



PROPOSAL TUGAS AKHIR - IFXXXXXX

Implementasi Agentic Retrieval Augmented Generation AI Inklusif berbasis Model Context Protocol dan Multimodal Inference Engine Menggunakan Arsitektur Gemma dan CLIP Vision Transformer (Studi Kasus Neutrack AI Glove)

Firania Putri Harsanti

NRP 5025221023

Dosen Pembimbing 1

Dini Adni Navastara, S.Kom., M.Sc.

NIP 198510172015042001

Dosen Pembimbing 2

Prof. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D

NIP 198106202005011003

Program Studi S1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknik Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

LEMBAR PENGESAHAN

IMPLEMENTASI AGENTIC AI INKLUSIF BERBASIS MCP DAN MULTIMODAL INFERENCE ENGINE MENGGUNAKAN ARSITEKTUR GEMMA DAN CLIP VISION TRANSFORMER (STUDI KASUS NEUTRACK AI GLOVE)

PROPOSAL PENELITIAN

Diajukan untuk memenuhi salah satu syarat

memperoleh gelar Sarjana Komputer pada

Program Studi S-1 Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Oleh: **FIRANIA PUTRI HARSANTI**

NRP. 5025221023

Disetujui oleh Tim Penguji Proposal Penelitian :

- | | |
|----------------------------------------------------------------|---------------|
| 1. Dini Adni Navastara, S.Kom., M.Sc. | Pembimbing |
| 2. Prof. Ary Mazharuddin Shiddiqi, S.Kom.,
M.Comp.Sc., Ph.D | Ko-pembimbing |
| 3. Nama dan gelar penguji | Penguji |
| 4. Nama dan gelar penguji | Penguji |
| 5. Nama dan gelar penguji | Penguji |

**SURABAYA
Surabaya, 2025**

APPROVAL SHEET

IMPLEMENTATION OF INCLUSIVE AGENTIC RETRIEVAL AUGMENTED GENERATION AI BASED ON MODEL CONTEXT PROTOCOL AND MULTIMODAL INFERENCE ENGINE USING GEMMA ARCHITECTURE AND CLIP VISION TRANSFORMER (CASE STUDY: NEUTRACK AI GLOVE)

FINAL PROJECT PROPOSAL

Submitted to fulfill one of the requirements
for obtaining a degree Informatics Engineering at
Undergraduate Study Program of Informatics
Department of Informatics
Faculty of Intelligent Electrical And Informatics Technology
Institut Teknologi Sepuluh Nopember

Oleh: **FIRANIA PUTRI HARSANTI**
NRP. 5025221023

Approved by Final Project Proposal Examiner Team:

- | | |
|----------------------------------------------------------------|------------|
| 1. Dini Adni Navastara, S.Kom., M.Sc. | Advisor |
| 2. Prof. Ary Mazharuddin Shiddiqi, S.Kom.,
M.Comp.Sc., Ph.D | Co-Advisor |
| 3. Name of Examiner and academic title | Examiner |
| 4. Name of Examiner and academic title | Examiner |
| 5. Name of Examiner and academic title | Examiner |

ABSTRAK

IMPLEMENTASI AGENTIC AI INKLUSIF BERBASIS MCP DAN MULTIMODAL INFERENCE ENGINE MENGGUNAKAN ARSITEKTUR GEMMA DAN CLIP VISION TRANSFORMER (STUDI KASUS NEUTRACK AI GLOVE)

Nama Mahasiswa / NRP	: Firania Putri Harsanti / 5025221023
Departemen	: Teknik Informatika FTEIC - ITS
Dosen Pembimbing I	: Dini Adni Navastara, S.Kom., M.Sc.
Dosen Pembimbing II	: Prof. Ary Mazharuddin Shiddiqi S.Kom., M.Comp.Sc., Ph.D

Abstrak

Penelitian ini mengembangkan NeutraGem, sebuah mesin inferensi multimodal lokal yang terintegrasi dengan Model Context Protocol (MCP) untuk mendukung sistem Agentic AI dalam alat bantu seperti Neutrack, yang ditujukan bagi penyandang tunanetra dan demensia. Solusi ini dirancang agar dapat menjalankan berbagai fungsi secara lokal—seperti *image captioning*, pengenalan wajah, dan respons suara—tanpa bergantung pada cloud, dengan tetap menjaga privasi dan efisiensi.

