BAB II

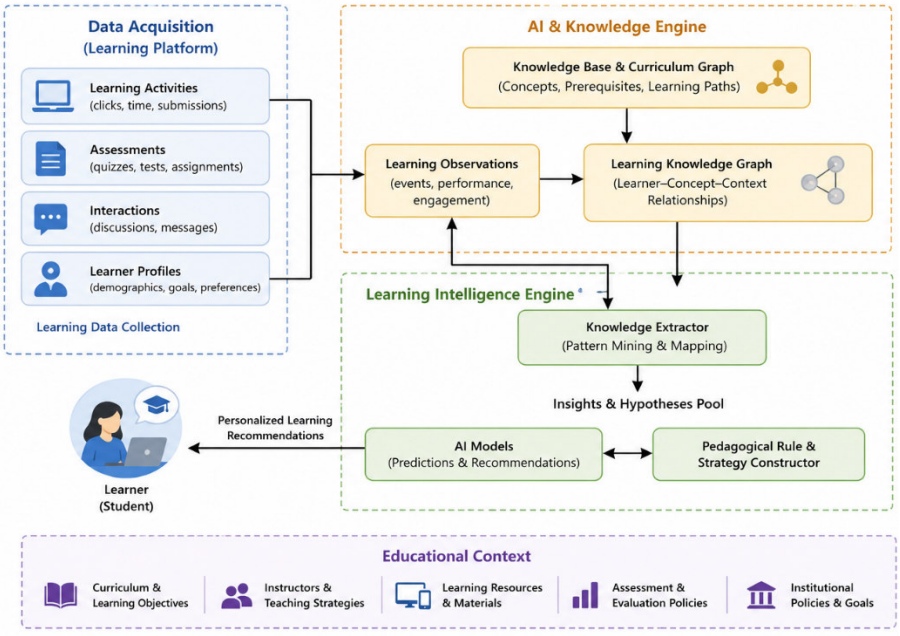
KAJIAN PUSTAKA

A. Kajian Teoritis

1. Sistem

a. *Neuro-Symbolic AI* (NSAI)

Neuro-symbolic AI didefinisikan sebagai integrasi metode jaringan saraf dengan pendekatan berbasis pengetahuan simbolik, yang bertujuan mengombinasikan kekuatan *representasional neural* dengan kemampuan penalaran logis dari pendekatan simbolik (Piplai et al., 2023:1), Sebagai gambar alurnya bisa dilihat pada gambar 2.1.



Gambar 2.1 Alur *Neuro-Symbolic AI*

Sumber: (Piplai et al., 2023:1)

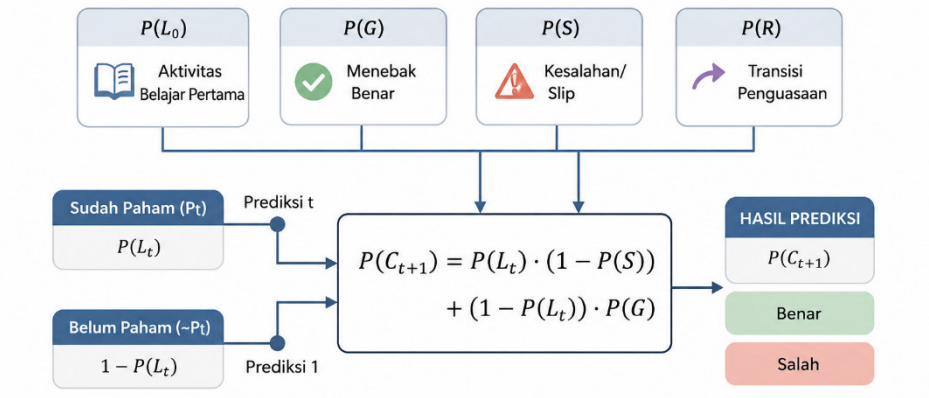
Pendekatan ini dipandang sebagai gelombang ketiga AI karena mengintegrasikan pembelajaran dan penalaran untuk

mengendalikan arsitektur atau fungsi loss, bukan sekadar mengutak-atik input seperti pada teknik prompting di *large language models* (Tran et al., 2025:5). Secara konseptual, *neurosymbolic AI* memadukan kemampuan aproksimasi kuat dari neural network dengan penalaran simbolik dari data terbatas, dan menghasilkan keluaran yang dapat dijelaskan (Sheth et al., 2023:2).

Dengan demikian, Neurosymbolic AI adalah arsitektur kecerdasan buatan gelombang ketiga yang memadukan kekuatan neural network dan penalaran simbolik untuk menangani konsep abstrak, generalisasi data terbatas, serta menghasilkan luaran yang dapat dijelaskan.

b. *Bayesian Knowledge Tracing* (BKT)

BKT dipahami sebagai model probabilistik yang memandang penguasaan keterampilan sebagai state laten biner dan memperbaruinya berdasarkan benar–salahnya respons siswa. Sebagai gambaran BKT sendiri alur dari teori ini bisa dilihat pada gambar 2.2.



Gambar 2.2 Bayesian Knowledge Tracing

Sumber: (Shchepakin et al., 2023:2)

Dengan empat parameter utamanya adalah *Prior Proficiency*, *Transition*, *Guess*, dan *Slip* (Shchepakin et al., 2023:2), Bisa dilihat pada tabel 2.1.

Tabel 2.1 Parameter *Bayesian Knowledge Tracing*

P(L_0)	*Prior Proficiency*	*the probability the learner is in proficient state for the KC prior to first feedback.*
P(G)	*Guess*	*the probability of correct guess at an assessment while not being proficient at the corresponding KC.*
P(S)	*Slip*	*the probability of slip (mistake) at an assessment while being proficient at the corresponding KC.*
P(R)	*Transition*	*the probability of a learner becoming proficient in KC after making an attemptat an assessment and reading the feedback.*
$P(L^{(d)}_t)$	*Proficiency*	*the probability of learner d being in a proficient state after receiving t rounds of feedback.*
$P(C^{(d)}_t)$	*Correctness of an Attempt*	*the probability of learner d answering assessment t correctly.*

Sumber: (Shchepakin et al., 2023:3)

Nilai probabilitas seorang pembelajar atau siswa dalam menjawab soal dengan benar pada langkah berikutnya (t+1). Dapat dihitung dalam perumusan *Bayesian Knowledge Tracing* yang bisa dilihat pada persamaan 2.1.

$$\begin{aligned} & P(C_{t+1}) \\ & = P(L_t) \cdot \big(1 - P(S)\big) \\ & + \big(1 - P(L_t)\big) \\ & \cdot P(G) \end{aligned} \tag{2.1}$$

Keterangan Mengenai Variabel dari rumus persamaan 2.1 diatas adalah berikut :

1) $P(C_{t+1})$: Probabilitas bahwa siswa menjawab pertanyaan berikutnya $(t+1)$ dengan benar (*correctness of an attempt*).
2) $P(L_t)$: Probabilitas bahwa siswa telah menguasai materi (*proficient*) pada langkah ke-t. Maka, $\left(1 - P(L_t)\right)$ merepresentasikan bahwa probabilitas siswa belum menguasai materi tersebut.
3) $P(S)$: Probabilitas kecerobohan (*slip*), yaitu kondisi di mana siswa sebenarnya sudah menguasai materi namun salah dalam menjawab soal.
4) $P(G)$: Probabilitas tebakan (*guess*), yaitu kondisi di mana siswa berhasil menjawab benar melalui tebakan meskipun belum menguasai materi.

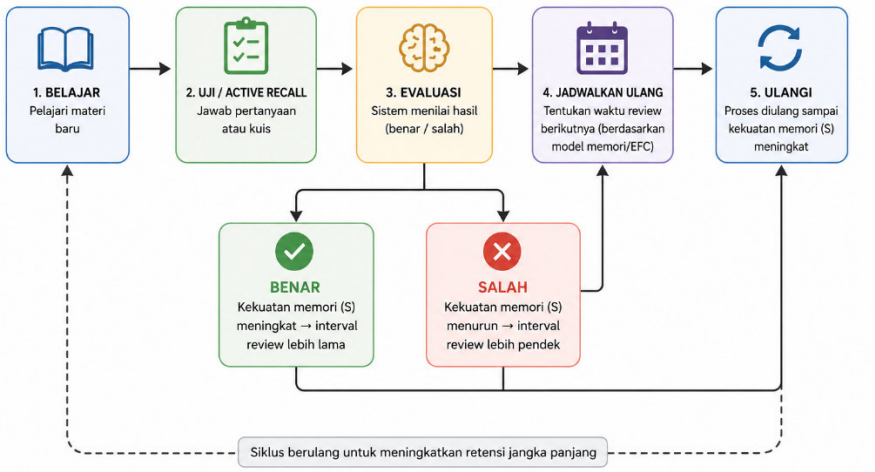
Dalam hal ini sistem mampu memprediksi probabilitas tingkat *proficiency* pembelajar *d* terhadap suatu komponen pengetahuan *Knowledge Component* (KC) yang disimbolkan sebagai $P(L_t^{(d)})$ (Shchepakin et al., 2023:4).

Di dalam ranah *educational learning analytics,* BKT digambarkan sebagai model probabilistik dinamis yang mengikuti kerangka *Hidden Markov* Model untuk mengestimasi state pengetahuan siswa seiring waktu (Bulut et al., 2023:771).

