar penting bagi individu maupun organisasi dalam mengevaluasi kondisi keuangan, mengendalikan pengeluaran, serta merencanakan keputusan finansial secara rasional dan terukur. Dalam konteks literasi keuangan, kebiasaan mencatat transaksi menjadi salah satu indikator penting dari kemampuan individu dalam mengelola keuangan pribadinya secara efektif [3]. Literasi keuangan tidak hanya mencakup pengetahuan dasar seperti pendapatan, pengeluaran, tabungan, dan utang, tetapi juga kemampuan menerapkan pengetahuan tersebut dalam kehidupan sehari-hari secara konsisten dan bertanggung jawab. Literasi keuangan menjadi fondasi utama untuk mencapai stabilitas dan kesejahteraan finansial yang berkelanjutan [12].

Penerapan pencatatan keuangan yang baik mendorong individu untuk memiliki kesadaran finansial yang lebih tinggi terhadap kondisi keuangannya. Melalui pencatatan keuangan yang sistematis, individu dapat mengevaluasi kesalahan dalam pengelolaan keuangan, seperti pemborosan, ketidakefisiensian keuangan, tidak memiliki tabungan dan dana darurat. Selain itu, pencatatan keuangan yang dilakukan secara konsisten dapat membantu individu dalam menciptakan kondisi ekonomi individu yang lebih stabil dan mendukung individu dalam mengambil keputusan yang lebih bijak dalam mengurangi pengeluaran [2].

Selain meningkatkan kesadaran dan pengelolaan keuangan pribadi, pencatatan keuangan juga berperan dalam mendorong peningkatan inklusi keuangan. Inklusi keuangan merujuk pada kondisi di mana individu dan pelaku usaha memiliki akses serta penggunaan produk dan layanan keuangan yang bermanfaat, terjangkau, dan sesuai kebutuhan, seperti transaksi, pembayaran, tabungan, kredit, dan asuransi, yang disediakan secara bertanggung jawab dan berkelanjutan. Layanan tersebut membantu masyarakat mengelola risiko, membangun kekayaan, dan berinvestasi, serta menjadi pendorong pencapaian beberapa

tujuan pembangunan berkelanjutan (SDGs) melalui pemberdayaan ekonomi, pertumbuhan usaha kecil, dan pengurangan kemiskinan [13]. Individu yang memiliki catatan keuangan yang rapi dan terstruktur cenderung lebih memahami kondisi finansialnya, sehingga lebih siap dalam mengakses dan memanfaatkan layanan keuangan formal, seperti produk tabungan, pembiayaan, dan investasi. Dengan demikian, pencatatan keuangan tidak hanya berkontribusi terhadap peningkatan literasi keuangan, tetapi juga menjadi faktor pendukung dalam memperluas inklusi keuangan dan meningkatkan kesejahteraan ekonomi secara berkelanjutan.

2.2 Aplikasi Pengelolaan Keuangan Pribadi

Aplikasi keuangan pribadi dirancang untuk membantu individu dalam mengatur pendapatan, pengeluaran, tabungan, investasi, serta perencanaan keuangan lainnya. Aplikasi tersebut menawarkan fitur-fitur canggih yang tidak hanya memudahkan pengguna dalam memantau keuangan mereka secara real-time, tetapi juga memberikan analisis mendalam tentang pola pengeluaran, kebiasaan menabung, dan peluang investasi [14]. Dengan adanya visualisasi data seperti grafik dan laporan terstruktur, pengguna dapat dengan mudah memahami kondisi keuangan mereka secara menyeluruh. Tujuan utama dari aplikasi tersebut adalah meningkatkan kesadaran finansial, membantu mencapai tujuan keuangan jangka pendek maupun jangka panjang, serta menciptakan kebiasaan pengelolaan uang yang lebih disiplin dan terarah.

Berikut manfaat yang dapat diberikan dari aplikasi keuangan pribadi :

1. Otomatisasi keuangan

   Merupakan kemampuan untuk mencatat transaksi secara otomatis yang dapat terhubung langsung dengan rekening bank, kartu kredit, atau dompet digital pengguna, sehingga setiap transaksi yang dilakukan seperti pembelian, transfer, atau pembayaran tagihan akan tercatat secara otomatis tanpa perlu *input* secara manual [14].

2. Analisis pola pengeluaran

   Memungkinkan pengguna untuk melihat pola pengeluaran mereka secara detail serta dapat mengkategorikan pengeluaran berdasarkan jenis pengeluaran dan menampilkan dalam bentuk grafik [14].

3. Pengingat tagihan

   Memungkinkan pengguna untuk mengatur pengingat tagihan rutin, seperti listrik, air, internet, atau cicilan kartu kredit [14].

2.3 *Artificial Intelligence* (AI)

Artificial Intelligence (AI) merupakan bidang dalam ilmu komputer yang berfokus pada pengembangan sistem yang mampu meniru kemampuan kognitif manusia, seperti penalaran, pembelajaran, pemahaman bahasa, dan pengambilan keputusan. Salah satu gagasan awal dalam pengembangan *Artificial Intelligence* (AI) adalah penekanannya pada kemampuan sistem untuk menjalankan proses kognitif tingkat tinggi. Fokus AI tidak terletak pada kemampuan dasar seperti persepsi sensorik atau keterampilan motorik yang juga dimiliki oleh makhluk hidup lain, melainkan pada kapasitas untuk melakukan penalaran bertahap (*multi-step reasoning*), memahami bahasa manusia, serta menghasilkan ide, solusi, atau keluaran baru secara mandiri [15].

Kecerdasan buatan dalam perkembangannya memiliki beberapa pendekatan dan cabang utama yang dirancang untuk merepresentasikan berbagai aspek kecerdasan manusia. *Machine learning* berperan sebagai pendekatan dasar yang memungkinkan sistem mempelajari pola dari data dan menyesuaikan kinerjanya secara adaptif. Dari pendekatan ini berkembang *deep learning* yang memanfaatkan jaringan saraf tiruan berlapis untuk memproses data berskala besar dan kompleks secara lebih mendalam. Kemampuan untuk memahami dan mengolah bahasa manusia diwujudkan melalui *natural language processing*, sedangkan kemampuan untuk mengenali dan menafsirkan informasi visual direalisasikan melalui *computer vision* yang memungkinkan sistem mengidentifikasi objek, pola, dan kondisi dari citra maupun video. Selain itu, *speech recognition* dikembangkan untuk memungkinkan sistem mengenali, mengonversi, dan memahami ucapan manusia menjadi data digital yang dapat diproses lebih lanjut [16].

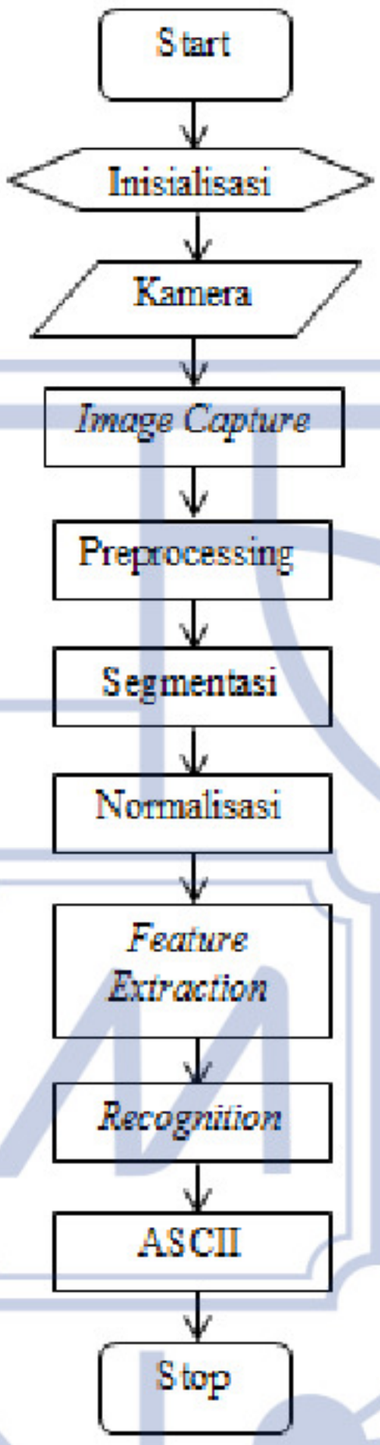
Perkembangan *Artificial Intelligence* (AI) yang pesat telah memberikan kontribusi signifikan dalam meningkatkan efisiensi, akurasi, dan produktivitas di berbagai sektor. AI mampu mengotomatisasi tugas-tugas repetitif, mempercepat proses analisis data dalam skala besar, serta mengurangi kesalahan manusia dalam pengambilan keputusan. Selain itu, AI mendukung peningkatan kualitas layanan melalui sistem yang bersifat adaptif dan personal, seperti asisten virtual, sistem rekomendasi, dan analitik prediktif. Dalam konteks organisasi dan individu, pemanfaatan AI membantu pengambilan keputusan yang lebih cepat dan berbasis data, meningkatkan efisiensi operasional, serta membuka peluang inovasi baru dalam penyelesaian permasalahan kompleks [17].

2.4 *Optical Character Recognition* (OCR)

Optical Character Recognition (OCR) merupakan teknologi yang memungkinkan konversi gambar berisi teks cetak atau tulisan tangan menjadi format digital yang dapat diproses oleh mesin [18]. Proses OCR secara umum terdiri dari dua komponen utama: text detection dan text recognition. Text detection bertujuan untuk mengidentifikasi dan mengekstraksi region dalam gambar yang mengandung teks melalui teknik object detection atau segmentation, sedangkan text recognition berfokus pada pengenalan dan konversi konten teks dalam region yang terdeteksi menjadi string karakter yang dapat dibaca [42]. OCR memainkan peran krusial dalam digitalisasi dokumen, meningkatkan produktivitas, aksesibilitas, dan preservasi arsip historis dengan mengotomatisasi ekstraksi teks dari dokumen fisik sehingga memungkinkan pencarian, editing, dan analisis informasi tekstual secara efisien [19]. Dalam konteks yang lebih luas, OCR menjadi fondasi bagi berbagai aplikasi seperti autonomous driving untuk membaca rambu lalu lintas, information security detection untuk mendeteksi teks berbahaya dalam gambar, serta digitalisasi arsip institusional untuk pengelolaan dokumen historis dalam jumlah besar.

Aplikasi OCR modern mencakup spektrum yang luas mulai dari *document digitization*, *automated data entry*, hingga *intelligent document processing* dalam berbagai sektor industri seperti perbankan, kesehatan, dan pemerintahan. Dalam konteks *document understanding*, OCR menjadi komponen fundamental yang memungkinkan sistem AI mengekstraksi informasi terstruktur dari dokumen tidak terstruktur seperti *invoice*, kontrak, dan formulir [7]. Integrasi OCR dengan *Natural Language Processing* (NLP) memungkinkan untuk pemahaman dokumen yang lebih mendalam, di mana teks yang diekstraksi dapat dianalisis lebih lanjut untuk *information extraction*, *named entity recognition*, dan *semantic analysis*. Namun demikian, OCR masih menghadapi berbagai tantangan termasuk penanganan dokumen dengan kualitas rendah, variasi *font* dan gaya tulisan tangan, teks dalam orientasi berbeda (*rotated* atau *curved text*), serta *multilingual* dan *multi-script recognition* [19], [20]. Tantangan khusus juga muncul untuk bahasa dengan karakteristik ortografis kompleks seperti bahasa Arab yang memiliki struktur morfologi kaya, variasi kontekstual bentuk huruf, dan gaya penulisan kursif yang menghasilkan overlapping strokes [19]. Perkembangan *Large Language Model* (LLM) dan *Vision-Language Models* (VLMs) membuka peluang baru dalam OCR. Model-model ini mampu mengenali teks langsung dari gambar secara utuh tanpa perlu memotong area teks terlebih dahulu, karena dapat memahami hubungan antara informasi visual dan tekstual secara bersamaan. Selain itu, teknik *fine-tuning* yang efisien seperti DLoRA-TrOCR

memungkinkan model OCR diadaptasi untuk kebutuhan spesifik dengan hanya melatih ulang sebagian kecil parameter (sekitar 0.6%), sehingga lebih hemat sumber daya komputasi tanpa mengurangi akurasi model tersebut [21].