Proses dimulai dengan pelatihan model menggunakan dataset Flickr8k, di mana gambar dipasangkan dengan deskripsi dalam Bahasa Inggris. Model CLIP digunakan untuk ekstraksi fitur visual, sementara Gemma 2B quantized C++ dipakai untuk generasi teks yang ringan. Hasil pelatihan ini membentuk NeutraGem Inference Engine, yang kemudian disusun dalam ekosistem MCP agar dapat memilih jalur tugas yang tepat secara dinamis. Diagram sistem menunjukkan bahwa gambar yang diambil oleh kamera akan diproses melalui modul inferensi, dan hasil caption-nya dapat dibacakan secara suara melalui VUI (Voice User Interface).

Pengujian dilakukan terhadap waktu inisialisasi model dan agen MCP, kedalaman rekursi logika agent, serta skor akurasi BLEU. Selain itu, uji coba dilakukan pada Raspberry Pi 5 untuk melihat efisiensi penggunaan sumber daya (*RAM, CPU, dan disk*) saat sistem berjalan penuh. Hasil evaluasi menunjukkan bahwa sistem ini mampu memberikan deskripsi yang relevan dengan latensi rendah dan performa yang stabil.

Penelitian ini menyajikan solusi Local Generative Prompting for Agentic AI yang adaptif, inklusif, dan open-source untuk pendampingan berbasis multimodal, terutama di wilayah yang memiliki keterbatasan infrastruktur jaringan.

Kata Kunci: *Agentic AI, Model Context Protocol (MCP), Gemma C++, CLI*

ABSTRACT

IMPLEMENTATION OF INCLUSIVE AGENTIC AI BASED ON MODEL CONTEXT PROTOCOL AND MULTIMODAL INFERENCE ENGINE USING GEMMA ARCHITECTURE AND CLIP VISION TRANSFORMER (CASE STUDY: NEUTRACK AI GLOVE)

Student Name / NRP	:	Firania Putri Harsanti / 5025221023
Department	:	Informatics Engineering FTEIC - ITS
Advisor I	:	Dini Adni Navastara, S.Kom., M.Sc.
Advisor II	:	Prof. Ary Mazharuddin Shiddiqi S.Kom., M.Comp.Sc., Ph.D

Abstract

This study introduces NeutraGem, a local multimodal inference engine integrated with the Model Context Protocol (MCP) to support Agentic AI capabilities in assistive devices such as Neutrack, designed for visually impaired and dementia users. The system enables local execution of multiple tasks—such as image captioning, face recognition, and voice responses—while preserving privacy and ensuring efficient operation without relying on cloud-based infrastructure.

The development process involves training on the Flickr8k dataset, which pairs images with textual captions. CLIP is used for visual feature extraction, while Gemma 2B quantized in C++ handles lightweight natural language generation. The trained NeutraGem Inference Engine is integrated into an MCP-based architecture to dynamically select task routes based on context. As depicted in the system diagram, the camera captures input images, which are processed locally and converted into voice-accessible captions via the Voice User Interface (VUI).

Evaluation includes testing initialization time, logical recursion depth, BLEU score accuracy, and resource utilization (RAM, CPU, disk) on a Raspberry Pi 5. The results indicate that the system delivers semantically relevant captions with low latency and stable runtime.

This research proposes a sustainable, open-source solution through Local Generative Prompting for Agentic AI, offering inclusive and adaptive multimodal assistance in areas with limited connectivity.

Keywords: *Agentic AI, Model Context Protocol (MCP), Gemma C++, CLIP*

DAFTAR ISI

DAFTAR ISI

HALAMAN JUDUL.....	i
LEMBAR PENGESAHAN.....	ii
ABSTRAK.....	iv
DAFTAR ISI.....	v
DAFTAR GAMBAR.....	vi
DAFTAR TABEL.....	vii
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	2
1.3 Batasan Masalah.....	2
1.4 Tujuan Penelitian.....	3
1.5 Manfaat Penelitian.....	3
BAB II TINJAUAN PUSTAKA.....	6
2.1 Penelitian Terkait.....	6
2.2 Dasar Teori.....	10
2.2.1 Face Perceptioning.....	10
2.2.2 Model Context Protocol.....	11
2.2.3 LangGraph dan Acyclic Graph.....	12
2.2.4 CLIP dan Vision Transformer.....	13
2.2.5 Inference Engine dan Model Kuantisasi.....	14
BAB III METODOLOGI PENELITIAN.....	16
3.1 Training Model.....	16
3.2 Inference Engine.....	17
3.3 Integrasi Model Context Protocol.....	18
3.4 Pengujian dan Evaluasi.....	20
DAFTAR PUSTAKA.....	53