Dengan demikian, BKT tetap relevan sebagai landasan teoretis maupun praktis untuk pengembangan sistem pembelajaran adaptif yang terukur dan dapat dijelaskan.

c. *Spaced Repetition System* (SRS)

Spaced repetition dipahami sebagai teknik untuk meningkatkan proses belajar dengan mengoptimalkan jarak waktu antar pengulangan materi yang sama, misalnya kosakata bahasa asing, dalam sistem *e-learning* berbasis butir-butir informasi kecil. Pendekatan ini dipandang mampu "melawan *forgetting curve*" *Ebbinghaus* dan secara signifikan meningkatkan retensi pengetahuan dibandingkan pembelajaran tradisional (Vagha et al., 2025:2), dalam penggambarannya sendiri bisa dilihat pada gambar 2.3.



Gambar 2.3 *Spaced Repetition System*

Sumber: (Vagha et al., 2025:2)

Exponential Forgetting Curve (EFC) adalah model memori yang menggambarkan penurunan ingatan secara eksponensial seiring berjalannya waktu (Xiao & Wang, 2024:4). Probabilitas pemanggilan kembali (*recall*) suatu materi pembelajaran dinyatakan melalui persamaan 2.2:

$$p_{recall} = e^{-\frac{\theta \cdot \Delta}{S}} \quad (2.2)$$

Keterangan Mengenai Variabel dari rumus persamaan 2.2 diatas adalah berikut :

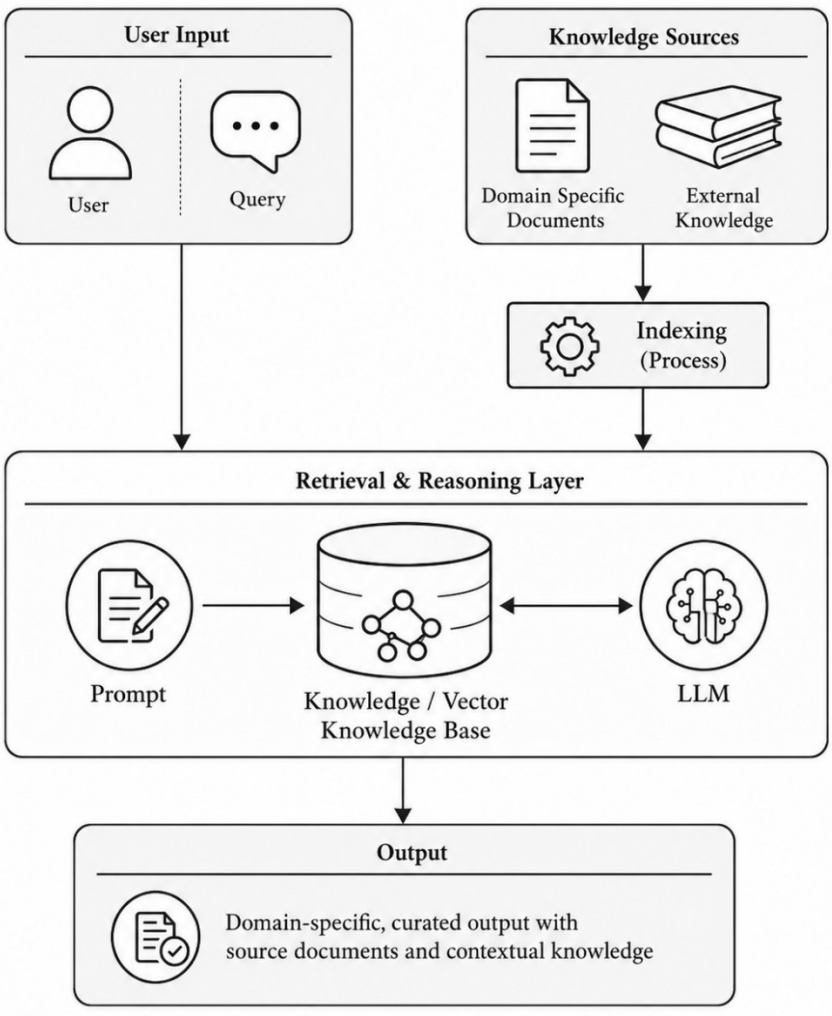
1) $P(\text{recall})$: Probabilitas siswa dapat mengingat kembali materi yang telah dipelajari (retensi memori).
2) e : Bilangan Euler atau konstanta basis logaritma natural ($\approx 2{,}718$).
3) d : Tingkat kesulitan dari materi pembelajaran (*item difficulty*).
4) t : Rentang waktu yang telah berlalu sejak peninjauan atau sesi belajar terakhir (*time elapsed*).
5) s : Kekuatan memori siswa saat ini terhadap materi tersebut (*memory strength*).

Dengan demikian, *spaced repetition* adalah teknik pembelajaran berbasis pengulangan yang mengoptimalkan suatu sistem evaluasi untuk melawan kelupaan terhadap suatu konteks, sehingga meningkatkan retensi memori jangka panjang dan *active recall* secara efektif.

d. *Graph Retrieval-Augmented Generation* (*GraphRAG*)

GraphRAG muncul sebagai paradigma baru yang smengatasi keterbatasan RAG berbasis teks datar melalui representasi pengetahuan berbentuk graf yang secara eksplisit menangkap relasi entitas dan hierarki domain (Zhang et al., 2025:1). Alur arsitekturnya dimulai ketika ontologi dipakai untuk mendeskripsikan domain RAG dan knowledge graph dipakai untuk menyimpan korpus dokumen. Basis

pengetahuan terintegrasi ini menyimpan sekaligus dokumen korpus, konteks domain tambahan, dan vector embeddings (DeBellis et al., 2024:1) Alurnya sendiri bisa dilihat pada gambar 2.4.



Gambar 2.4 Alur *Graph Retrieval-Augmented Generation*

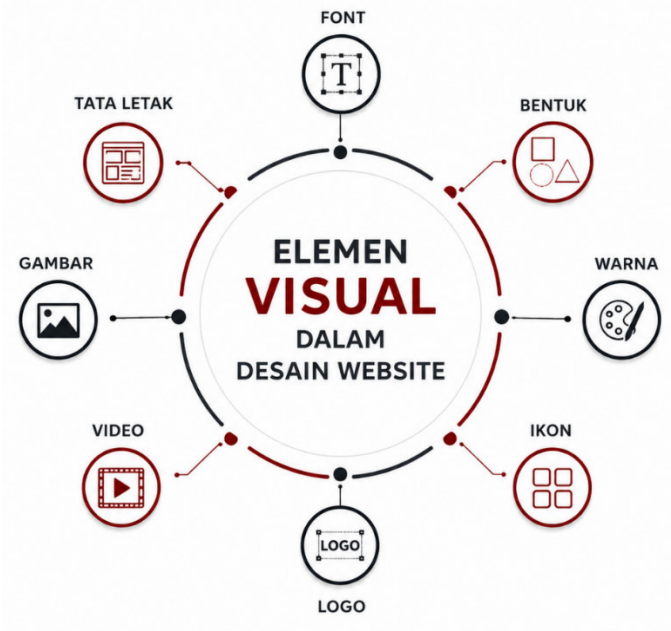
Sumber: (DeBellis et al., 2024:1)

Berbeda dari RAG konvensional yang hanya memfokuskan *retrieval* pada dokumen individual, pendekatan seperti GRAG menekankan pentingnya konteks graf tekstual dan subgraf sehingga LLM dapat memanfaatkan informasi topologis dan tekstual secara bersama-sama untuk penalaran multi-hop yang lebih kuat (Hu et al., 2025:1).

Dengan demikian, *GraphRAG* adalah paradigma baru yang mengintegrasikan manajemen data graf dengan RAG untuk menangkap relasi entitas secara topologis, memungkinkan penalaran multi-hop yang kuat, serta meningkatkan reliabilitas faktual pada LLM.

2. Website

Website modern dipahami sebagai media layanan yang harus cepat, fleksibel, dan mendorong interaksi; jika loading lama dan sulit digunakan, pengguna tidak nyaman dan cenderung meninggalkan situs karena merasa kebutuhannya tidak terpenuhi (Csontos & Heckl, 2025:2), Dari sisi desain, susunan elemen seperti navigasi, jenis huruf, harmoni warna, dan estetika keseluruhan sangat menentukan apakah website mampu menarik pengguna dan memberikan pengalaman yang memuaskan (Tomić et al., 2025:2), Sebagai gambaran teori tersebut bisa dilihat pada gambar 2.5.