Gambar 2.1 Proses Kerja OCR [22]

OCR bekerja melalui beberapa tahapan yang saling berkaitan untuk menghasilkan pengenalan karakter yang akurat. Adapun tahapan-tahapan dalam proses kerja OCR adalah sebagai berikut:

1. Inisiasi

   Merupakan proses awal dari OCR yang berfungsi untuk menyiapkan semua kebutuhan yang diperlukan oleh sistem [22].

2. Kamera/ *Image Capture*

   Merupakan proses pengambilan gambar [22].

3. *Preprocessing*

   Merupakan proses untuk menghilangkan bagian citra yang tidak diperlukan agar lebih mudah dikenali [22].

4. Segmentasi

Berfungsi untuk memisahkan bagian gambar menjadi baris, kata, atau karakter [22].

5. Normalisasi

   Berfungsi untuk menyamakan ukuran, bentuk, atau orientasi karakter [22].

6. *Feature extraction*

   Berfungsi untuk mengambil ciri penting dari karakter [22].

7. *Recognition*

   Proses pengenalan karakter berdasarkan fitur yang telah diekstraksi [22].

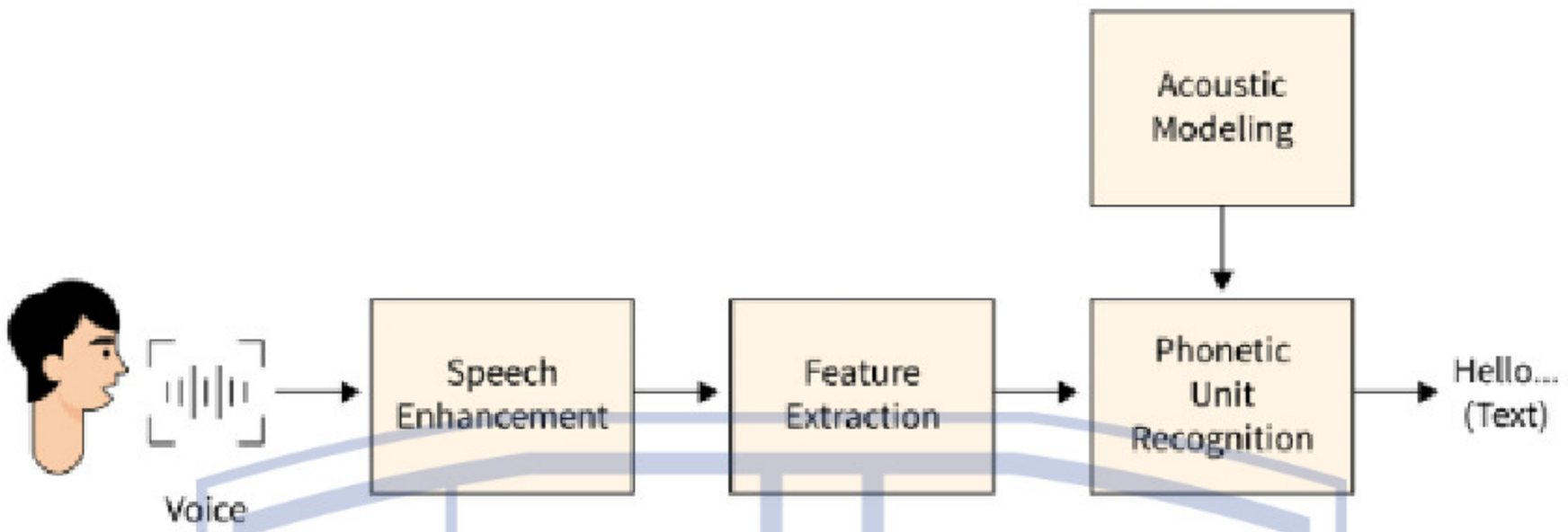
8. *ASCII*

   Mengubah hasil pengenalan menjadi kode teks [22].

2.5 *Speech Recognition*

Speech Recognition merupakan teknologi komputer yang berfungsi untuk mengenali, memahami, dan mengonversi suara manusia menjadi teks secara otomatis [23]. *Speech Recognition* memanfaatkan kecerdasan buatan untuk mengoptimalkan kinerja sistem dalam memproses ucapan manusia, sehingga sistem mampu memahami, mengidentifikasi, dan menginterpretasikan suara dengan lebih tepat dan efisien [9]. Teknologi ini telah diimplementasikan dalam asisten suara virtual seperti Siri, Google Assistant, dan Alexa, sehingga memudahkan pengguna untuk mengakses informasi dan mengontrol perangkat rumah pintar hanya dengan suara mereka [24]

Dengan bantuan AI, akurasi pengenalan suara terus meningkat, sehingga mampu mengenali berbagai aksen, intonasi, dan kondisi lingkungan. Hal ini menjadikan *speech recognition* sebagai komponen penting dalam pengembangan sistem interaksi manusia–komputer yang lebih cerdas, responsif, dan *user*-friendly.



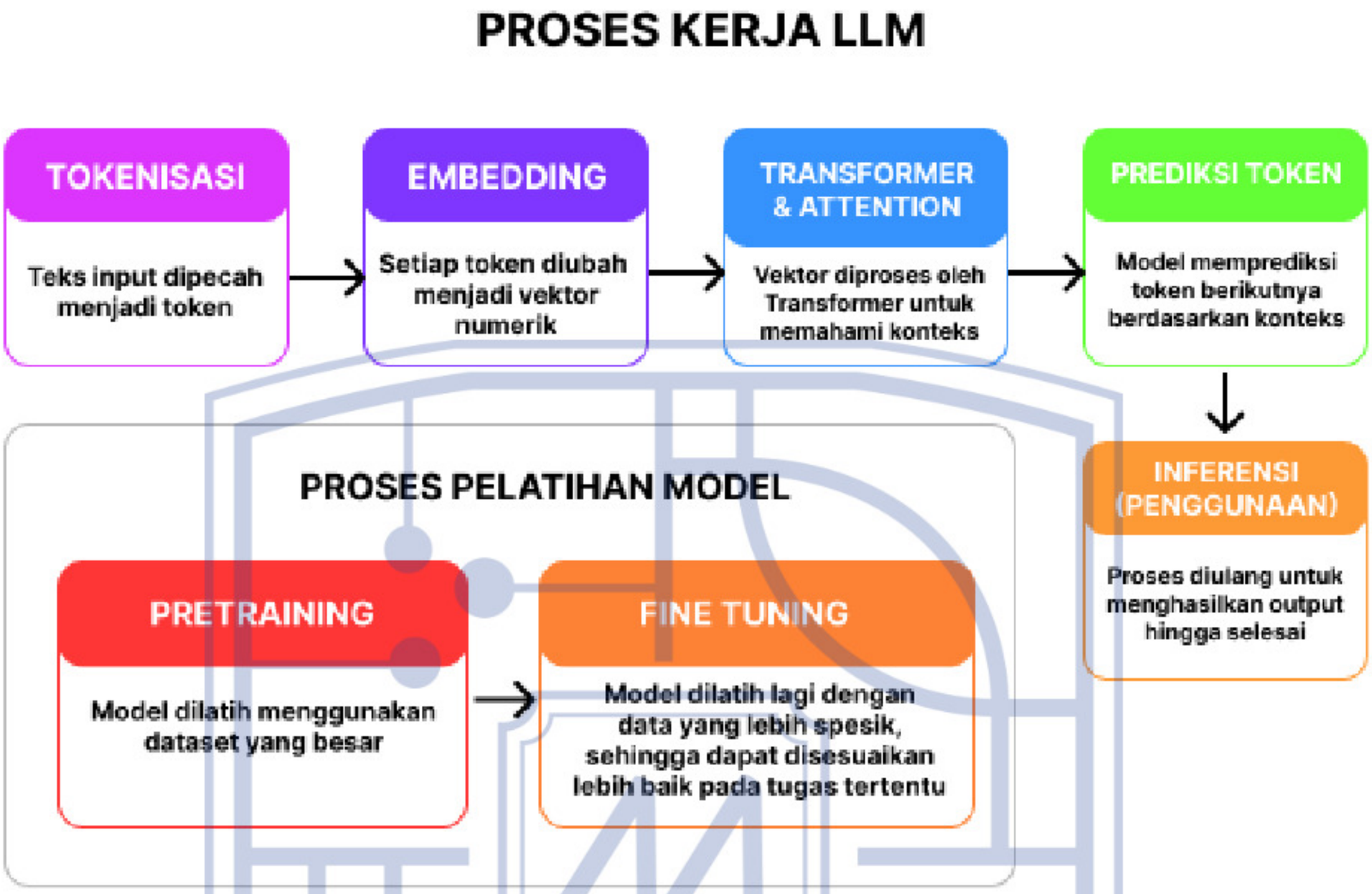
Gambar 2.2 Proses *Speech Recognition* [25]

Cara kerja *speech recognition* dimulai dari pengambilan sinyal suara melalui proses sampling yang mengubah sinyal analog menjadi digital, diikuti dengan proses *endpoint detection* untuk memisahkan bagian ucapan dari noise serta menentukan titik awal dan akhir ujaran. Selanjutnya dilakukan ekstraksi ciri ucapan (*feature extraction*) yang stabil dan representative. Pada tahap pelatihan (*training*), parameter ciri ucapan disimpan sebagai templat referensi dalam basis data. Templat referensi adalah kumpulan data atau model ciri suara yang diperoleh dari hasil pelatihan sistem pengenalan ucapan. Pada tahap pengenalan (*recognition*), ciri ucapan masukan dibandingkan dengan templat referensi menggunakan metode pencocokan tertentu dan hasil dengan tingkat kemiripan tertinggi ditetapkan sebagai hasil pengenalan [9].

2.6 *Large Language Model* (LLM)

Large Language Model (LLM) merupakan kategori model bahasa yang menggunakan jaringan saraf (*neural network*) dengan jumlah parameter yang sangat besar serta dirancang untuk menangani berbagai tugas pemrosesan bahasa alami (NLP). Model ini dilatih ulang dengan menggunakan sejumlah besar data teks tanpa label melalui pendekatan *self-supervised* [26]. LLM memiliki kemampuan untuk menangkap konteks, makna, serta hubungan antar kata dalam suatu percakapan, sehingga mampu menghasilkan respons yang relevan dan kontekstual. Dalam aplikasi pencatatan keuangan, LLM memungkinkan pengguna untuk memasukkan transaksi dalam bentuk kalimat alami, yang kemudian dipahami dan dikonversi menjadi data transaksi terstruktur. Kemampuan ini menjadikan LLM sebagai komponen utama dalam pengembangan sistem pencatatan keuangan otomatis

yang lebih fleksibel, adaptif, dan personal dibandingkan sistem berbasis formulir konvensional.