DAFTAR GAMBAR/GRAFIK/DIAGRAM

Gambar 2.1 Model Face Perception oleh Bruce dan Young.....	17
Gambar 2.2 Arsitektur Client dan Server Model Context Protocol.....	18
Gambar 2.3 Acyclic Graph dari LangGraph.....	19
Gambar 2.4 CLIP Vision Model.....	20
Gambar 2.5 Kuantisasi Model dengan Inference Engine.....	21
Gambar 3.1 Alur Training Model Image Captioning.....	24
Gambar 3.2 Alur Inference Engine.....	25
Gambar 3.3 Alur Model Context Protocol Proxy Manager.....	26
Gambar 3.4 Alur Tahap Pengujian dengan Perintah Suara.....	28
Gambar 3.5 Konfigurasi Hardware Neutrack AI Glove.....	29

DAFTAR TABEL

Tabel 2.1 Penelitian Terkait.....	16
Tabel 3.1 Pseudocode Pelatihan Model Berbasis Gemma 2 dan CLIP VIT.....	32
Tabel 3.2 Pseudocode Inference Engine.....	34
Tabel 3.3 Pseudocode Embedding Pipeline Retrieval Augmented Generation.....	36
Tabel 3.4 Pseudocode Retrieval with Rerank dan Generate Response Berbasis Vektor.....	37
Tabel 3.5 Pseudocode Inisialisasi Model LLM Lokal untuk Inference RAG.....	39
Tabel 3.6 Pseudocode CRUD AI Agent dalam Ekosistem MCP.....	40
Tabel 3.7 Pseudocode Model Context Protocol Proxy Manager.....	42
Tabel 3.8 Pseudocode Monitoring Status Tools MCP.....	44
Tabel 3.9 Pseudocode Pengujian MCP dan AI Agent.....	47
Tabel 3.10 Pseudocode Pengujian BLEU Score Image Captioning.....	47

DAFTAR SIMBOL (jika ada)

DAFTAR SINGKATAN (jika ada)

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Kesehatan global saat ini tengah menghadapi tantangan besar seiring dengan meningkatnya jumlah kasus gangguan penglihatan dan demensia di seluruh dunia. Berdasarkan data dari World Health Organization (WHO), diperkirakan terdapat 2,2 miliar kasus gangguan penglihatan, dengan 1,1 miliar di antaranya mengalami kebutaan total. Sementara itu, jumlah kasus demensia terus meningkat, dengan lebih dari 5,5 juta kasus yang tercatat saat ini dan proyeksi mencapai 10 juta kasus baru setiap tahunnya.

Tunanetra merupakan salah satu jenis kebutaan khusus yang ditandai dengan hilangnya fungsi indra visual. Individu dengan kondisi ini sering kali kehilangan akses terhadap informasi penting dari lingkungan sekitar yang biasanya dapat diperoleh dengan mudah oleh individu yang memiliki penglihatan normal. Keterbatasan akses terhadap informasi visual ini menimbulkan tantangan besar yang dapat menghambat mobilitas, interaksi dengan lingkungan, hingga keselamatan pribadi, terutama dalam hal membaca teks, mengenali rintangan, dan bernavigasi. Selanjutnya, Saleh, Ali, dan Ezzat (2022) merancang kacamata cerdas berbasis pengenalan wajah untuk penderita Alzheimer menggunakan algoritma Viola-Jones dan ekstraksi fitur Local Binary Pattern. Meskipun akurasi K-NN mencapai 93,36%, sistem ini masih terbatas pada pengenalan wajah tunggal tanpa reasoning agent atau data multimodal.

Sebagai upaya untuk menjawab tantangan tersebut, telah dikembangkan sistem bantu berbasis Agentic AI yang difokuskan untuk mendukung penyandang tunanetra dan individu dengan demensia tahap awal. Salah satu pendekatan efisien ditunjukkan oleh FlexGen (Ying et al., 2023), yang memungkinkan inferensi Large Language Models (LLMs) berukuran besar hanya dengan GPU tunggal, melalui optimasi memori dan pemrograman linear untuk pengaturan akses tensor. Solusi ini sangat relevan dalam pengembangan sistem lokal (on-premise) berbasis edge device seperti Raspberry Pi.