Gambar 2.5 Komponen Website Modern

Sumber: (Tomić et al., 2025:2)

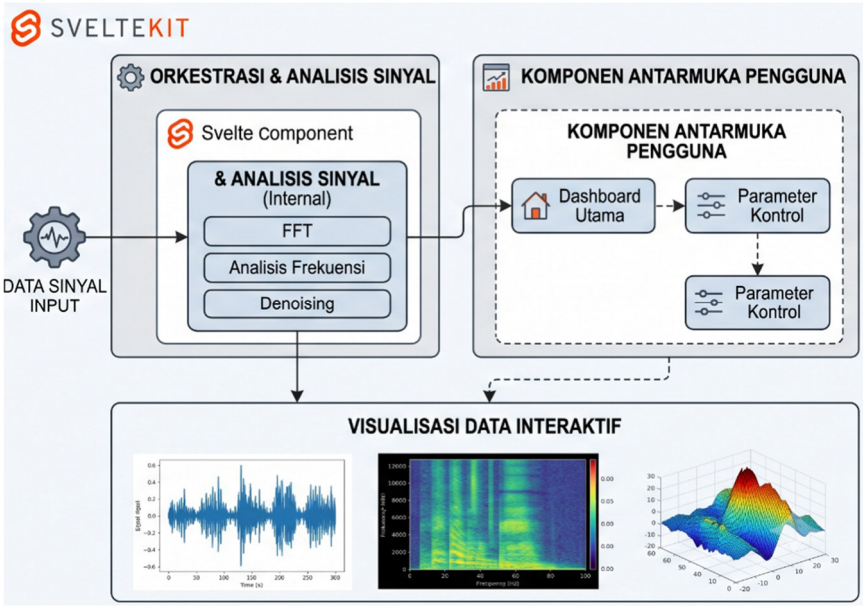
Dengan demikian, website modern harus dirancang cepat, fleksibel, memiliki estetika visual yang menarik, serta aksesibilitas yang baik demi menjamin kenyamanan pengguna dan mencegah mereka meninggalkan situs.

3. *Framework*

a. SvelteKit

Svelte dipahami sebagai *compiler-based framework* yang menghilangkan kebutuhan *virtual DOM* dan menyederhanakan struktur kode, sehingga kebutuhan sumber daya menjadi lebih rendah dan pemeliharaan kode lebih mudah (Nurzyński & Badurowicz, 2025:284).

Dalam implementasi praktis, Svelte digunakan sebagai dasar antarmuka komponen yang dapat dikomposisi untuk mengorkestrasi pemrosesan sinyal dan visualisasi pada aplikasi web berbasis SvelteKit (Seow et al., 2026:2), Lebih jelasnya bisa dilihat pada gambar 2.6.



Gambar 2.6 Implementasi Praktis *Svelte Kit*

Sumber: (Seow et al., 2026:2)

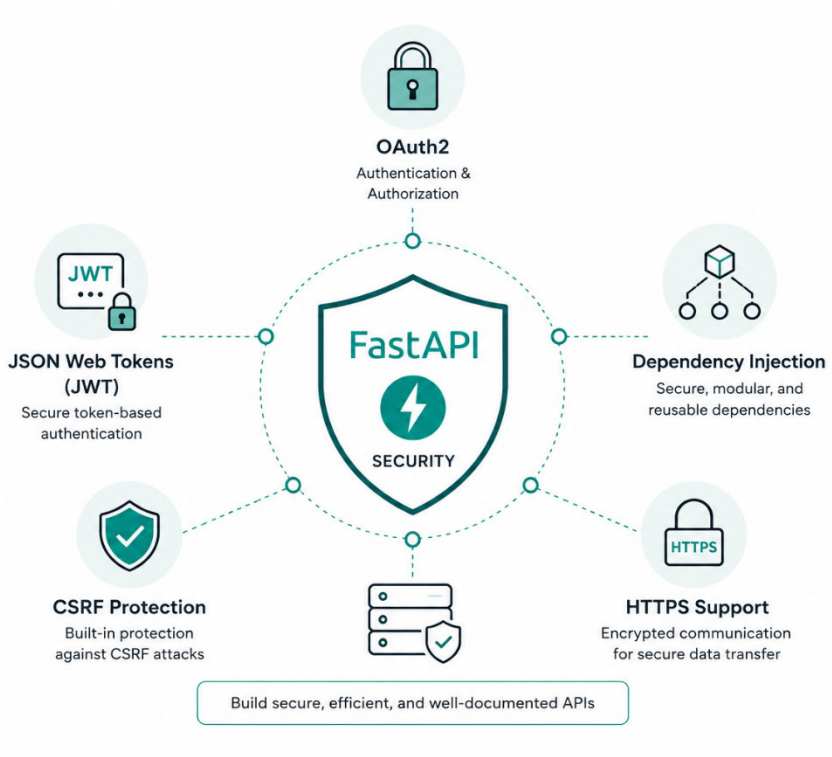
Keunggulan ini dikaitkan dengan arsitektur berbasis *compiler* yang mengubah komponen menjadi JavaScript murni yang dioptimalkan tanpa ketergantungan *virtual DOM* atau *runtime* yang berat (Putra et al., 2025:355).

Dengan demikian, Svelte adalah compiler-based framework tanpa virtual DOM yang menghasilkan JavaScript murni yang ringan dan efisien, walaupun penerapannya tetap harus mempertimbangkan skala aplikasi dan ekosistem komunitasnya.

b. FastAPI

Integrasi algoritma ke dalam platform web kini semakin banyak memanfaatkan FastAPI karena kemampuannya memisahkan skrip algoritma dari backend sehingga arsitektur menjadi modular (Chen, 2023:7)

FastAPI juga banyak digunakan untuk mengekspos model machine learning, dan menyajikan alat pendukung keputusan secara real-time dengan latensi minimal dan throughput tinggi (Odofin et al., 2023:390). Di sisi keamanan, FastAPI menyediakan fitur inti seperti OAuth2, JSON Web Tokens (JWT), dependency injection, serta dukungan CSRF dan HTTPS, sehingga pengembang dapat membangun API yang aman sekaligus tetap efisien dan terdokumentasi dengan baik (Khandelwal, 2024;1), Bisa dilihat pada gambar 2.7.



Gambar 2.7 *Fast Api Scurity*

Sumber: (Khandelwal, 2024:1)

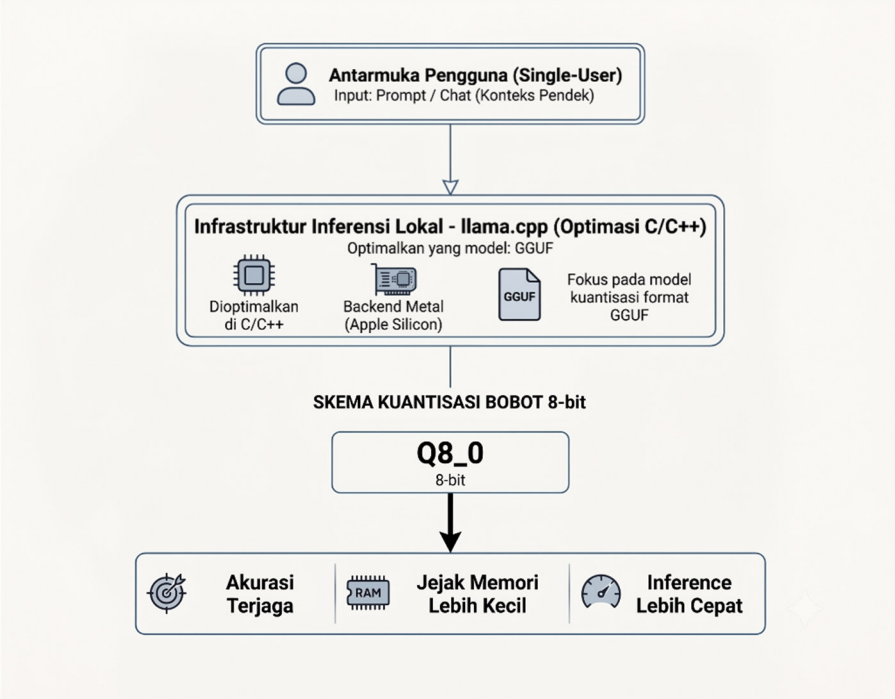
Dengan demikian, FastAPI adalah framework backend berkinerja tinggi yang ideal untuk integrasi model machine learning secara modular, aman (via OAuth2/JWT), dan real-time dengan latensi minimal.

4. Llama-cpp

Implementasi llama.cpp sebagai kerangka kerja inference lokal yang dioptimalkan di C/C++ menjadikannya fondasi bagi banyak eksperimen efisiensi LLM, dengan dukungan beberapa skema kuantisasi bobot dan tingkat presisi bit yang berbeda (Husom et al., 2025:5).

Dari sisi penelitian sistem, *PowerInfer* mengintegrasikan sekitar 4.200 baris kode C++ dan CUDA langsung ke dalam llama.cpp sebagai "*a*

state-of-the-art open-source LLM inference framework designed for PCs", dan melaporkan bahwa semua komponen lain dari llama.cpp dibiarkan tidak berubah sebagai baseline engine (Song et al., 2024:9), Sebagai ilustrasi bisa dilihat pada gambar 2.8.



Gambar 2.8 Optimasi Skema Kuantisasi

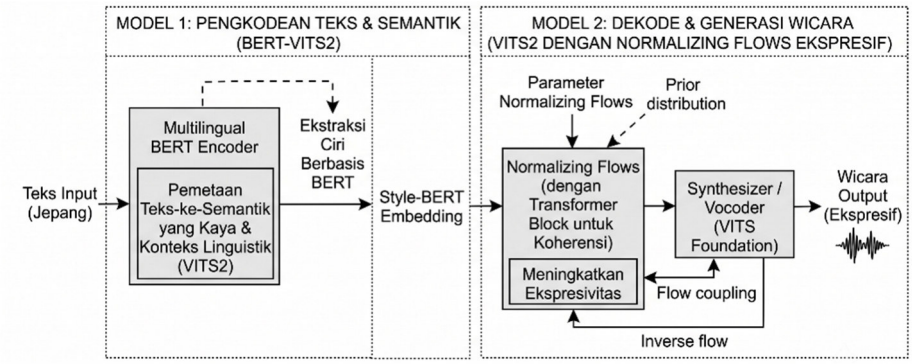
Sumber: (Song et al., 2024:9)

Di sisi lain, studi kuantisasi LLM menyebut bahwa Ollama hanyalah abstraksi tingkat tinggi di atas llama.cpp, dengan format kuantisasi q4, q8, dan varian K-means yang berbeda untuk menyeimbangkan akurasi, jejak memori, dan kecepatan inference (Husom et al., 2025:5).