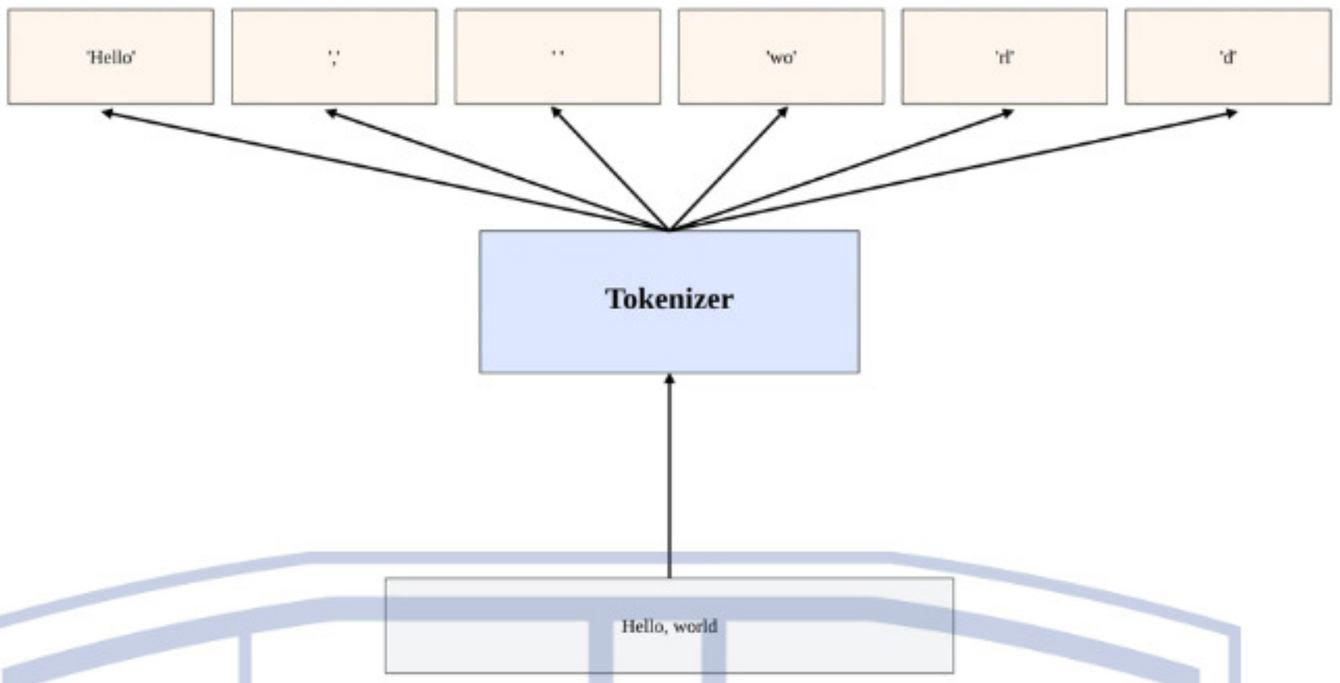


Gambar 2.3 Proses Kerja LLM

Berikut merupakan penjelasan dari alur kerja dari LLM:

1. Tokenisasi

Tokenisasi merupakan tahap awal dalam pemrosesan teks pada *Large Language Model*, di mana teks masukan dipecah menjadi unit-unit kecil yang disebut *token*. *Token* dapat berupa kata utuh, potongan kata (*subword*), atau bahkan karakter, tergantung pada metode tokenisasi yang digunakan. Teknik tokenisasi modern seperti *Byte Pair Encoding* (BPE) dan *WordPiece* dirancang untuk mengatasi permasalahan kosakata yang sangat besar serta menangani kata-kata baru (*out-of-vocabulary words*) secara efisien. Dengan pendekatan ini, LLM mampu memahami berbagai variasi bahasa alami tanpa harus menyimpan seluruh kata dalam kamus eksplisit [27], [28].



Gambar 2.4 Tokenisasi [29]

Dalam konteks aplikasi pencatatan keuangan, tokenisasi memungkinkan sistem memecah kalimat *input* pengguna seperti "beli kopi 25 ribu hari ini" menjadi *token-token* bermakna yang dapat dianalisis lebih lanjut. Proses ini menjadi fondasi penting karena kualitas tokenisasi sangat memengaruhi kemampuan model dalam memahami konteks dan maksud pengguna. Tokenisasi yang baik akan meningkatkan akurasi pemetaan informasi ke dalam struktur data transaksi yang dibutuhkan sistem [30].

2. *Embedding*

Embedding merupakan proses mengubah *token* hasil tokenisasi menjadi representasi numerik dalam bentuk vektor berdimensi tinggi. Representasi ini memungkinkan komputer memahami makna kata secara matematis, di mana *token* yang memiliki makna serupa akan memiliki vektor *embedding* yang berdekatan dalam ruang vektor. Berbeda dengan representasi klasik seperti *one-hot encoding*, *embedding* pada LLM bersifat kontekstual, artinya makna *token* dapat berubah tergantung pada konteks kalimatnya [31].

Dalam LLM berbasis Transformer, *embedding* tidak hanya merepresentasikan makna *token*, tetapi juga digabungkan dengan positional encoding untuk menyimpan informasi urutan kata. Hal ini sangat penting karena arsitektur Transformer tidak memproses data secara berurutan seperti RNN [32]. Dengan *embedding* yang kontekstual dan kaya informasi, LLM mampu memahami perbedaan makna kata yang sama dalam konteks transaksi keuangan yang berbeda, misalnya antara "transfer" sebagai kata kerja dan "transfer" sebagai jenis transaksi [33]

3. *Transformer and Attention*

Arsitektur Transformer merupakan inti dari *Large Language Model* modern. Berbeda dengan pendekatan sekuensial tradisional, Transformer menggunakan mekanisme *self-attention* untuk mempelajari hubungan antar *token* dalam satu teks secara paralel. *Self-attention* memungkinkan model menentukan seberapa penting setiap *token* terhadap *token* lainnya, sehingga konteks global kalimat dapat dipahami secara efisien dan akurat [34]. Mekanisme *attention* sangat relevan dalam pemrosesan bahasa alami yang kompleks, termasuk *input* transaksi keuangan berbentuk kalimat bebas. Dengan attention, model dapat mengenali hubungan antara nominal, kategori, dan waktu dalam satu pernyataan pengguna [35]. Hal ini membuat LLM mampu mengekstraksi informasi penting secara lebih presisi dibandingkan pendekatan berbasis aturan atau formulir statis.

4. *Pretraining*

Pretraining merupakan tahap pelatihan awal LLM menggunakan data teks dalam jumlah sangat besar dengan pendekatan *self-supervised* learning. Pada tahap ini, model dilatih untuk memprediksi *token* berikutnya berdasarkan konteks sebelumnya, sehingga secara tidak langsung mempelajari struktur bahasa, tata bahasa, serta pengetahuan umum [36]. *Pretraining* memungkinkan model memperoleh kemampuan bahasa yang bersifat umum sebelum digunakan untuk tugas spesifik.

Keunggulan utama tahap *pretraining* adalah kemampuannya menghasilkan model yang fleksibel dan dapat digunakan ulang (*general-purpose model*) [37]. Dalam konteks sistem pencatatan keuangan, *pretraining* memungkinkan LLM memahami bahasa sehari-hari pengguna tanpa perlu dilatih dari nol, sehingga pengembangan sistem menjadi lebih efisien dan skalabel.

5. *Fine-Tuning*

Fine-tuning merupakan tahap penyesuaian LLM menggunakan data spesifik sesuai domain aplikasi tertentu. Pada tahap ini, model yang telah dilatih ulang dengan dataset yang lebih kecil namun relevan, seperti data transaksi keuangan atau contoh percakapan pengguna [38]. Proses ini bertujuan untuk meningkatkan akurasi, relevansi respons, serta kesesuaian perilaku model dengan kebutuhan sistem.

Selain *fine-tuning* berbasis data, pendekatan modern juga memanfaatkan *human feedback* untuk meningkatkan kualitas respons model, baik dari sisi keamanan maupun kegunaan [38], [39]. Dalam sistem pencatatan keuangan otomatis, *fine-tuning*

memungkinkan LLM memahami format transaksi, istilah finansial, dan kebiasaan pengguna secara lebih personal dan kontekstual.

6. Inferensi (Generasi Teks)

Inferensi merupakan tahap penggunaan LLM untuk menghasilkan teks berdasarkan masukan pengguna. Pada tahap ini, teks masukan terlebih dahulu diproses melalui tokenisasi dan *embedding* untuk diubah menjadi representasi numerik yang dapat dipahami oleh model [40]. Inferensi memungkinkan LLM memberikan jawaban yang bersifat kontekstual, koheren, dan menyerupai bahasa manusia.

Proses generasi teks dalam inferensi dilakukan secara autoregresif, yaitu *token* dihasilkan satu per satu, di mana setiap *token* baru bergantung pada *token-token* sebelumnya yang telah dihasilkan [41]. Dalam praktiknya, inferensi melibatkan metode decoding tertentu seperti *greedy decoding*, *beam search*, *top-k sampling*, atau *top-p (nucleus) sampling* untuk menyeimbangkan antara koherensi dan keberagaman teks yang dihasilkan [42]. Pemilihan metode decoding sangat memengaruhi kualitas keluaran, terutama dalam aplikasi interaktif yang membutuhkan respons yang akurat dan relevan secara kontekstual [44]. Dalam sistem pencatatan keuangan otomatis, tahap inferensi memungkinkan LLM mengekstraksi informasi penting dari bahasa alami pengguna dan menghasilkan keluaran berupa data transaksi terstruktur secara konsisten dan efisien [35].

2.6.1 OpenAI GPT-5

ChatGPT merupakan kependekan dari *Chat Generative Pre-trained Transformer*. GPT berarti model yang dilatih lebih dulu dengan arsitektur transformer, sehingga model dapat menghasilkan teks baru secara natural dan relevan. Jadi ChatGPT adalah *chatbot* berbasis kecerdasan buatan yang bisa memahami dan menghasilkan teks layaknya manusia. ChatGPT dikembangkan oleh OpenAI di San Francisco, Amerika Serikat yang didirikan pada Desember 2015 [43].

GPT-5 merupakan generasi terbaru dari model bahasa yang dikembangkan oleh *OpenAI*. Model ini menggantikan beberapa model sebelumnya, seperti *GPT-4o*, *OpenAI o3*, *OpenAI o4-mini*, *GPT-4.1*, dan *GPT-4.5* [44]. Dalam penggunaannya, pengguna hanya perlu memasukkan pertanyaan atau perintah, kemudian sistem akan memproses *input* tersebut secara otomatis. *GPT-5* mampu menerapkan kemampuan penalaran (*reasoning*) secara adaptif sesuai dengan kompleksitas pertanyaan, sehingga dapat menghasilkan jawaban yang lebih tepat dan informatif.

GPT-5 menunjukkan peningkatan signifikan dalam kemampuan mengikuti instruksi dan penggunaan alat berbasis agen. Model ini lebih efektif dalam menangani tugas kompleks dan dinamis, serta mampu menjalankan instruksi dengan lebih akurat dan menyelesaikan pekerjaan secara lebih komprehensif [44]. Selain itu, *GPT-5* juga mengimplementasikan mekanisme optimasi dalam efisiensi komputasi dan manajemen konteks, sehingga mampu memproses *input* dengan panjang yang lebih besar tanpa kehilangan relevansi informasi. Dengan kemampuan tersebut, *GPT-5* menjadi solusi yang andal dalam mendukung pengembangan sistem berbasis kecerdasan buatan yang memerlukan efisiensi, ketepatan, dan fleksibilitas dalam pemrosesan informasi.

ChatGPT dilatih menggunakan metode *Reinforcement Learning from Human Feedback* (RLHF), yaitu teknik pembelajaran mesin yang memanfaatkan umpan balik manusia untuk meningkatkan kualitas respons model. Proses pelatihan diawali dengan tahap *supervised fine-tuning*, di mana pelatih manusia menyediakan contoh percakapan ideal antara pengguna dan asisten AI. Contoh-contoh ini digunakan untuk membentuk dasar perilaku awal model agar dapat menghasilkan respons yang sesuai dengan ekspektasi manusia. Selanjutnya, model menghasilkan beberapa kandidat jawaban untuk satu instruksi yang sama. Jawaban-jawaban tersebut kemudian dievaluasi dan diberi peringkat oleh pelatih manusia berdasarkan aspek seperti relevansi, kejelasan, keamanan, dan kualitas informasi. Hasil peringkat ini digunakan untuk membangun *reward model*, yaitu model yang bertugas meniru preferensi manusia dalam menilai kualitas respons [45].

Pada tahap berikutnya, proses *reinforcement learning* diterapkan dengan memanfaatkan reward model tersebut sebagai sinyal umpan balik. Salah satu pendekatan yang umum digunakan adalah optimasi berbasis kebijakan seperti *Proximal Policy Optimization* (PPO), yang membantu model memperbaiki responsnya secara bertahap agar lebih selaras dengan preferensi manusia. Melalui proses iteratif ini, model tidak hanya belajar menghasilkan jawaban yang benar secara faktual, tetapi juga lebih aman, sopan, dan sesuai dengan konteks penggunaan [45].

Penggunaan *GPT-5* dinilai unggul dalam menghasilkan kualitas analisis yang baik daripada versi *GPT* lainnya. Pada penelitian terdahulu yang menggunakan beberapa model *GPT* sebagai objek eksperimen, *GPT-5 Mini* menghasilkan performa terbaik dengan nilai *overall score* tertinggi dibandingkan *GPT-4.1 Mini* dan *GPT-4o Mini*.