Pendekatan pengujian performa protokol Model Context Protocol (MCP) diperkenalkan secara komprehensif oleh Gao et al. (2025) dalam MCP-RADAR, sebuah benchmark yang menilai efisiensi dan ketepatan penggunaan tool oleh LLM dalam berbagai domain tugas. Penilaian dilakukan secara objektif melalui lima dimensi dan menampilkan perbedaan signifikan antara berbagai model, menjadikannya panduan penting dalam membangun sistem agentik berbasis tool.

Dalam ranah sistem penerjemahan, Wang dan Duan (2024) mengusulkan integrasi LangGraph sebagai kerangka modular untuk menyusun alur kerja agent secara dinamis dalam LLM-based machine translation. Arsitektur ini memungkinkan pengembangan agen multibahasa yang efisien dan scalable, sejalan dengan kebutuhan sistem bantu berbasis interaksi dialog.

Optimalisasi inferensi juga ditunjukkan oleh Seesaw, sebuah engine LLM yang memanfaatkan strategi re-sharding dinamis antara tahap prefill dan decode, meningkatkan throughput hingga $1.78\times$ dibanding vLLM (Park et al., 2025). Selain itu, Shiddiqi et al. (2023) melalui RAViS Framework membuktikan bahwa algoritma deteksi objek berbasis deep learning dapat dijalankan secara efisien pada perangkat terbatas seperti Raspberry Pi, dengan pemantauan sumber daya secara real-time.

Dalam konteks perbaikan kode otomatis, Wang dan Duan (2025) menunjukkan bahwa sistem LangGraph yang digabungkan dengan GLM-4-Flash dan ChromaDB dapat memperbaiki fungsi Python secara iteratif berdasarkan error runtime, membuktikan efektivitas pendekatan agentic dalam pengembangan perangkat lunak adaptif. Studi oleh SuperAGI (2025) menyoroti perbandingan antara MCP dan custom integrations dalam membangun sistem AI modular. Hasil menunjukkan bahwa MCP lebih unggul dalam hal interoperabilitas lintas framework dan efisiensi kolaboratif antar agent dan tools. Terakhir, Zongxi Liu dan Hongyang Du (2025) mengusulkan Internet of Experts (IoX) berbasis MCP untuk memperkaya LLM

dengan kemampuan reasoning terhadap kondisi lingkungan nirkabel. Dengan mengakses expert models untuk mendeteksi atribut fisik seperti LoS, Doppler, dan fading, agen LLM dapat merespon secara kontekstual tanpa pelatihan ulang, meningkatkan akurasi klasifikasi hingga 40–50% dan membuka jalur baru bagi integrasi AI dalam sistem komunikasi adaptif.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam penelitian ini adalah sebagai berikut:

1. Bagaimana pengembangan *Model Context Protocol* (MCP) dapat memperkuat kinerja agentic AI dalam meningkatkan kinerja *image captioning* maupun fungsi asistensi lainnya pada Neutrack?
2. Bagaimana *inference engine* dapat mempercepat LLM secara lokal di sistem benam untuk menjaga privasi/keamanan data serta efisiensi anggaran mengingat tidak dibutuhkannya running LLM secara cloud?
3. Bagaimana hasil evaluasi dari segi waktu, uji SUS, resource monitoring (RAM, CPU, dan Disk) setelah implementasi?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam penelitian ini memiliki beberapa batasan, diantaranya sebagai berikut: implementasi Agentic AI dibatasi pada pemrosesan data multimodal berupa citra dan suara menggunakan CLIP Vision Transformer dan perintah suara sederhana. Sistem dikembangkan dengan arsitektur Gemma C++ dan dioptimalkan untuk pada Neutrack AI Glove. Studi kasus difokuskan pada pengguna tunanetra dalam skenario navigasi dasar dan pengenalan objek.