Dengan demikian, llama.cpp adalah *framework inference* lokal berbasis C/C++ yang sangat teroptimisasi untuk kuantisasi bobot (format GGUF), yang menjadi fondasi berkinerja tinggi bagi akselerasi lokal (seperti Metal) maupun abstraksi tingkat tinggi seperti Ollama.

5. Style-Bert-VITS2

Dalam kajian terkait TTS ekspresif berbahasa Jepang, VITS2 digambarkan mempertahankan komponen fondasional VITS namun menambahkan transformer block pada normalizing flows untuk meningkatkan koherensi dan ekspresivitas wicara (Rackauckas & Hirschberg, 2025:2). Pada jalur yang sama, BERT-VITS2 memperluas VITS2 dengan mengintegrasikan multilingual BERT encoder sehingga pemetaan teks-ke-semantik menjadi lebih kaya dan mampu menangani konteks linguistik yang beragam (Rackauckas & Hirschberg, 2025:2), Sebagai mana bisa dilihat pada gambar 2.9.



Gambar 2.9 Arsitektur Style-Bert-Vits2

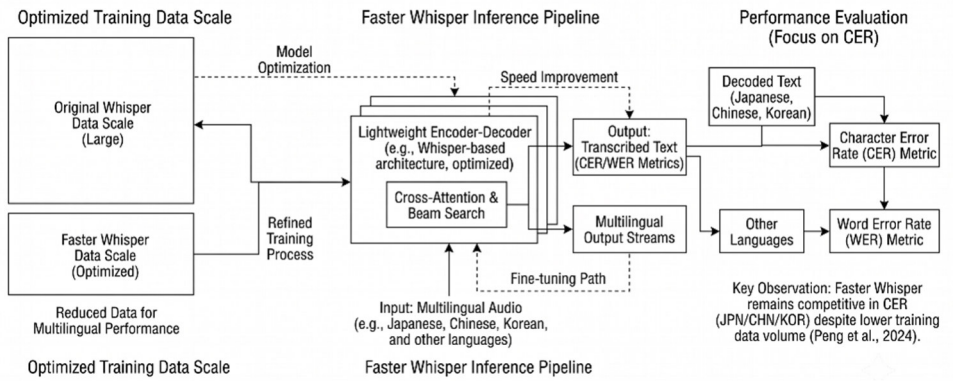
Sumber: (Rackauckas & Hirschberg, 2025:2)

Dengan demikian, Style-BERT-VITS2 adalah evolusi VITS2 yang memadukan *BERT encoder* untuk kekayaan semantik teks dan *style embeddings* untuk kontrol emosi serta ekspresi wicara yang halus.

6. Faster Whisper

Kemampuan Whisper sebagai model multibahasa yang menjadi dasar banyak implementasi cepat seperti Faster Whisper dibangun dari pra-latih pada 680.000 jam audio multibahasa (96 bahasa), sehingga mampu

melakukan transkripsi dan terjemahan secara kuat di berbagai lingkungan kebisingan, aksen, dan istilah khusus (Bajo et al., 2024:2). Dalam pengujian ASR multibahasa, kinerja untuk bahasa Jepang biasanya diukur menggunakan *Character Error Rate (CER)*, bersama bahasa Tionghoa dan Korea, memperlihatkan bahwa model bergaya Whisper dapat tetap kompetitif meski dilatih pada data yang jauh lebih sedikit dibanding Whisper asli (Peng et al., 2024:6), Sebagai ilustrasi bisa dilihat pada gambar 2.10.



Gambar 2.10 Konsep Arsitektur Faster Whisper

Sumber: (Peng et al., 2024:6)

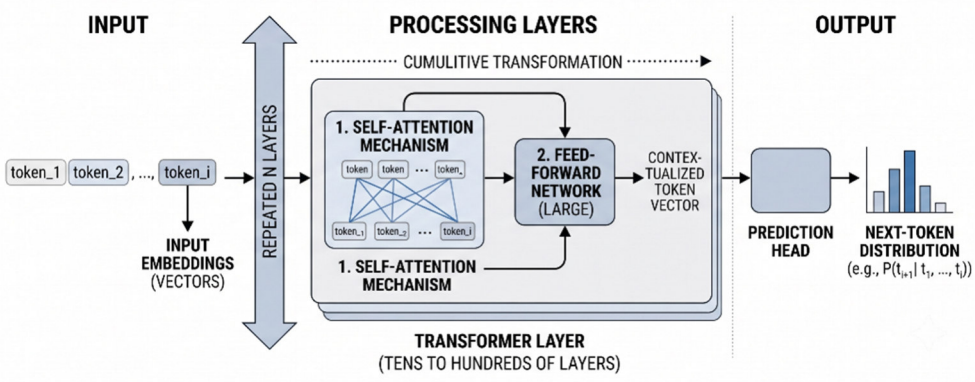
Dengan demikian, Whisper adalah model ASR multibahasa tangguh yang performanya pada bahasa Jepang (diukur via CER) dapat dioptimalkan secara signifikan melalui *fine-tuning* (LoRA/*end-to-end*) pada varian kecil seperti Whisper-Tiny hingga melampaui Whisper-Base.

7. *Large Language Model* (LLM)

Large Language Models umumnya dipahami sebagai model bahasa berbasis Transformer dengan jumlah parameter yang sangat besar, sehingga menunjukkan kapasitas kuat untuk memahami bahasa alami dan

menyelesaikan tugas kompleks melalui generasi teks (W. X. Zhao et al., 2026:4).

Perkembangan dari statistical LM → neural LM → pre-trained language models (PLM) → LLM ditandai pergeseran dari pelatihan terawasi tugas-spesifik ke pra-latih *self-supervised* pada korpus besar, di mana peningkatan skala parameter dan ukuran data memberikan lonjakan kinerja signifikan serta memicu lahirnya berbagai LLM baru (Naveed et al., 2024:2). Secara praktis, LLM modern biasanya berupa model Transformer dengan puluhan hingga ratusan lapisan, memetakan setiap token ke vektor terkontekstualisasi melalui jaringan self-attention dan feed-forward besar, dengan jumlah parameter dari ratusan juta hingga ratusan miliar untuk memprediksi distribusi token berikutnya (Kumar, 2024:13), Bisa dilihat pada gambar 2.11.



Gambar 2.11 Alur Proses *Large Language Model*

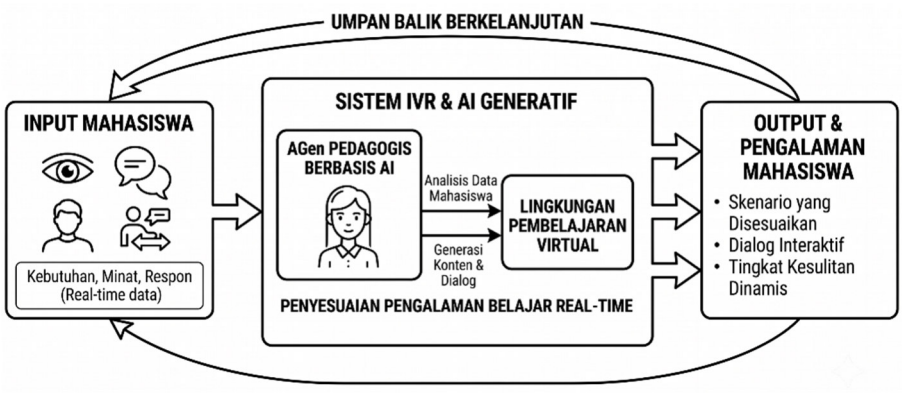
Sumber: (Kumar, 2024:13)

Dengan demikian, LLM adalah model bahasa berbasis arsitektur Transformer yang berevolusi dari statistical/neural LM menjadi model berskala miliaran parameter melalui self-supervised pre-training, sehingga

mampu memahami bahasa alami dan menyelesaikan tugas kompleks secara generatif.

8. *Virtual Reality Model* (VRM)

VR dalam pendidikan dipahami sebagai teknologi interaksi manusia–komputer yang "menciptakan kesan realistis" melalui simulasi indera dalam lingkungan buatan yang memungkinkan eksplorasi bebas dan praktik tanpa batasan fisik (Criollo-C et al., 2024:114371). Dari sisi desain pedagogis, pendekatan *Intelligent Virtual Reality* (IVR) menggabungkan karakter pedagogis berbasis *generative AI* untuk menyesuaikan pengalaman belajar secara real-time dengan kebutuhan dan minat mahasiswa (Hemminki-Reijonen et al., 2025:7), Sebagai mana bisa dilihat pada gambar 2.12.