Knowledge	Kes. Chunk	Kel. Analisis	Rel. Analisis	Acc	Struktur Penjelas	All Score
GPT 5-mini	6,615	5,72	8,078	7,191	8,666	7,264
GPT 4.1 mini	5,073	3,9	7,177	5,784	8,01	5,996
GPT 4o mini	3,881	2,871	6,837	4,89	6,944	5,09

Gambar 2.5 Performa Model GPT [46]

Model tersebut mampu menghasilkan analisis yang lebih lengkap, relevan, akurat, serta terstruktur sehingga memberikan kualitas interpretasi yang lebih baik pada tugas analisis teks. Oleh karena itu, *GPT-5* dinilai sebagai model yang lebih efektif dalam menangani tugas analisis teks yang membutuhkan pemahaman konteks, penalaran, dan penyusunan respons yang terstruktur [46].

2.7 AI *Guardrail*

AI *Guardrail* adalah kerangka kerja yang dirancang untuk memastikan sistem AI beroperasi secara aman dan bertanggung jawab dalam batasan yang telah ditentukan [47]. Guardrails berfungsi sebagai mekanisme pengawasan yang mengatur perilaku model AI agar tetap menghasilkan keluaran yang relevan, akurat, dan sesuai dengan tujuan sistem. Dalam pengembangan aplikasi berbasis *Large Language Model* (LLM), AI *Guardrail* menjadi komponen penting untuk meningkatkan keamanan, keandalan, dan kualitas layanan yang diberikan kepada pengguna. Oleh karena itu, AI *Guardrails* menjadi salah satu komponen penting dalam pengembangan aplikasi berbasis AI karena dapat meningkatkan kualitas layanan, mengurangi risiko kesalahan informasi, serta membangun kepercayaan pengguna terhadap sistem yang dikembangkan.

AI *Guardrails* dirancang untuk melindungi ancaman seperti :

1. Prompt Injection dan Jailbreak

   Upaya pengguna untuk memberikan instruksi tertentu yang bertujuan memanipulasi perilaku AI agar mengabaikan aturan yang telah ditetapkan dan menghasilkan respons yang tidak seharusnya diberikan [48] .
2. Paparan Informasi Sensitif

   Risiko ketika sistem AI menghasilkan atau mengungkapkan informasi rahasia, seperti data pribadi pengguna, informasi perusahaan, atau data sensitif lainnya yang seharusnya tidak dapat diakses oleh pihak lain [48].
3. Misinformasi dan Konten Berbahaya

Kondisi ketika AI menghasilkan informasi yang tidak akurat, menyesatkan, mengandung ujaran kebencian, atau konten lain yang dapat merugikan pengguna [48].

4. Perilaku Model yang Tidak Dapat Diprediksi

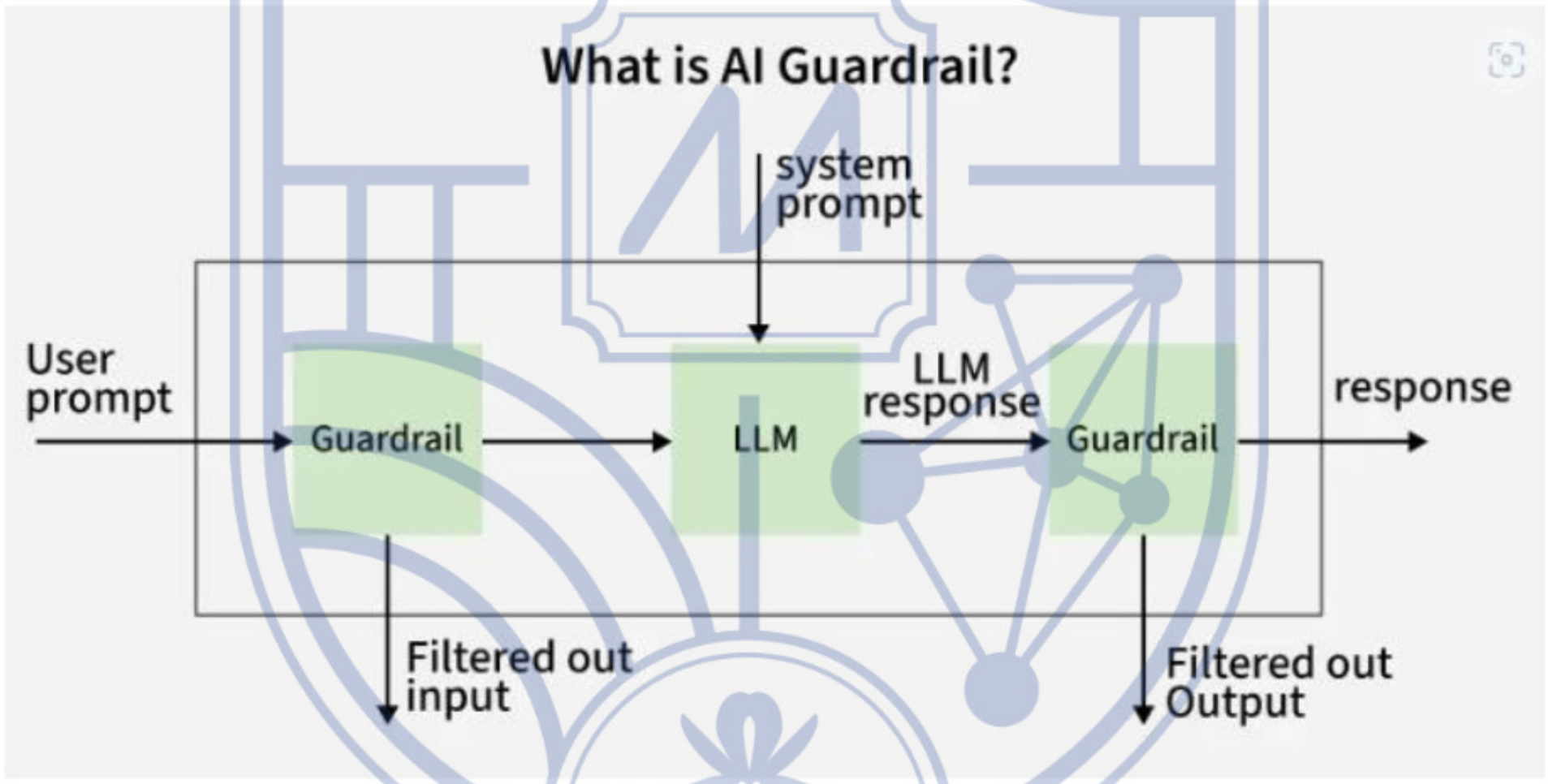
   Kemungkinan model AI menghasilkan respons yang tidak sesuai harapan, tidak konsisten, atau berpotensi menimbulkan risiko karena kurangnya mekanisme pengendalian [48].

5. Kerentanan Sumber Terbuka

   Risiko keamanan yang muncul akibat penggunaan model AI dan API open source yang tidak memiliki mekanisme perlindungan yang memadai, sehingga dapat dimanfaatkan untuk tujuan yang tidak semestinya [48].

6. *Input* Pengguna Tanpa Filter

   Input pengguna yang tidak difilter dapat mendorong sistem untuk menghasilkan respon yang tidak aman, tidak relevan atau bertentangan dengan tujuan aplikasi [48].



Gambar 2.6 AI Guardrails [50]

AI Guardrails digunakan untuk mengawasi *input* dan *output* sistem AI agar tetap aman dan sesuai aturan yang telah diterapkan. *Guardrails* bekerja dengan memeriksa apakah terdapat konten atau perilaku yang berpotensi menimbulkan masalah, sehingga tindakan pencegahan dapat dilakukan secara langsung jika diperlukan [49].

AI Guardrails dapat diterapkan melalui beberapa cara, yaitu:

1. Filter berbasis aturan, yaitu menyaring atau memblokir kata, frasa, atau pola tertentu yang dianggap tidak sesuai [49].

2. Pemantauan algoritmik, yaitu menggunakan model pembelajaran mesin untuk mendeteksi perilaku atau respons yang berisiko secara otomatis [49].
3. Integrasi kebijakan, yaitu memasukkan aturan organisasi atau regulasi ke dalam sistem AI agar selalu dipatuhi [49].
4. Pengawasan manusia, yaitu melibatkan manusia untuk meninjau kasus-kasus yang kompleks atau memiliki risiko tinggi [49].

2.8 *System Prompt*

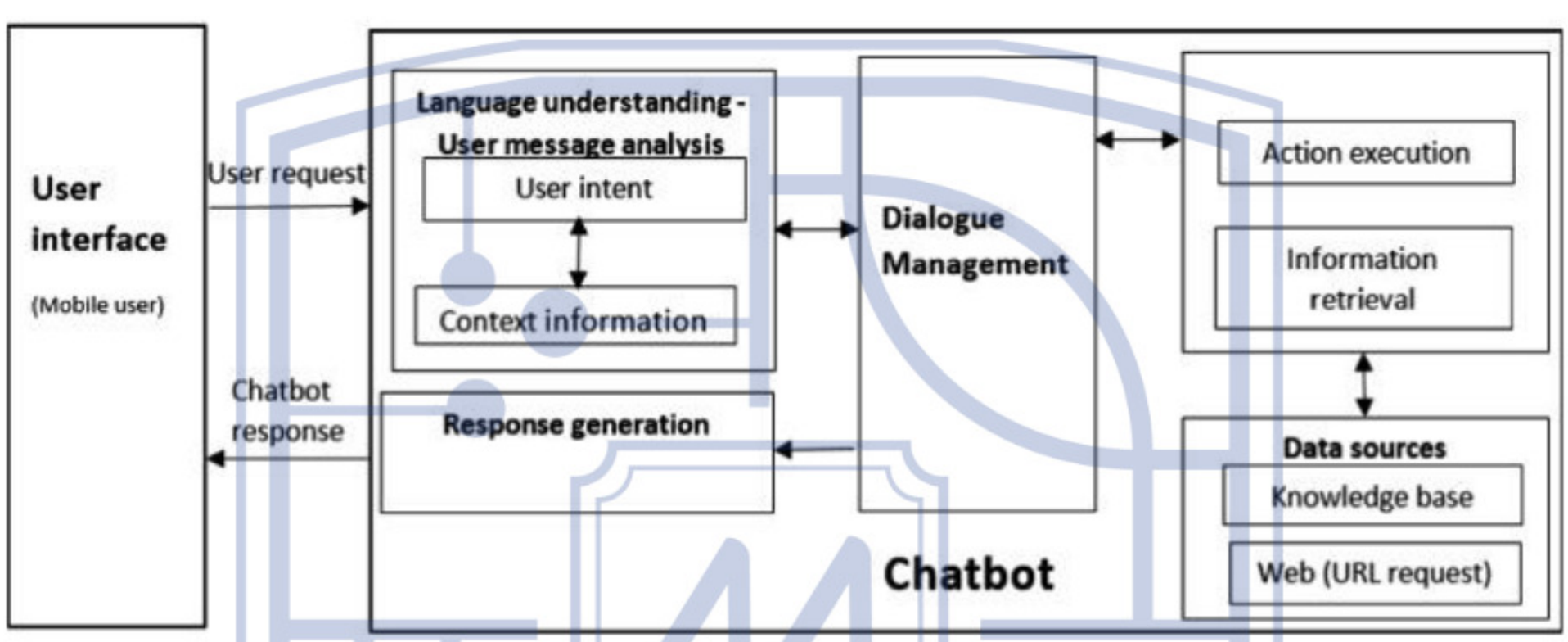
System Prompt adalah instruksi khusus yang diberikan kepada LLM untuk menentukan peran, perilaku, dan karakteristik respons model sebelum interaksi dengan pengguna dimulai. *System Prompt* berfungsi sebagai pedoman utama yang memastikan respons AI tetap konsisten, sesuai konteks, dan memenuhi tujuan aplikasi [50]. Berbeda dengan *user prompt* yang berisi pertanyaan pengguna, *system prompt* menentukan peran, konteks, dan batasan AI agar respons yang dihasilkan konsisten dan sesuai tujuan, sebagai contoh instruksi yang mengarahkan AI untuk berperan sebagai pengajar yang menjelaskan materi serta hanya berfokus pada tugas tersebut merupakan system prompt, sementara pengguna yang menanyakan penjelasan suatu materi adalah *user* prompt.

2.9 *Chatbot*

Chatbot adalah program atau aplikasi yang memungkinkan pengguna berkomunikasi melalui teks atau suara. Program ini dirancang untuk mensimulasikan percakapan manusia serta memberikan respons terhadap pertanyaan atau permintaan pengguna secara otomatis. *Chatbot* biasanya menggunakan aturan yang ditentukan sebelumnya untuk berkomunikasi dengan pengguna dan memberikan jawaban tertulis [51]. Dalam penerapannya, *chatbot* banyak digunakan di berbagai bidang, seperti layanan pelanggan, *e-commerce*, pendidikan, perbankan, hingga aplikasi kesehatan. *Chatbot* dapat membantu menjawab pertanyaan yang sering diajukan, memberikan panduan penggunaan layanan, memproses permintaan sederhana, serta meningkatkan efisiensi operasional suatu organisasi.

Dalam pengembangannya, *Chatbot* memanfaatkan beberapa konsep dan teknologi penting. Salah satunya adalah *Pattern Matching*, yaitu metode yang mencocokkan *input* pengguna dengan pola jawaban yang telah ditentukan sebelumnya. Selain itu, terdapat *Natural Language Processin*g (NLP), yaitu cabang kecerdasan buatan yang mempelajari bagaimana komputer memahami dan mengolah bahasa manusia. *NLP* memungkinkan *chatbot* memahami teks atau suara yang diberikan pengguna sehingga dapat menghasilkan

respons yang sesuai. Di dalam *NLP* terdapat *Natural Language Understanding* (NLU), yaitu proses untuk memahami maksud atau *intent* dari pengguna serta mengenali informasi penting yang disebut *entity*. Selain *intent* dan *entity*, *chatbot* juga menggunakan *context* untuk menyimpan konteks percakapan agar komunikasi dapat berlangsung secara berkelanjutan. Dengan adanya *context, chatbot* dapat memahami hubungan antara percakapan sebelumnya dan percakapan berikutnya [52].



Gambar 2.7 Proses Kerja *Chatbot* [52]

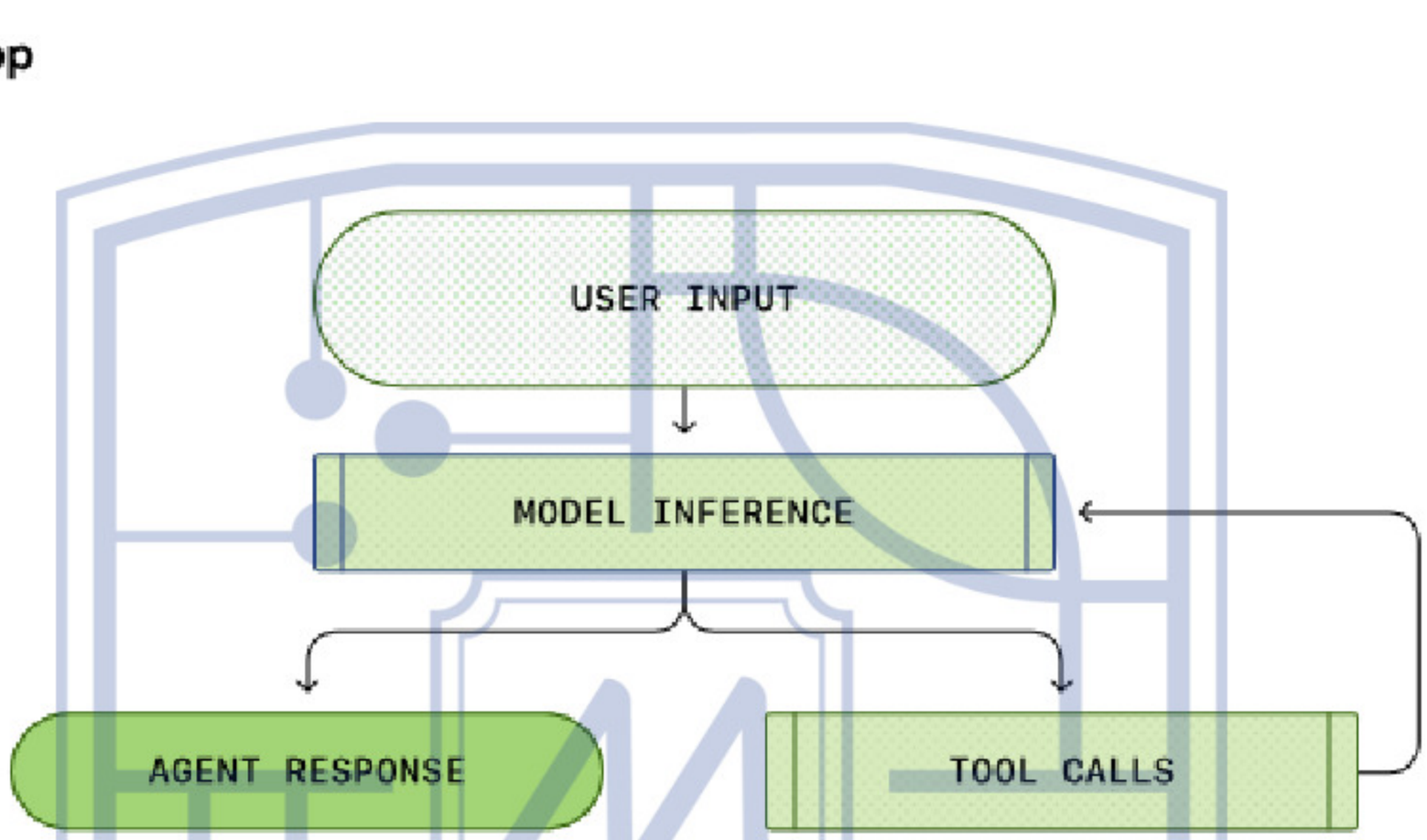
Secara umum, proses kerja *chatbot* adalah sebagai berikut:

1. Pengguna mengirimkan pesan melalui aplikasi atau sistem tertentu.
2. *Chatbot* menganalisis pesan tersebut menggunakan teknologi *NLU* untuk memahami maksud pengguna.
3. Setelah *intent* dan *entity* dikenali, *chatbot* menentukan tindakan yang akan dilakukan, seperti mengambil data dari database atau sumber eksternal melalui API.
4. Informasi yang diperoleh kemudian diolah oleh komponen *Response Generation* untuk menghasilkan jawaban yang alami dan mudah dipahami pengguna.
5. Selama proses percakapan berlangsung, komponen *Dialogue Management* akan menyimpan dan memperbarui konteks percakapan agar *chatbot* dapat memberikan respons yang lebih relevan. [52]

2.10 *Agent Loop*

Agent Loop adalah konsep dalam kecerdasan buatan di mana AI agent melakukan perulangan untuk mencapai tujuan. *Agent Loop* bertanggung jawab dalam mengatur alur interaksi antara pengguna, model, dan berbagai tool yang dipanggil model untuk

mengerjakan tugas perangkat lunak yang bermakna [53]. Berbeda dengan sistem AI yang hanya memberikan satu respons terhadap satu masukan, AI agent dengan mekanisme *agent loop* mampu terus mengamati kondisi, menganalisis hasil yang diperoleh, menentukan langkah berikutnya, menjalankan tindakan, kemudian mengevaluasi hasil tindakan tersebut. Siklus ini akan terus berlangsung hingga tujuan yang telah ditentukan berhasil dicapai atau kondisi penghentian (*stopping condition*) terpenuhi.



Gambar 2.8 Proses Agent Loop [66]

Gambar 2.8 merupakan ilustrasi sederhana dari alur kerja *agent loop*. Proses dimulai ketika agen menerima masukan dari pengguna. Masukan tersebut kemudian dirangkai bersama sekumpulan instruksi tekstual lain menjadi sebuah *prompt* yang akan dikirimkan kepada model. Tahap berikutnya adalah proses inferensi model, yaitu proses pengiriman *prompt* kepada model untuk menghasilkan tanggapan. Pada tahap ini, *prompt* diubah menjadi sekumpulan token masukan. Rangkaian token tersebut selanjutnya digunakan untuk melakukan *sampling* terhadap model sehingga dihasilkan rangkaian token keluar. *Token* keluar tersebut dikonversi menjadi teks kembali yang akan menjadi tanggapan model. Karena token dihasilkan secara bertahap, proses konversi dapat berlangsung bersamaan dengan proses inferensi. Hal inilah yang memungkinkan banyak aplikasi berbasis LLM menampilkan keluaran secara *streaming*. Pada penerapannya, proses inferensi umumnya dibungkus dalam sebuah API yang bekerja pada tingkat teks sehingga detail tokenisasi tidak perlu ditangani secara langsung oleh pengembang.

Hasil dari tahap inferensi dapat berupa dua kemungkinan. Pertama, model menghasilkan tanggapan akhir atas permintaan pengguna. Kedua, model mengajukan

permintaan pemanggilan *tool* yang harus dijalankan oleh agen, misalnya menjalankan perintah tertentu dan melaporkan hasilnya. Pada kemungkinan kedua, agen mengeksekusi pemanggilan tool tersebut, lalu menambahkan hasil eksekusinya ke dalam *prompt* sebelumnya. Gabungan tersebut menjadi masukan baru untuk melakukan kueri ulang terhadap model, sehingga model dapat mempertimbangkan informasi tambahan tersebut dalam menghasilkan tanggapan berikutnya.

Siklus ini berlangsung berulang hingga model berhenti mengeluarkan permintaan pemanggilan tool dan menghasilkan pesan yang ditujukan kepada pengguna. Pesan tersebut umumnya berisi jawaban langsung atas permintaan awal pengguna, namun dapat pula berupa pertanyaan lanjutan untuk memperjelas kebutuhan pengguna [53].

2.11 *Tool Calling*

Tool Calling merujuk pada kemampuan model AI (*Artificial Intelligence*) untuk berinteraksi dengan perangkat eksternal, application programming interface (API), maupun sistem lain guna memperluas fungsi yang dimilikinya. Pada penerapannya, *tool calling* memungkinkan sistem otonom untuk menyelesaikan tugas kompleks dengan mengakses dan memanfaatkan sumber daya eksternal secara dinamis. Dengan kemampuan ini, *Large Language Model* (LLM) tidak lagi terbatas pada menjawab pertanyaan, melainkan dapat mengotomatiskan alur kerja, berinteraksi dengan basis data, menyelesaikan persoalan secara bertahap, serta mengambil keputusan secara *real-time*. Pergeseran ini mengubah peran LLM dari sekadar asisten pasif menjadi agen digital yang mampu bertindak secara proaktif dalam menyelesaikan tugas kompleks [54]. Dengan kemampuannya, RAG dapat berfokus pada penyediaan konteks pengetahuan, sementara *tool calling* dapat berfokus kemampuan bertindak dalam memutuskan kapan perlu melakukan pencarian dokumen.

Berikut merupakan tahapan dari proses *tool calling*:

1. Pengguna mengajukan pertanyaan kepada model, model memanfaatkan kemampuan pemahaman bahasa alami untuk mengidentifikasi apakah jawaban tersebut memerlukan data dari luar yang tidak tersedia di dalam pengetahuan statis hasil pelatihannya. Setiap permintaan pemanggilan *tool* secara otomatis diberi *tool call ID* yang bersifat unik, yang berfungsi sebagai petanda untuk menghubungkan permintaan dengan hasil yang dikembalikan [54].
2. Model menentukan tool yang paling sesuai dengan kebutuhan. Pemilihan tools ini bertujuan informasi yang diperoleh akurat dan relevan. Setiap tool dilengkapi metadata dan informasi terstruktur berupa nama tool (atau nama fungsi) yang unik, deskripsi,

parameter, serta tipe masukan dan keluaran yang dibutuhkan. Berdasarkan metadata tersebut, model melakukan *tool choice*, yaitu pemilihan tool dari sekumpulan tool yang tersedia. Format *prompt* terstruktur (*template*) digunakan untuk mengarahkan model dalam menentukan tool yang dipanggil beserta argumen (*args*) yang diberikan, sehingga interaksi dengan API menjadi lebih terkendali dan terstruktur [54].