Gambar 2.12 Diagram Pedagogis *Virtual Reality*

Sumber: (Hemminki-Reijonen et al., 2025:7)

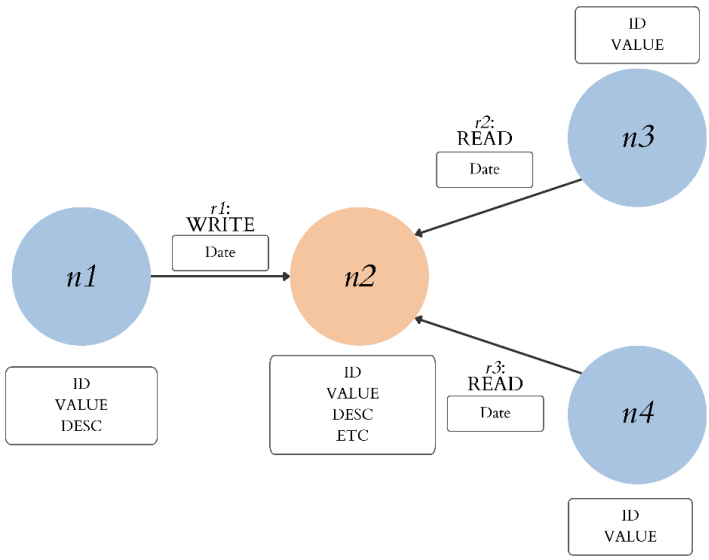
Dengan demikian, VR dalam pendidikan paling efektif diterapkan melalui arsitektur *Intelligent Virtual Reality* (IVR) berbasis *generative AI*, yang mampu mensimulasikan lingkungan imersif bebas hambatan fisik sekaligus mempersonalisasikan materi secara *real-time* sesuai beban kognitif siswa.

9. Database

a. Neo4j

Neo4j dikenal sebagai salah satu graph database paling banyak digunakan karena model labeled property graph yang memungkinkan baik node maupun edge menyimpan atribut sebagai pasangan key–value, sehingga sangat efisien untuk melakukan traversal graf pada aplikasi seperti pendeteksian kecurangan, analisis jejaring sosial, dan sistem rekomendasi (Anuyah et al., 2024:3).

Bahasa query Cypher pada Neo4j dirancang khusus untuk struktur graf dengan sintaks yang intuitif dan memungkinkan pattern matching relasi yang kompleks, sehingga efisien untuk traversal dan query berbasis hubungan yang sulit dilakukan di basis data relasional (Anuyah et al., 2024:3). Sebagai ilustrasi relasi antar *node* dan *edge* bisa dilihat pada gambar 2.13.



Gambar 2.13 Ilustrasi Hubungan Antar Nodes

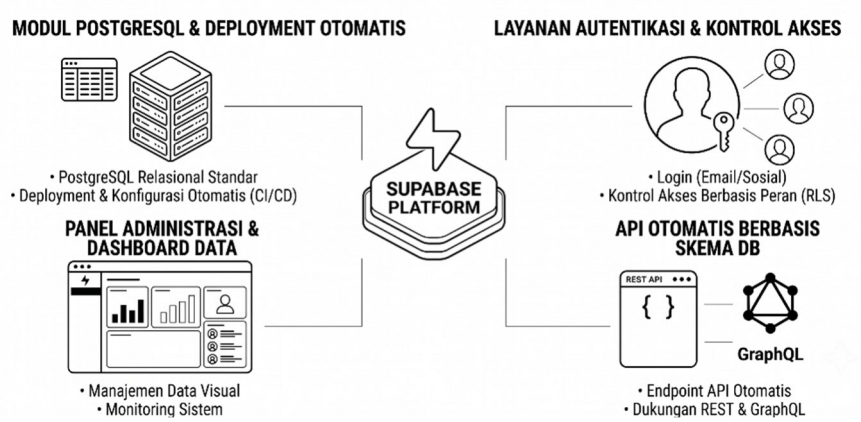
Sumber: (Tang et al., 2025:3)

Cypher juga dipahami sebagai bahasa kueri graf yang deklaratif, dengan MATCH sebagai klausa untuk mendefinisikan graph pattern dan RETURN untuk menentukan data apa yang dikembalikan. Selain fitur inti tersebut, Cypher mendukung pola yang lebih kompleks seperti chained MATCH clauses, OPTIONAL MATCH, variable-length paths, sorting, agregasi, dan UNION ALL, sehingga cukup ekspresif untuk berbagai kebutuhan kueri graf (Tang et al., 2025:3).

Dengan demikian, Neo4j adalah *graph database* berbasis *labeled property graph* yang sangat efisien untuk traversal data kompleks menggunakan bahasa *query* Cypher berbasis *pattern matching*, melewati keterbatasan operasi *join* pada basis data relasional.

b. Supabase

Secara arsitektural, Supabase diposisikan sebagai platform BaaS dan FaaS (*Function as a Service*) yang "relatif terhadap *Backend as Service* dan *Function as Service* dalam kasus tertentu seperti pengembangan information system yang hanya perlu *proof of concept,* Supabase adalah platform tool yang akan membantu secara radikal meningkatkan *Time to Market*" (Kolesnikov et al., 2024:100). Supabase menyediakan berbagai solusi siap pakai, antara lain PostgreSQL ter-deploy otomatis, autentikasi, panel administrasi, hingga API yang dihasilkan dari skema database (Kolesnikov et al., 2024:100), Sebagaimana bisa dilihat pada gambar 2.14.



Gambar 2.14 Konsep Alur Modul *Supabase*

Sumber: (Kolesnikov et al., 2024:100)

Dengan demikian, Supabase adalah platform BaaS dan FaaS berbasis PostgreSQL yang menyediakan autentikasi, panel admin, dan API otomatis untuk memangkas konfigurasi infrastruktur serta mempercepat *Time to Market* (PoC)..

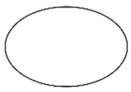
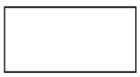
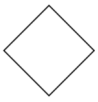
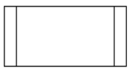
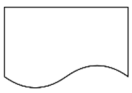
10. Flowchart

Flowchart merupakan cara umum untuk menggambarkan proses logis dalam desain rekayasa dan pengembangan perangkat lunak, terutama karena node dan alurnya dapat merepresentasikan proses yang kompleks secara terstruktur dan terurut (P. Zhang et al., 2023:1).

Di lingkungan apapun, metode flowchart menyederhanakan komunikasi dan memungkinkan analisis proses yang efektif; flowchart digunakan untuk berbagai tujuan termasuk desain proses yang sesuai dengan hasil yang ditargetkan, mendukung pengambilan keputusan bisnis, serta membantu menemukan kesalahan dalam pengembangan program (Kis & Viharos, 2024:2).

Dengan demikian, *flowchart* adalah metode representasi visual terstruktur yang menyederhanakan komunikasi proses logis, membantu *debugging* dalam rekayasa perangkat lunak, serta mempercepat retensi materi prosedural dalam pendidikan, Sebagaimana Simbol-simbol *flowchart* bisa dilihat pada tabel 2.2.

Tabel 2.2 Simbol-Simbol *Flowchart*

Simbol	Nama	Fungsi
	Terminal	Menandai titik awal (*start*) atau titik akhir (*end*).
	Input/Output	Menunjukkan proses penerimaan data.
	Process	Menunjukkan operasi pemrosesan pengolahan data.
	Decision	Menandai titik penyeleksian kondisi untuk menentukan alur berikutnya.
	Connector	Menghubungkan alur proses yang terputus pada halaman/lembar kerja yang sama.
	Predefined Process	Menunjukkan pelaksanaan suatu subproses yang telah didefinisikan sebelumnya.
	Document	Menunjukkan input atau output dalam bentuk dokumen.
	Flow Line	Menunjukkan arah aliran atau urutan jalannya proses.

Sumber: (Tarsini & Anggraeni, 2024:3)

11. *Unified Modeling Language*

Unified Modeling Language dipahami sebagai bahasa visual standar untuk memodelkan, mendesain, dan mendokumentasikan sistem perangkat lunak maupun non-perangkat lunak. UML menyediakan notasi baku dan beragam jenis diagram yang memudahkan komunikasi dan kolaborasi antar anggota tim pengembang dalam menggambarkan struktur dan perilaku sistem yang kompleks (Hindarto & Hariadi, 2023:626).

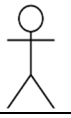
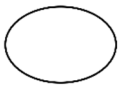
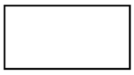
Dengan demikian, UML adalah bahasa visual standar dengan notasi baku yang berfungsi untuk memodelkan, mendesain, dan mendokumentasikan cetak biru sistem secara terstruktur guna menyederhanakan komunikasi serta kolaborasi tim pengembang.

a. *Use case diagram*

Penggunaan *use case diagram* yang selaras dengan *Software Requirement Specification* (SRS) sangat penting karena SRS menjadi "blueprint" kebutuhan fungsional dan nonfungsional, sementara use case diagram memberikan representasi visual interaksi sistem–pengguna sehingga keduanya harus dianalisis dan disejajarkan agar tidak terjadi misimplementasi kebutuhan system (Umar et al., 2025:2).