3. Model kemudian menyusun permintaan terstruktur yang dapat dipahami oleh tool atau API terkait. Setiap tool memiliki fungsi tertentu yang mendefinisikan perilakunya, dan fungsi tersebut mengacu pada dokumentasi API yang menjelaskan alamat *endpoint*, metode permintaan, serta format tanggapan. Sebagian layanan eksternal mensyaratkan *API key* sebagai identitas unik untuk memperoleh izin akses. Setelah tool dipilih dan parameter ditetapkan, permintaan dikirimkan melalui protokol HTTP menuju server eksternal untuk memperoleh data yang dibutuhkan [54].
4. Tool eksternal mengembalikan data yang selanjutnya harus diuraikan oleh model. API dapat mengembalikan objek berformat JSON. Model kemudian menyaring dan menyusun data tersebut menjadi tanggapan yang bermakna bagi pengguna [54].
5. Model menyampaikan hasil pemrosesan kepada pengguna dalam bentuk yang mudah dipahami [54].
6. Apabila pengguna meminta rincian tambahan atau melakukan perubahan permintaan, model dapat mengulangi keseluruhan proses dengan permintaan yang telah disesuaikan, sehingga tanggapan yang dihasilkan terus disempurnakan sesuai kebutuhan pengguna [54].

2.12 *Vector Database*

Vector Database adalah sistem basis data yang dirancang untuk menyimpan dan melakukan pencarian data dalam bentuk representasi numerik atau disebut dengan *embedding vector*. Representasi ini memungkinkan sistem melakukan pencarian berdasarkan makna, sehingga informasi yang diperoleh relevan dengan kebutuhan pengguna [55]. Selain itu, *Vector Database* umumnya dilengkapi dengan algoritma pencarian kemiripan (*similarity search*) yang mampu menemukan data dengan tingkat kemiripan tertinggi terhadap suatu kueri. Teknologi ini banyak digunakan pada aplikasi berbasis kecerdasan buatan (*Artificial Intelligence*), seperti sistem rekomendasi, pencarian semantik dan *chatbot*.

Untuk mendukung proses pengambilan informasi semantik yang cepat dan efisien, basis data vektor memiliki tiga fungsi utama, yaitu sebagai berikut:

1. Penyimpanan Vektor

   Basis data vektor menyimpan data dalam bentuk *embedding vector*, yaitu representasi numerik yang dihasilkan dari data seperti teks, gambar, atau audio. Setiap vektor memiliki jumlah dimensi tertentu dan biasanya disertai metadata, seperti judul, sumber, tanggal, atau kategori. Penyimpanan *embedding* yang telah dihitung sebelumnya memungkinkan sistem melakukan pencarian tanpa perlu membuat *embedding* ulang setiap kali kueri diajukan, sehingga meningkatkan kecepatan proses pencarian [56].

2. Pengindeksan Vektor

   Untuk mempercepat pencarian pada ruang vektor berdimensi tinggi, basis data vektor membangun indeks khusus pada data yang tersimpan. Pengindeksan memungkinkan sistem menemukan vektor yang paling mirip tanpa harus membandingkan seluruh data yang ada. Teknik yang umum digunakan adalah *Approximate Nearest Neighbor (ANN)*, seperti *Hierarchical Navigable Small World (HNSW)* dan *Locality Sensitive Hashing (LSH)*. Dengan adanya indeks, proses pencarian menjadi lebih cepat dan tetap efisien meskipun jumlah data sangat besar [56].

3. Pencarian Vektor

   Pencarian vektor digunakan untuk menemukan data yang memiliki kemiripan semantik dengan kueri pengguna. Ketika pengguna mengajukan pertanyaan, sistem akan mengubah kueri tersebut menjadi *query vector* dan membandingkannya dengan vektor yang tersimpan dalam basis data. Tingkat kemiripan dihitung menggunakan metrik tertentu, seperti *Cosine Similarity*, *Dot Product*, atau *Euclidean Distance*. Hasil dengan tingkat kemiripan tertinggi kemudian dikembalikan sebagai informasi yang paling relevan. Mekanisme ini menjadi dasar bagi berbagai aplikasi AI, termasuk *semantic search*, *chatbot* dan sistem rekomendasi [56].

2.13 *Cosine Similarity*

Cosine similarity adalah metode pengukuran tingkat kedekatan antara dua vektor pada suatu ruang berdimensi *n*, yang dihitung berdasarkan nilai kosinus dari sudut yang terbentuk di antara keduanya. Semakin kecil sudut yang terbentuk, semakin tinggi nilai kosinusnya, sehingga kedua vektor dinilai semakin serupa. Rentang nilai yang dihasilkan berada antara 0 hingga 1, dengan nilai 1 menandakan kedua vektor memiliki arah identik dan nilai 0 menandakan keduanya saling tegak lurus atau tidak berkaitan sama sekali [57].

Perhitungan *cosine similarity* dinyatakan melalui persamaan berikut:

$$\cos \alpha = \frac{Q \cdot D}{|Q||D|} = \frac{\sum_{i=1}^{n}(q_i \times d_i)}{\sqrt{\sum_{i=1}^{n}(q_i)^2} \times \sqrt{\sum_{i=1}^{n}(d_i)^2}} \quad (2.1)$$

Keterangan:

Q = Vektor query yang dibandingkan kemiripannya.

D = Vektor dokumen yang dibandingkan kemiripannya.

Q · D = dot product antara vektor Q dan D.

|Q| = panjang vektor Q.

|D| = panjang vektor D.

|Q||D| = hasil kali panjang vektor Q dan D.

q_i = nilai vektor query pada dimensi ke-i.

d_i = nilai vektor dokumen pada dimensi ke-i.

n = jumlah dimensi vektor.

2.14 Vercel AI SDK

Vercel AI SDK adalah *open source library* berbasis *TypeScript* yang dirancang untuk membantu developer dalam membangun aplikasi berbasis AI [58]. Melalui Vercel AI SDK, developer tidak perlu membangun mekanisme komunikasi dengan model AI dari awal karena berbagai fungsi penting, seperti pengiriman *prompt*, pengelolaan respons, *streaming output*, dan integrasi dengan framework *web* modern telah disediakan secara bawaan. Dengan demikian, developer dapat lebih fokus pada penulisan code yang berkaitan dengan logika bisnis tanpa menghabiskan waktu pada hal-hal teknis [59]. Selain itu, Vercel AI SDK juga berguna untuk menyederhanakan perbedaan antar penyedia model, menghilangkan kode yang berulang saat membangun *chatbot*, serta memungkinkan *developer* membuat fitur AI yang tidak hanya menampilkan teks, tetapi juga komponen yang lebih kaya dan interaktif [58].

Vercel AI SDK memiliki 2 pustaka utama, yaitu AI SDK Core dan AI SDK UI. AI SDK Core adalah API yang digunakan untuk menghasilkan teks, objek terstruktur, panggilan alat, dan membangun agen dengan *Large Language Model* (LLM). Sementara itu, AI SDK UI adalah serangkaian hook yang tidak bergantung pada framework untuk membangun antarmuka pengguna (UI) dan generatif dengan cepat [60]. AI SDK Core biasa digunakan untuk *generate text*, *generate* struktur data, dan *tool usage* [61]. Sementara itu, AI SDK UI digunakan untuk membantu membangun chat yang interaktif, completion dan assistant application dengan mudah [62]. Sebagai contoh AI SDK Core bertugas untuk menjawab pertanyaan pengguna atau menghasilkan data dalam bentuk JSON, sementara AI SDK UI

menampilkan percakapan jawaban tersebut di layar chat secara real-time dan menerima input dari pengguna. Dengan demikian, AI SDK Core berfokus pada logika dan pemrosesan AI, sedangkan AI SDK UI berfokus pada penyajian dan interaksi pengguna.

2.15 pgvector

pgvector adalah ekstensi untuk PostgreSQL yang menyederhanakan pekerjaan dengan vektor, sehingga dapat disimpan, menelusuri, dan mengindeksnya langsung di database relasional [63]. pgvector berfungsi untuk pencarian berbasis kemiripan makna (*semantic search*), bukan sekedar kemiripan kata. Sebagai contoh, kata *"bahagia"* dan *"senang"* memiliki bentuk kata yang berbeda, namun memiliki makna yang serupa. Dalam representasi vektor, kedua kata tersebut akan memiliki posisi yang berdekatan atau nilai kemiripan yang tinggi di ruang vektor karena merepresentasi konsep yang hampir mirip. Dengan pendekatan ini, pgvector mampu menangkap kesamaan antar kata melalui representasi *vektor*, sehingga pencarian menjadi lebih relevan dibandingkan pencarian berbasis konvensional.

Alur kerja dari pgvector adalah sebagai berikut :

1. Data mentah diproses menggunakan model *embedding* untuk mengubah data menjadi *vektor*.
2. *Vektor* hasil *embedding* kemudian disimpan di database PostgreSQL.
3. Pengguna melakukan pencarian.
4. *Input* pengguna diubah menjadi *vektor* dengan menggunakan model *embedding* yang sama.
5. pgvector menghitung kemiripan antara *vektor* pencarian dengan *vektor* yang disimpan di database.

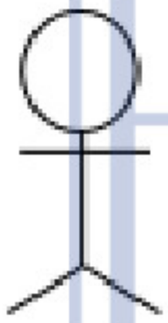
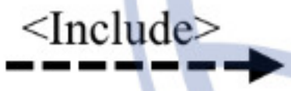
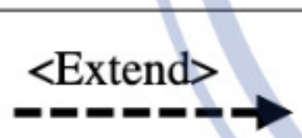
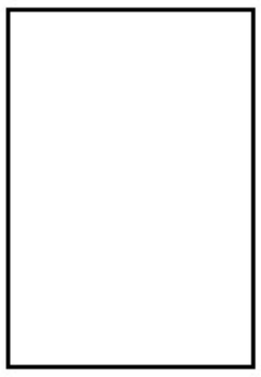
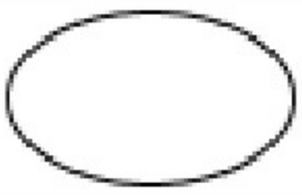
2.16 *Unified Modeling Language* (UML)

Unified Modeling Language merupakan sebuah bahasa yang bertujuan untuk memberikan gambaran dan spesifikasi dalam pembangunan dan dokumentasi dari sebuah pengembangan berorientasi objek (*object oriented*) dalam bentuk gambar maupun grafik [64]. UML digunakan untuk menggambarkan bagaimana *user* berinteraksi dengan sistem serta untuk menjelaskan batasan-batasan yang dimiliki oleh sistem tersebut [65]. Dengan demikian, UML tidak hanya membantu dalam tahap perancangan, tetapi juga membantu mempermudah komunikasi antar pemangku kepentingan. Di dalam UML, terdapat berbagai jenis diagram yang memiliki fungsi dan tujuan yang berbeda, yaitu sebagai berikut:

2.16.1 *Use Case Diagram*

Use Case Diagram mendeskripsikan bagaimana sistem merespon berbagai kondisi atau permintaan dari pemangku kepentingan untuk mencapai satu tujuan [66]. *Diagram* ini menggambarkan interaksi antara aktor dan sistem, serta menunjukkan fungsi-fungsi utama yang disediakan oleh sistem guna memenuhi kebutuhan tersebut [67]. Dengan demikian, *Use Case Diagram* membantu menjelaskan bagaimana sistem bekerja dari sudut pandang pengguna dan bagaimana setiap permintaan diproses untuk menghasilkan keluaran yang sesuai dengan tujuan yang diharapkan. Berikut adalah simbol dari *use case diagram*.