Dengan demikian, penyelarasan *use case diagram* sebagai representasi visual interaksi aktor-sistem sangat krusial untuk memastikan konsistensi dokumentasi dan mencegah misimplementasi saat pengembangan kode. Sebagaimana Simbol-simbol *Use case diagram* bisa dilihat pada tabel 2.3,

Tabel 2.3 Simbol-Simbol *Use Case Diagram*

Simbol	Nama	Fungsi
	Actor	Menggambarkan peran pengguna
	Use Case	Menggambarkan rincian fungsionalitas kepada aktor.
	System Boundary	Membatasi ruang lingkup interaksi antara aktor dan fitur di luar dan dalam sistem.
	Association	Penghubung aktor dengan *use case*.
	Extend	Menunjukkan bahwa *use case* target secara opsional dapat memperluas fungsionalitas *use case* sumber pada kondisi tertentu.
	Generalization	Menunjukkan hubungan spesialisasi atau pewarisan (*inheritance*) antara aktor atau *use case* umum ke yang lebih spesifik.

Sumber: (Rospricilia et al., 2024:4)

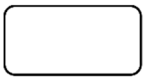
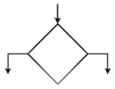
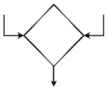
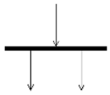
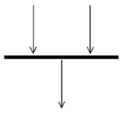
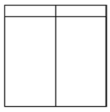
b. *Activity diagram*

Dalam konteks pemodelan proses, activity diagram didefinisikan sebagai representasi grafis dari suatu proses, di mana node-node yang saling terhubung dengan edge digunakan untuk menunjukkan alur kontrol dan informasi pada sistem yang dimodelkan (Jha et al., 2023:6).

Dengan demikian, *activity diagram* adalah representasi grafis terstruktur berbasis *node* dan *edge* yang menggambarkan dinamika alur kerja (*workflow*), kontrol, dan informasi sistem, termasuk logika

percabangan serta proses paralel, Sebagaimana Simbol-simbol *Activity diagram* bisa dilihat pada tabel 2.4.

Tabel 2.4 Simbol-Simbol *Activity Diagram*

Simbol	Nama	Fungsi
	Initial Node	Menandai titik awal dimulainya suatu alur aktivitas.
	Activity / Action	Menggambarkan suatu tindakan, proses, atau operasi yang sedang dilakukan oleh pengguna atau sistem.
	Control Flow	Menunjukkan urutan eksekusi atau perpindahan alur kerja.
	Decision Node	Menandai titik percabangan alur kerja berdasarkan evaluasi kondisi.
	Merge Node	Menggabungkan kembali beberapa alur percabangan bercabang menjadi satu alur tunggal.
	Fork Node	Memecah satu alur kerja menjadi beberapa alur kerja.
	Join Node	Menggabungkan kembali beberapa alur paralel menjadi satu alur tunggal.
	Swimlane	Memisahkan tanggung jawab aktivitas berdasarkan entitas
	Final Node	Menandai titik akhir selesainya seluruh rangkaian aktivitas dalam diagram.

Sumber: (Siewe & Ngounou, 2025:4)

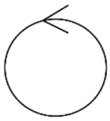
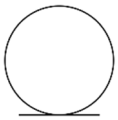
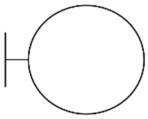
c. Sequence diagram

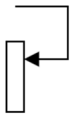
Sequence diagram dipandang sebagai diagram interaksi yang merinci bagaimana operasi dijalankan dan menekankan urutan pesan secara visual dengan menggunakan sumbu vertikal sebagai representasi

waktu, sedangkan objek ditempatkan pada sumbu horizontal dan dilengkapi lifeline, serta berbagai jenis pesan sinkron, asinkron, self, create, dan destroy (Suriya & S., 2023:5).

Dengan demikian, sequence diagram adalah diagram interaksi yang menekankan urutan kronologis pesan secara visual melalui sumbu waktu vertikal dan barisan objek horizontal, lengkap dengan komponen lifeline, activation bar, serta berbagai tipe pesan fungsional, Sebagaimana Simbol-simbol *Activity diagram* bisa dilihat pada tabel 2.5.

Tabel 2.5 Simbol-Simbol *Sequence Diagram*

Simbol	Nama	Fungsi
	Actor	Menggambarkan interaksi pengguna.
	Control Object	Mengkoordinasikan perilaku sistem dan mengontrol alur kerja sistem.
	Entity Object	Suatu Object yang mencakup informasi kegiatan dan disimpan pada database.
	Boundary Object	Sebuah Object yang menjadi penghubung sistem
	Lifeline	Menggambarkan keberadaan dan rentang waktu hidup suatu objek/komponen selama interaksi berlangsung.
	Activation Box	Menunjukkan periode waktu ketika objek sedang aktif melakukan pemrosesan.

	Synchronous Message	Pengiriman pesan di mana pengirim menunggu respons hingga pemrosesan selesai
	Asynchronous Message	Pengiriman pesan di mana pengirim langsung melanjutkan proses tanpa menunggu balasan.
	Return Message	Menunjukkan pesan balasan atau pengembalian nilai (*return value*) dari operasi yang dipanggil sebelumnya.
	Self Call	Menunjukkan operasi internal yang dilakukan oleh objek terhadap dirinya sendiri.
	Combined Fragment	Membatasi area logika bersyarat (alt untuk percabangan/if-else, loop untuk pengulangan aktivitas).

Sumber: (Suriya & S., 2023:5)

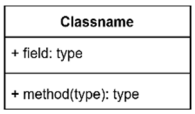
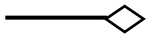
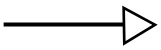
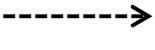
d. Class diagram

Dalam pemodelan berorientasi objek, class diagram digunakan untuk menggambarkan struktur statis sistem melalui kelas, atribut, metode, dan relasi antar kelas. Class diagram merinci struktur data yang digunakan oleh aplikasi, termasuk relasi antar kelas yang diperlukan untuk mengelola data sehingga setiap entitas penting dalam sistem dapat direpresentasikan secara jelas (Putri & Wasino, 2025:4).

Dengan demikian, *class diagram* adalah pemodelan orientasi objek yang menggambarkan struktur statis sistem dengan memetakan kelas, atribut, dan metode, serta mendefinisikan relasi antar-kelas (seperti

asosiasi hingga pewarisan) demi menjaga integritas arsitektur data, Sebagaimana Simbol-simbol *Activity diagram* bisa dilihat pada tabel 2.6.

Tabel 2.6 Simbol-Simbol *Class Diagram*

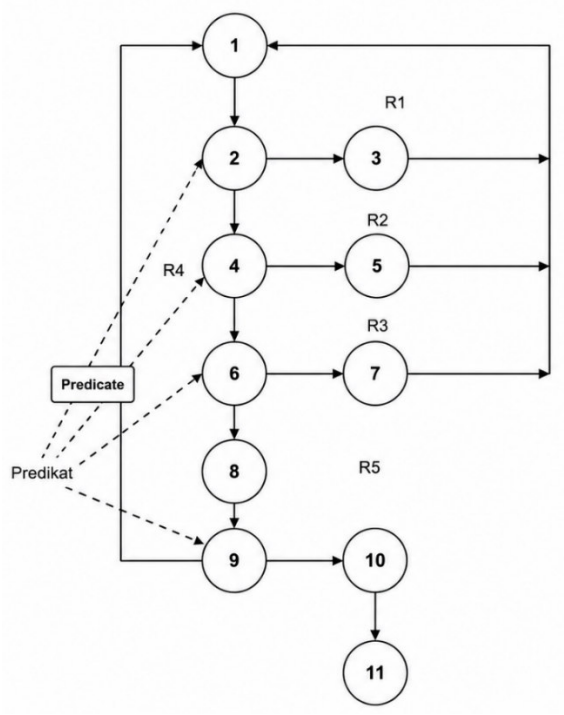
Simbol	Nama	Fungsi
	Class	Menggambarkan struktur kelas.
	Association	Menunjukkan hubungan struktural statis antar kelas.
	Aggregation	Hubungan kepemilikan lemah (*has-a*) di mana objek bagian dapat berdiri sendiri secara independen.
	Composition	Hubungan kepemilikan kuat (*part-of*) di mana jika objek induk dihapus, maka objek bagian di dalamnya juga akan terhapus.
	Generalization	Menunjukkan hubungan pewarisan sifat (*is-a*) dari kelas induk (*superclass*) ke kelas turunan (*subclass*).
	Dependency	Menunjukkan bahwa perubahan pada satu kelas dapat mempengaruhi kelas lainnya secara fungsional.

Sumber: (Baul et al., 2025:236)

12. Pengujian

a. *White-Box Testing*

White-box testing adalah teknik pengujian yang menekankan pemeriksaan struktur internal dan logika kode program, sehingga penguji dapat menilai apakah setiap bagian program berjalan sesuai rancangan (Pirdaus & Hidayana, 2024:69). Pendekatan ini relevan karena pengujian *black-box* yang umum dipakai sering tidak mampu menilai aspek eksekusi nontrivial seperti efisiensi, penggunaan memori, atau algoritma internal (Caniço & Santos, 2023:69), Rancangan alur tersebut bisa dilihat pada gambar 2.15.



Gambar 2.15 *The Flowgraph*

Sumber: (Pirdaus & Hidayana, 2024:73)

Berdasarkan struktur diagram alir (*flowgraph*), nilai kompleksitas ini dapat dihitung menggunakan pendekatan rumus persamaan 2.3 berikut:

1) Berdasarkan Jumlah *Edges* dan *Nodes*

$$V(G) = E - N + 2 \quad (2.3)$$

Keterangan Mengenai Variabel dari rumus persamaan 2.3 diatas adalah berikut :

a) E merupakan jumlah *edges* (garis alur/panah)

b) N merupakan jumlah *nodes* (simpul pernyataan/lingkaran).

Dengan demikian, *white-box testing* sangat krusial untuk memeriksa struktur internal dan logika kode program, karena mampu menilai aspek eksekusi nontrivial (seperti efisiensi, memori, dan algoritma) yang sering kali luput dari pengujian *black-box*.

1) *Property-Based Fuzz Testing (PBT)*

Property-Based Testing (PBT) merupakan pendekatan pengujian yang memeriksa apakah suatu properti umum selalu terpenuhi pada domain input tertentu, bukan sekadar memverifikasi beberapa contoh kasus tertentu (Maaz et al., 2025:1).

Dalam praktiknya, pengembang terlebih dahulu merumuskan spesifikasi formal sebagai properti yang dapat dieksekusi, lalu sistem pengujian akan memeriksa properti tersebut terhadap ratusan hingga ribuan input acak yang dihasilkan secara otomatis (Goldstein et al., 2024:1).

secara formal menurut (Reis, 2025:3-4), prinsip dasar PBT dirumuskan sebagai Persamaan 2.4 berikut:

$$\forall x \in X, \quad P(x) = \text{True} \tag{2.4}$$

Keterangan Mengenai Variabel dari rumus persamaan 2.4 diatas adalah berikut :

a) $\forall x \in X$: Untuk setiap kemungkinan sampel input x di dalam domain ruang input X (misal: semua nilai probabilitas antara 0.0 hingga 1.0).
b) $P(x)$: Fungsi predikat properti/aturan yang diuji. Hasil eksekusi fungsi terhadap input x harus selalu menghasilkan nilai benar (*True*).

Dibandingkan pengujian berbasis contoh, PBT lebih mampu menjangkau variasi input yang luas dan lebih efektif untuk menemukan *counterexample* yang melanggar invariannya (Maaz et al., 2025:1).

Dengan demikian, *Property-Based Testing* (PBT) adalah pendekatan pengujian berbasis rumusan properti formal yang diuji terhadap ribuan input acak otomatis, sehingga sangat unggul dalam memverifikasi *invariant*, menemukan kesalahan logika, serta mendeteksi *bug* secara sistematis jika diintegrasikan dengan *fuzzing*.

2) Metamorphic Testing (MT)

Masalah *test oracle* muncul ketika penguji sulit atau tidak mampu menentukan apakah keluaran sistem sudah benar untuk suatu masukan tertentu. Dalam literatur terbaru, oracle problem dijelaskan sebagai hambatan mendasar dalam otomatisasi pengujian karena sistem sering tidak memiliki mekanisme praktis untuk membedakan keluaran yang benar dan salah secara otomatis (Ayerdi et al., 2024:1).

Untuk mengatasi masalah tersebut, Metamorphic Testing (MT) menggunakan relasi antareksekusi program alih-alih memeriksa kebenaran setiap keluaran secara terpisah. MT dijelaskan sebagai pendekatan yang memanfaatkan metamorphic relations (MRs), yaitu hubungan yang seharusnya tetap berlaku antara input dan output pada dua atau lebih eksekusi yang saling berkaitan (Cho et al., 2025:2).

Secara formal, perinsip dari MT dapat dilihat pada persamaan 2.5 berikut:

$$R\big(f(x), f(x_{\text{new}})\big) = \text{True} \tag{2.5}$$

Keterangan Mengenai Variabel dari rumus persamaan 2.5 diatas adalah berikut :

a) Jika input awal x (siswa menjawab benar) menghasilkan kekuatan memori S, maka dibuat input transformasi x_{new} di mana

siswa menjawab benar dengan waktu pengerjaan yang jauh lebih cepat.

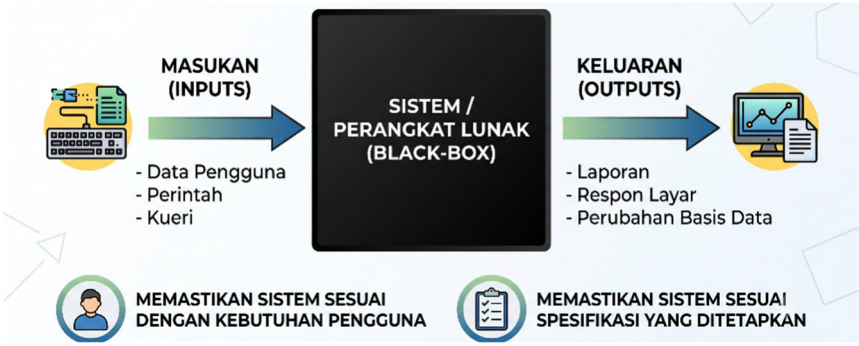
b) Hubungan metamorfiknya (R) menentukan bahwa output kekuatan memori baru harus lebih besar atau sama dengan output awal ($S_{\text{new}} \geq S$). Jika nilainya justru lebih kecil, maka ditemukan *bug* pada logika algoritma adaptif.

Dengan demikian, *Metamorphic Testing* (MT) merupakan solusi efektif untuk mengatasi *test oracle problem* dengan memanfaatkan relasi antar-eksekusi (*Metamorphic Relations*), meskipun keberhasilannya sangat bergantung pada ketepatan perumusan relasi tersebut agar mampu merefleksikan cacat sistem yang sesungguhnya.

b. *Black-Box Testing*

Black-box testing adalah metode pengujian perangkat lunak yang menilai fungsi sistem berdasarkan masukan dan keluaran tanpa meninjau struktur kode internal. Pendekatan ini digunakan untuk memastikan bahwa aplikasi telah berjalan sesuai kebutuhan pengguna dan spesifikasi yang ditetapkan (Mintarsih, 2023:34).

Fokus pengujian ini terletak pada antarmuka, alur fungsi, serta kesesuaian perilaku sistem terhadap requirement, sehingga penguji tidak harus memahami source code, tetapi harus memahami ekspektasi sistem (Mintarsih, 2023:34), Sebagai ilustrasi bisa dilihat pada gambar 2.16.



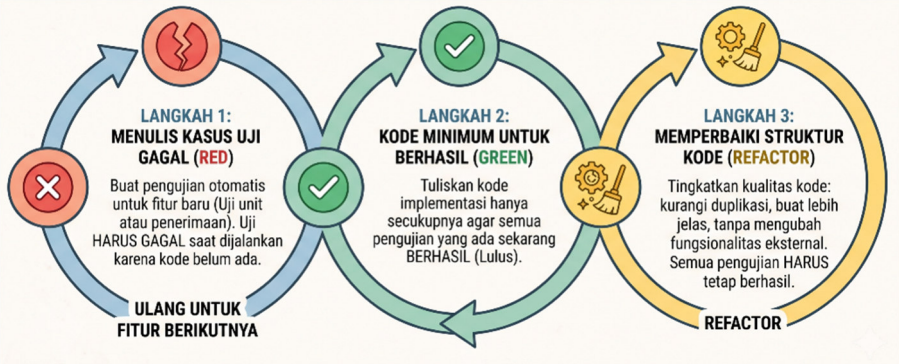
Gambar 2.16 Ilustrasi alur *Black-box*

Sumber: (Mintarsih, 2023:34)

Dengan demikian, *black-box testing* adalah metode pengujian perangkat lunak yang berfokus pada evaluasi fungsionalitas, antarmuka, dan kesesuaian perilaku sistem berdasarkan masukan serta keluaran (*input-output*) sesuai spesifikasi pengguna, tanpa perlu memeriksa struktur kode internalnya.

13. *Test-Driven Development* (TDD)

Test-Driven Development (TDD) merupakan metode pengembangan perangkat lunak yang menempatkan penulisan tes sebelum kode produksi dan menggabungkan aktivitas pemrograman, pengujian unit, serta refactoring dalam siklus iteratif (Agha et al., 2023:268). Dalam praktiknya, TDD dijalankan melalui pola pengembangan bertahap yang menulis kasus uji gagal terlebih dahulu, lalu mengembangkan kode minimum agar pengujian berhasil, sebelum akhirnya memperbaiki struktur kode melalui refactoring (Agha et al., 2023:268), Alur lebih jelasnya sendiri bisa dilihat pada gambar 2.17.



Gambar 2.17 Alur Pengembangan *Test-Driven Development*

Sumber: (Agha et al., 2023:268)

Perkembangan riset terbaru juga memperlihatkan integrasi TDD dengan Generative AI, LLM, serta CI/CD. Studi tentang code generation berbasis LLM menyebut bahwa TDD dapat menjadi kerangka penting untuk memvalidasi *correctness* dan *functionality* dari kode yang dihasilkan model, karena output model dapat terus diuji terhadap spesifikasi yang telah didefinisikan (Mathews & Nagappan, 2024:1).

Dengan demikian, TDD adalah metode desain perangkat lunak berbasis siklus tes awal (*red*), kode minimum (*green*), dan *refactoring* berkelanjutan yang meningkatkan kualitas serta *maintainability* kode, meskipun menantang dalam hal waktu dan keahlian, serta kini kian relevan diintegrasikan dengan *Generative AI/LLM* dan *CI/CD*.