Tabel 2.1 *Use Case Diagram* [68]

No	Gambar	Nama	Keterangan
1		Actor	Peran yang pengguna mainkan ketika berinteraksi dengan *Use case*
2		Dependency	Hubungan yang terjadi antara dua elemen, di mana satu elemen bergantung pada elemen lainnya
3		Generalization	Hubungan di mana objek anak berbagi perilaku dan struktur data dari objek yang ada diatasnya
4		Include	Hubungan di mana suatu use case selalu menyertakan *use case* lain.
5		Extend	Hubungan di mana suatu *use case* menambahkan perilaku ke *use case* lain, tetapi hanya dalam kondisi tertentu.
6		Association	Menghubungkan satu objek dengan objek lainnya
7		System	Menspesifikasikan paket yang menampilkan sistem secara terbatas
8		Use Case	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur

2.17 *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) adalah tipe dari flowchart yang menggambarkan bagaimana suatu entitas seperti orang, benda atau konsep saling berhubungan satu sama lain di dalam suatu sistem. ERD menggunakan simbol seperti persegi panjang, belah ketupat, oval serta garis penghubung untuk menggambarkan keterkaitan entitas, hubungan, dan atributnya [69]. Dengan menggunakan ERD, proses perancangan basis data menjadi lebih terstruktur dan sistematis, sehingga dapat meminimalkan kesalahan dalam perancangan tabel dan relasi antar tabel. Selain itu, ERD juga mempermudah komunikasi antara pengembang sistem dan pihak terkait dalam memahami kebutuhan data suatu sistem secara menyeluruh.

Dalam ERD, terdapat 3 konsep utama yang menjadi dasar dalam pemodelan data, yaitu:

1. Entitas

Sebuah entitas dapat berupa orang, tempat, objek, atau kejadian yang dianggap penting oleh organisasi. Setiap entitas memiliki beberapa atribut yang mendeskripsikan objek. Entitas terbagi atas 2 macam, yaitu *strong entity* dan *weak entity*. *Strong entity* tidak bergantung pada entitas lain karena memiliki *primary key* yang mampu mengidentifikasi setiap data secara unik, sedangkan *weak entity* bergantung pada entitas lain karena tidak memiliki *primary key* dan keberadaannya bergantung pada *strong entity*. Contoh dari *strong entity* adalah entitas mahasiswa yang memiliki *primary key* berupa NIM, sedangkan contoh dari *weak entity* adalah entitas detail pembayaran, yang memiliki keterkaitannya dengan entitas pembayaran.

2. Atribut

Atribut berfungsi untuk mendeskripsikan karakteristik tertentu yang ada pada entitas yang disimpan dalam basis data. Berdasarkan sifat dan bentuknya, atribut dapat dibedakan menjadi beberapa jenis, yaitu *simple attribute* yang tidak dapat diuraikan lagi, *composite attribute* yang terdiri dari beberapa bagian, *single-valued attribute* yang memiliki satu nilai, *derived atribute* yang nilainya diperoleh dari atribut lain, serta *key attribute* yang digunakan sebagai penanda unik suatu entitas. Salah satu jenis *key attribute* yang paling penting adalah *primary key*, yaitu atribut yang berfungsi sebagai identitas unik untuk membedakan setiap data atau record dalam suatu entitas. *Primary key* memastikan bahwa setiap data dapat dikenali secara jelas dan tidak terjadi duplikasi dalam sistem basis data.

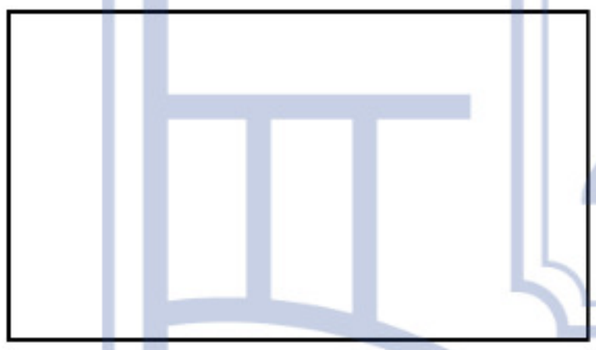
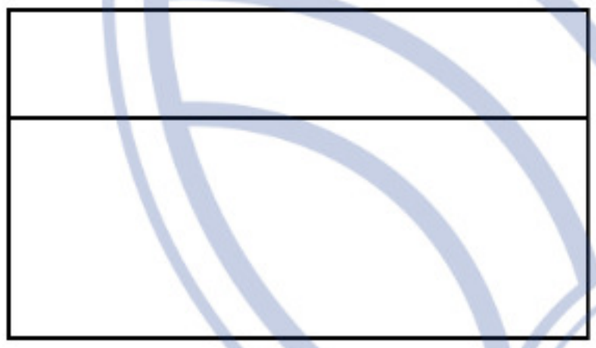
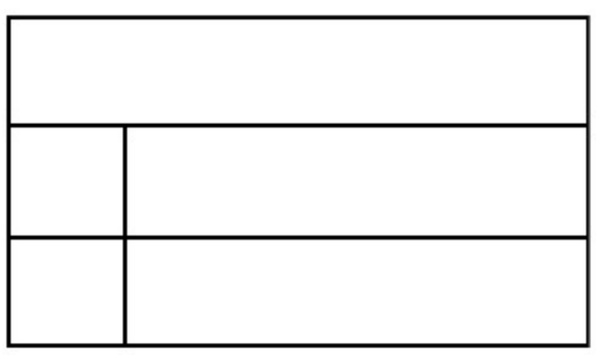
3. Relasi

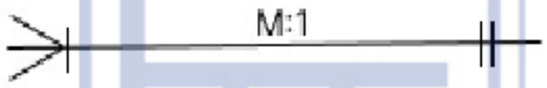
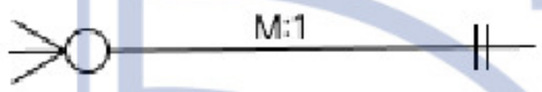
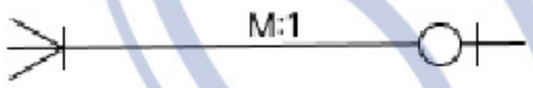
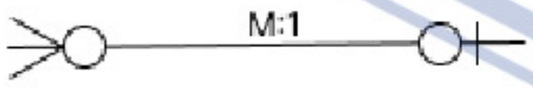
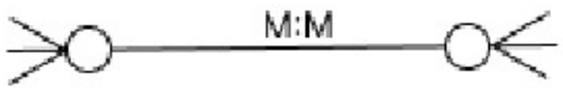
Relasi merupakan hubungan yang terjadi antara dua atau lebih entitas yang saling berkaitan di dalam suatu sistem. Relasi digunakan untuk menunjukkan bagaimana suatu

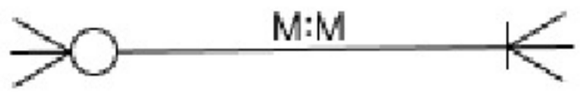
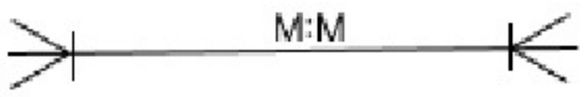
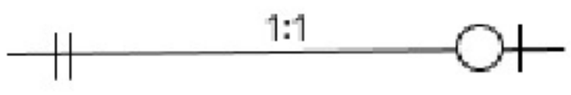
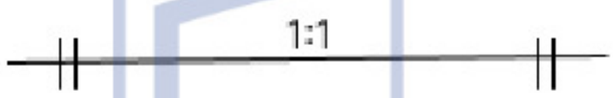
entitas berhubungan satu sama lain. Relasi dibedakan atas 3 jenis, yaitu *unary* (hubungan dengan dirinya sendiri), *binary* (hubungan antara dua entitas), dan *ternary* (hubungan yang melibatkan tiga entitas) [70].

Pada *Entity Relationship Diagram* (ERD), terdapat berbagai gaya notasi yang digunakan untuk menggambarkan kardinalitas atau hubungan antar entitas. Berbagai notasi telah dikembangkan untuk memudahkan perancangan dan pemahaman struktur basis data, seperti *Chen Notation*, *Crow's Foot Notation*, *Bachman Notation*, dan *Information Engineering Notation* [71]. Masing-masing notasi memiliki karakteristik visual yang berbeda dalam merepresentasikan hubungan *one-to-one* (1:1), *one-to-many* (1:N), maupun *many-to-many* (M:N). Pada penelitian ini, *Crow's Foot Notation* digunakan sebagai dasar perancangan basis data sistem informasi. Berikut penjelasan simbol-simbol pada *Crow's Foot Notation*.

Tabel 2.2 Crow Foot Notation [71]

Bentuk	Nama Bentuk
	Entitas yang tidak memiliki atribut
	Entitas yang memiliki atribut
	Entitas yang memiliki atribut dengan kolom
	Entitas yang memiliki atribut dengan kolom dan jumlah baris

Relationship	
	Zero or One
	One or More
	One and only One
	Zero or More
Many to One	
	Satu ke banyak notasi di satu sisi hubungan dan satu dan hanya satu ke sisi lain
	Nol ke banyak notasi di satu sisi dan satu hanya satu di sisi lain
	Satu ke banyak notasi di satu sisi hubungan dan nol atau satu notasi di sisi lain
	Nol ke banyak notasi ke satu sisi hubungan dan nol ke satu notasi di sisi lain
Many to Many	
	Nol hingga banyak pada kedua sisi hubungan

	Nol hingga banyak di satu sisi dan satu ke banyak di sisi lain
	Satu ke banyak di kedua sisi hubungan
	Satu dan hanya satu notasi di satu sisi hubungan dan nol atau satu di sisi lain
	Satu dan hanya satu notasi di kedua sisi hubungan

2.18 *Flowchart*

Flowchart merupakan representasi grafis yang digunakan untuk menggambarkan alur kerja, langkah-langkah, serta urutan suatu proses atau prosedur dalam sebuah sistem maupun program [72]. *Flowchart* disusun menggunakan simbol-simbol standar yang saling dihubungkan oleh garis alir (flow line) untuk menunjukkan arah dan urutan eksekusi proses. Dengan adanya *flowchart*, proses bisnis maupun logika program dapat divisualisasikan secara sistematis sehingga lebih mudah dipahami, dianalisis, dan dikembangkan.

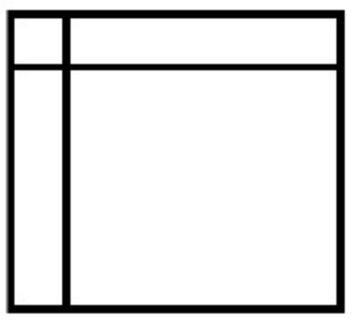
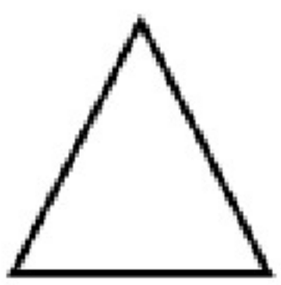
Adapun petunjuk pembuatan *Flowchart* adalah sebagai berikut:

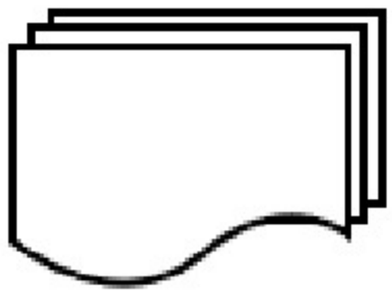
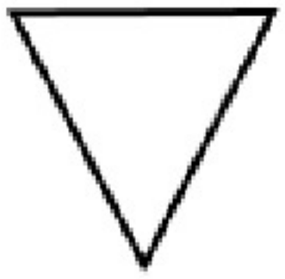
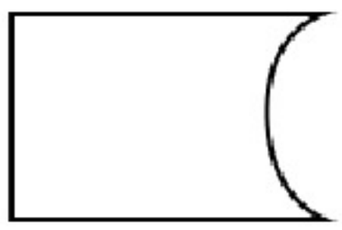
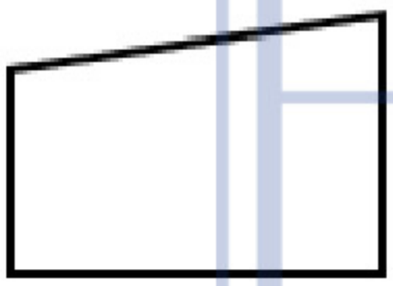
1. *Flowchart* sebaiknya disusun dengan arah alur yang konsisten, yaitu dimulai dari bagian atas menuju bawah serta dari kiri ke kanan agar mudah diikuti.Kegiatan yang digambarkan harus dapat dimengerti pengguna [72].
2. Setiap aktivitas atau proses yang digambarkan harus disajikan secara jelas sehingga dapat dipahami oleh seluruh pengguna atau pembaca [72].
3. *Flowchart* harus memiliki penanda yang jelas mengenai titik awal (start) dan titik akhir (end) dari suatu proses [72].
4. Setiap tahapan proses sebaiknya dijelaskan menggunakan kata kerja yang menggambarkan aktivitas yang sedang dilakukan [72].
5. Urutan langkah dalam *flowchart* harus disusun secara logis dan sistematis sesuai dengan alur proses yang sebenarnya [72].
6. Ruang lingkup serta aliran proses yang digambarkan perlu dianalisis dan ditelusuri secara menyeluruh agar tidak terdapat tahapan yang terlewat [72].