14. Agile

Agile cocok dipakai dalam metodologi penelitian ketika desain riset perlu berjalan secara iteratif dan terbuka terhadap penyempurnaan berulang. Dalam studi *action research* tentang transisi ke agile, rancangan penelitian dijelaskan sebagai proses yang tidak hanya mengamati, tetapi juga

melibatkan partisipasi aktif pemangku kepentingan dalam pemecahan masalah dan pengambilan keputusan (Natarajan & Pichai, 2023:1). Di luar software murni, paper terlampir juga menunjukkan bahwa agile semakin dipakai pada konteks yang lebih luas sehingga memperkuat kelayakannya sebagai kerangka metodologis yang fleksibel. Dalam konstruksi skala besar, agile digambarkan menekankan *iterative planning, cross-functional collaboration*, dan *rapid customer feedback loops* (Gur et al., 2025:1368), Struktur dari pada Metodologi Agile bisa dilihat pada gambar 2.18.

Gambar 2.18 Metodologi Agile

Sumber: (Gur et al., 2025:1368)

Namun, bukti yang sama juga menekankan bahwa konteks sangat menentukan: agile unggul untuk fleksibilitas, paparan risiko dini, dan komunikasi pada proyek kompleks, sedangkan pendekatan tradisional tetap berguna pada proyek linear dan sangat terikat regulasi (Gur et al., 2025:1374).

Dengan demikian untuk pengembangan metodologi penelitian, agile paling kuat bila dipakai pada studi yang lingkungan proyeknya sering berubah cepat, melibatkan banyak objek, dan membutuhkan umpan balik berulang selama proses berjalan.

B. Kajian Empiris

Neuro-symbolic AI (NSAI) menggabungkan kekuatan jaringan saraf yang mahir belajar dari data dengan penalaran simbolik yang terstruktur dan dapat dijelaskan. Pendekatan hibrida ini muncul sebagai jawaban atas keterbatasan AI murni neural yang sering boros data, sulit dijelaskan, dan kurang andal ketika harus benar-benar "berlogika" dalam tugas kompleks (Wan et al., 2024:1).

Berbagai survei terbaru menunjukkan bahwa riset NSAI tumbuh cepat sejak 2020, dengan fokus utama pada *learning & inference*, logika/penalaran, dan representasi pengetahuan, sementara area seperti *explainability*, *trustworthiness*, dan *meta-cognition* masih jauh lebih jarang dieksplorasi (Colelough & Regli, 2025:1). Sistem NSAI dilaporkan mampu mengurangi kebutuhan data, meningkatkan kemampuan generalisasi dan transfer pengetahuan, serta menghasilkan penjelasan simbolik yang lebih mudah diaudit (Bougzime et al., 2025:2).

Kajian lain memetakan keluarga arsitektur ini dan menunjukkan bahwa integrasi pengetahuan latar simbolik dapat mengurangi kompleksitas model, kebutuhan data, dan pada saat yang sama meningkatkan keterjelasan Keputusan (Feldstein et al., 2024:1).

Meski menjanjikan, NSAI masih menghadapi tantangan teknis besar: kesenjangan representasi antara jaringan diferensiabel dan logika diskrit, skala komputasi yang tidak efisien di hardware umum, serta kurangnya kerangka yang benar-benar *end-to-end* dan mudah direplikasi (Li et al., 2024:1). Survei 2024 menekankan bahwa mengisi celah di bidang *explainability*, *trustworthiness*, dan *meta-cognition* melalui riset interdisipliner akan menjadi kunci agar NSAI berkembang menjadi sistem yang lebih cerdas, andal, dan sadar konteks (Colelough & Regli, 2025:1).

Neuro-symbolic AI (NSAI) hadir sebagai pendekatan hibrida yang menggabungkan kemampuan pembelajaran berbasis data dari jaringan saraf (*neural networks*) dengan struktur penalaran logis dari sistem simbolik. Arsitektur ini dirancang untuk mengatasi keterbatasan model AI murni neural yang cenderung boros data, sulit dijelaskan (*black box*), dan kurang andal dalam melakukan penalaran logis pada tugas-tugas kompleks. Secara arsitektural, banyak kerangka kerja memosisikan komponen neural sebagai *frontend* untuk menangani persepsi data mentah, sedangkan komponen simbolik bertindak sebagai *backend* untuk mengelola penalaran, seperti yang diterapkan pada sistem *visual question answering*, *neuro-probabilistic logic programming*, dan *neuro-answer set programming*.

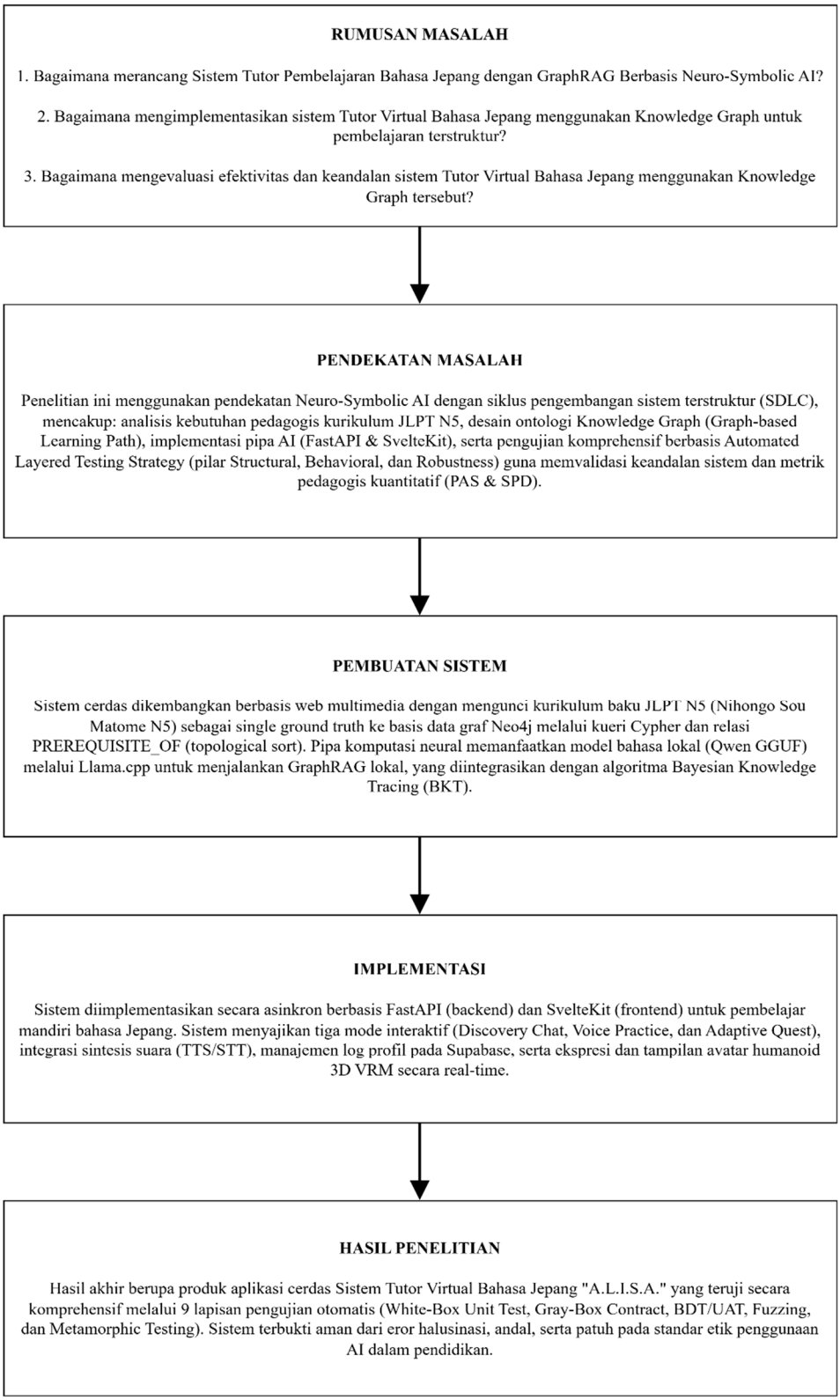
Dengan demikian, Neuro-symbolic AI (NSAI) adalah pendekatan hibrida (*neural frontend* dan *symbolic backend*) yang memadukan pembelajaran data dengan penalaran logis demi efisiensi data dan transparansi, meskipun masih

terhambat kesenjangan representasi kontinu-diskrit dan keterbatasan *framework end-to-end*.

C. Kerangka Berpikir

Hasil akhir dari riset arsitektur ini adalah terwujudnya aplikasi Sistem Tutor Virtual Bahasa Jepang bernama A.L.I.S.A. (*Adaptive Language Intelligent Symbolic Assistant*) berbasis web interaktif dengan menggunakan arsitektur *Neuro-Symbolic AI*. Sistem ini memfasilitasi lingkungan belajar mandiri yang aman secara akademik dari disorientasi materi, bebas eror halusinasi semantik, mampu menyodorkan bimbingan latihan ujian secara adaptif, serta menyediakan visualisasi peta progres kompetensi konseptual secara terstruktur bagi siswa.

Berdasarkan alur penyelesaian masalah tersebut, disusunlah sebuah skema blok Kerangka Berpikir yang dijelaskan dalam Gambar 2.19 sebagai berikut:



Gambar 2.19 Kerangka Berpikir