7. Dalam penyusunan *flowchart*, disarankan menggunakan simbol-simbol standar atau simbol baku agar *diagram* mudah dipahami dan sesuai dengan kaidah yang berlaku [72].

Simbol-simbol *Flowchart* yang biasanya dipakai adalah sebagai berikut:

Tabel 2.3 Simbol *Flowchart* [72]

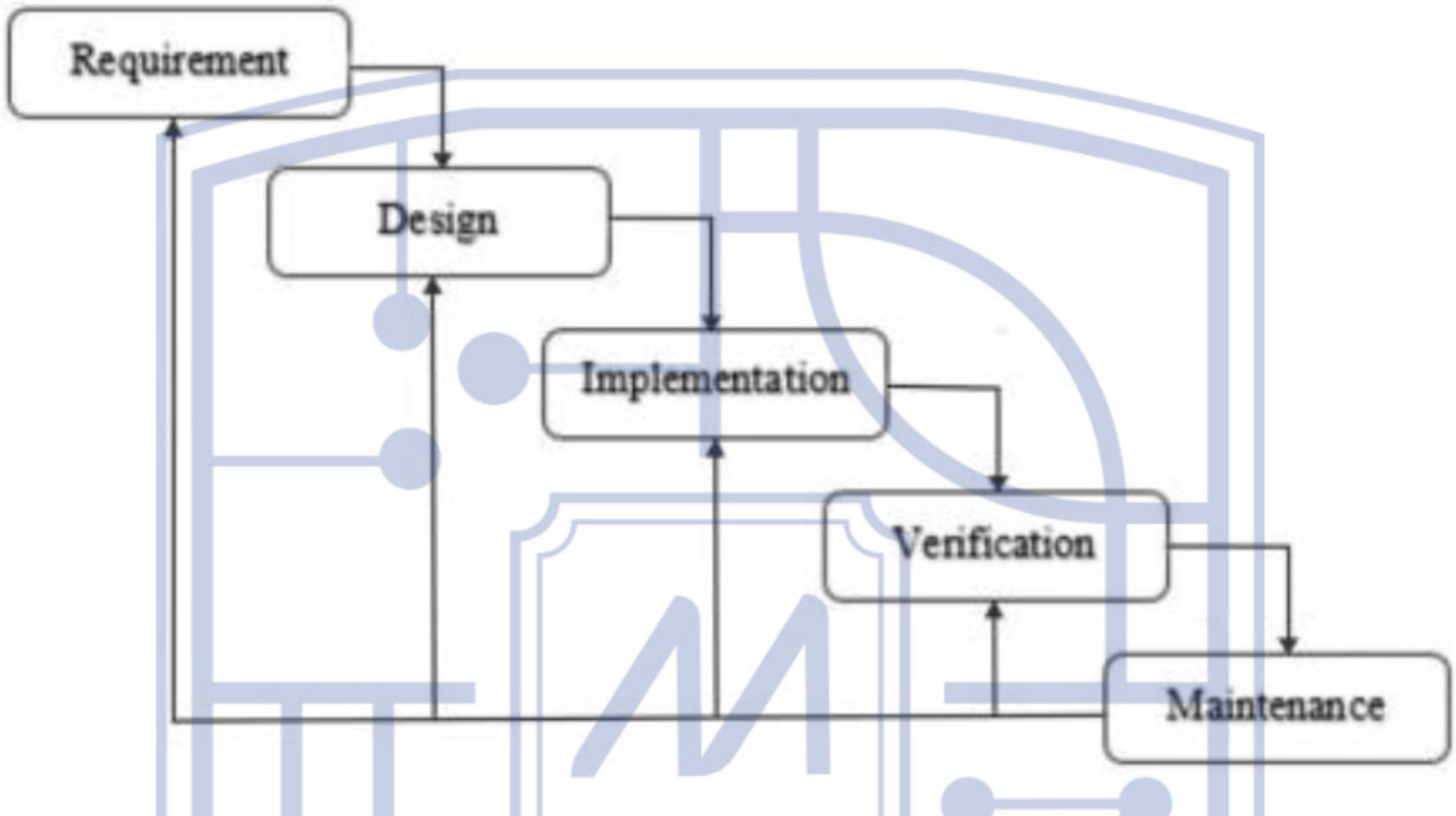
Gambar	Fungsi	Gambar	Fungsi
	Proses		Card
	Proses pilihan		Punched Tape
	Keputusan		Summing Junction
	Input Data dan Output Informasi		Or
	Predefine Proses		Collate
	Internal Storage		Sort
	Dokumen		Extract

	Multi Dokumen		Merge
	Terminator (mulai dari Akhir)		Storage Data
	Preparasi		Delay
	Manual Input		Seqeuntial Access Storage
	Manual Operasi		Magnetic Disk
	Penghubung		Direct Access Storage
	*Off Page* Penghubung		Display

2.19 Metodologi *Waterfall*

Metodologi *waterfall* merupakan model pengembangan perangkat lunak yang harus dilakukan secara sistematis dan berurutan. Metodologi ini mengalir dari atas sampai ke bawah tanpa kembali ke tahap sebelumnya sehingga disebut *waterfall*. Setiap tahap dalam metodologi *waterfall* bergantung pada penyelesaian tahap sebelumnya yang berarti bahwa tahap selanjutnya tidak dapat dimulai sampai tahap sebelumnya selesai dan tervalidasi.

Meskipun demikian, metode ini memiliki kekurangan, terutama dari segi fleksibilitas dan penanganan kesalahan. Metode ini dianggap kurang fleksibel karena setiap tahap bergantung pada tahap sebelumnya yang membuat perubahan menjadi sangat mahal dan sulit untuk diterapkan. Selain itu, kesalahan yang terdeteksi setelah tahap awal pengembangan dapat menjadi mahal untuk diperbaiki karena perlu kembali ke tahap yang relevan [73].



Gambar 2.9 Tahapan Metodologi Waterfall

Tahapan metodologi *waterfall* meliputi :

1. Analisis Kebutuhan *(Requirement Analysis)*

   Pada tahap ini, developer memahami perangkat lunak yang diharapkan oleh pengguna dan batasan perangkat lunak tersebut. Informasi dikumpulkan dari wawancara, diskusi atau survei langsung.

2. Perancangan Sistem (*System Design*)

   Pada tahap ini berfokus pada perancangan desain arsitektur sistem, perancangan basis data, serta desain antarmuka pengguna. Developer perlu menentukan perangkat keras dan sistem persyaratan serta mendefinisikan arsitektur sistem secara keseluruhan.

3. Implementasi *(Implementation)*

   Pada tahap ini, sistem pertama kali dikembangkan di program kecil yang disebut *unit*. Setiap unit dikembangkan dan diuji untuk fungsionalitas yang disebut sebagai *unit testing.*

4. Pengujian *(Verification)*

Pada tahap ini, sistem dilakukan verifikasi dan pengujian untuk dapat menentukan apakah sistem memenuhi persyaratan sistem dan memastikan bahwa seluruh fungsi berjalan sesuai yang diharapkan.

5. Pemeliharaan *(Maintenance)*

Merupakan tahap terakhir dari *waterfall*. Perangkat lunak yang sudah jadi dijalankan serta dilakukan pemeliharaan yang mencakup perbaikan kesalahan, pembaruan sistem, serta penyesuaian kebutuhan baru [74].

2.20 *Black box Testing*

Black box Testing merupakan salah satu teknik pengujian perangkat lunak yang berfokus pada pemeriksaan fungsi dan perilaku sistem berdasarkan kebutuhan dan spesifikasi yang telah ditetapkan tanpa perlu memperhatikan implementasi kode [75]. Pada metode ini, penguji hanya mengevaluasi hubungan antara masukan (*input*) yang diberikan dengan keluaran (*output*) yang dihasilkan untuk memastikan bahwa setiap fungsi sistem bekerja sesuai dengan yang diharapkan. Pengujian dilakukan dari sudut pandang pengguna (user), sehingga lebih menekankan pada validasi terhadap kebutuhan fungsional sistem dibandingkan mekanisme pemrosesan di dalamnya.

Tujuan dilakukannya *Black box Testing* adalah untuk mengidentifikasi berbagai jenis kesalahan yang mungkin terjadi pada perangkat lunak, meliputi:

1. Fungsi yang tidak berjalan sebagaimana mestinya atau fungsi yang belum diimplementasikan [76]
2. Kesalahan pada antarmuka (*interface*) antara pengguna dengan sistem maupun antar komponen sistem [76].
3. Kesalahan yang berkaitan dengan struktur data atau mekanisme akses ke basis data eksternal [76].
4. Permasalahan yang memengaruhi kinerja (*performance*) sistem [76].
5. Kesalahan yang terjadi selama proses inisialisasi maupun penghentian (terminasi) sistem [76].
6. Ketidaksesuaian fungsi sistem terhadap kebutuhan atau spesifikasi fungsional yang telah ditetapkan [76].
7. Sensitivitas sistem terhadap nilai masukan tertentu yang berpotensi menyebabkan perilaku yang tidak diharapkan [76].
8. Kesalahan yang muncul pada batasan atau rentang nilai data yang diterima oleh sistem [76].

Black box testing memiliki beberapa metode pengujian, seperti *equivalent partitioning*, *boundary value analysis*, *cause effect graph*, *comparison testing, random data selection*, *feature test*, *all-pair testing*, *fuzzing*, *orthogonal array testing*, *sample testing, use case testing,* dan lain-lain. Di antara berbagai metode yang terdapat di *black box testing*, *use case testing* merupakan salah satu teknik pengujian yang berfokus pada validasi fungsionalitas sistem berdasarkan skenario pengguna yang telah dirancang [77]. *Use Case Testing* memastikan suatu sistem berfungsi sesuai dengan skenario penggunaan yang telah dirancang. Berikut tahapan dalam penerapan *Use Case Testing*:

1. Identifikasi Use Case

   Setiap fitur utama diidentifikasi dan dijelaskan dalam bentuk *use case* yang menggambarkan interaksi pengguna dengan sistem [77].

2. Penyusunan Skenario Uji

   Untuk setiap *use case*, disusun skenario pengujian yang mencakup jalur utama (kondisi normal) dan jalur alternatif (kondisi tidak normal, seperti kesalahan input atau kegagalan sistem) [77].

3. Pelaksanaan Pengujian

   Skenario dijalankan menggunakan data nyata untuk memastikan sistem berfungsi sesuai kebutuhan pengguna [77].

4. Analisis Hasil

   Hasil pengujian dibandingkan dengan hasil yang diharapkan untuk mengidentifikasi ketidaksesuaian pada alur sistem, termasuk penanganan kondisi alternatif [77].

2.21 Tingkat Akurasi

Tingkat akurasi merupakan ukuran yang menyatakan seberapa besar proporsi hasil yang benar terhadap keseluruhan data yang diuji. Perhitungannya dilakukan dengan membandingkan jumlah prediksi yang benar terhadap jumlah sampel, kemudian dikalikan seratus persen [78]. Ukuran ini digunakan untuk menilai sejauh mana keluaran suatu sistem sesuai dengan data acuan yang telah ditetapkan sebelumnya. Perhitungan tingkat akurasi dinyatakan pada Persamaan 2.2.

$$Tingkat\ akurasi = \frac{jumlah\ prediksi\ benar}{jumlah\ sampel} \times 100\% \quad (2.2)$$

Keterangan:

jumlah prediksi benar = banyaknya keluaran sistem yang sesuai dengan data acuan

jumlah sampel = banyaknya keseluruhan data yang diuji