1.4 Tujuan

Tujuan dari penelitian ini diantaranya sebagai berikut:

1. MCP dapat memperkuat kinerja agentic AI dalam meningkatkan kinerja *image captioning* maupun fungsi asistensi lainnya pada Neutrack.
2. *Inference engine* dapat mempercepat LLM secara lokal di sistem benam untuk menjaga privasi/keamanan data serta efisiensi anggaran mengingat tidak dibutuhkannya running LLM secara cloud.
3. Hasil evaluasi dari segi waktu, *resource monitoring* (RAM, CPU, dan Disk) setelah implementasi dapat memberikan pengalaman pengguna yang lebih baik bagi penyandang tunanetra maupun demensia.

1.5 Manfaat

Manfaat yang diharapkan dari pembuatan penelitian ini adalah membuat *agentic AI* dengan mengimplementasikan *inference engine* berbasis Gemma C++ dan *CLIP Vision Model* dalam lingkungan pada Neutrack AI Glove sebagai asisten virtual penyandang tunanetra.

BAB 2 TINJAUAN PUSTAKA

Pada bab ini, berisikan pembahasan literatur penelitian sebelumnya yang relevan dengan penelitian yang dikerjakan dan akan dijelaskan mengenai dasar teori yang digunakan dalam pembuatan penelitian ini.

2.1 Penelitian Terkait

Dalam proses penelitian ini, informasi dari berbagai penelitian sebelumnya dikumpulkan untuk digunakan sebagai pembanding dan referensi saat melakukan pengujian, yang tersaji dalam Tabel 2.1. Sumber informasi diperoleh dari jurnal, buku, dokumentasi resmi, dan skripsi yang relevan dengan judul Penelitian ini guna memperoleh dasar teoritis yang kuat.

Salah satu referensi penting adalah penelitian oleh Sayifa dan Pandian (2023) yang mengembangkan kerangka kerja *real-time image captioning* untuk penyandang tunanetra, menggunakan kombinasi model VGG16-LSTM dan ResNet-50, dengan akurasi tertinggi 83,88% pada model VGG-16. Penelitian ini menunjukkan efektivitas visi komputer tetapi belum dioptimalkan untuk perangkat edge tanpa GPU tambahan dan tidak mendukung input multimodal seperti suara.

Lalu, penelitian selanjutnya oleh Saleh, Ali, dan Ezzat (2022) merancang kacamata cerdas berbasis pengenalan wajah untuk penderita Alzheimer menggunakan algoritma Viola-Jones dan ekstraksi fitur Local Binary Pattern. Meskipun akurasi K-NN mencapai 93,36%, sistem ini masih terbatas pada pengenalan wajah tunggal tanpa adanya reasoning agent atau pemrosesan data multimodal.

Penelitian berikutnya oleh Ying et al. (2023), berjudul *FlexGen: High-Throughput Generative Inference of LLMs with a Single GPU*, menanggapi tantangan utama dalam inferensi model bahasa besar (Large Language Models/LLMs), terutama kebutuhan komputasi tinggi yang biasanya memerlukan banyak GPU kelas atas. FlexGen hadir sebagai solusi untuk menjalankan LLM berukuran besar pada perangkat dengan sumber daya terbatas, seperti GPU tunggal 16GB, melalui pendekatan offloading dan optimasi kuantisasi 4-bit. Sistem ini secara cerdas menggabungkan memori GPU, CPU, dan disk dengan menyelesaikan masalah pemrograman linear untuk mengatur alur akses tensor yang efisien. Dalam pengujinya, FlexGen mampu mencapai throughput hingga 1 token/detik untuk model OPT-175B, serta mampu menjalankan model 30B dalam benchmark HELM pada tujuh skenario dalam waktu hanya 21 jam. Dengan desain fleksibel dan performa yang kompetitif, FlexGen menjadi salah satu platform inferensi LLM paling hemat sumber daya yang relevan untuk kebutuhan edge computing dan skenario deployment di luar lingkungan server konvensional.

Selanjutnya, penelitian oleh Gao et al. (2025) memperkenalkan *MCP-RADAR*, sebuah benchmark pertama yang dirancang untuk mengevaluasi performa model LLM dalam konteks *Model Context Protocol (MCP)*. Berbeda dengan benchmark konvensional yang mengandalkan penilaian subjektif atau metrik biner sederhana, MCP-RADAR menawarkan pendekatan evaluasi objektif melalui lima dimensi kunci: akurasi jawaban, efisiensi pemilihan tool, efisiensi penggunaan sumber daya komputasi, ketepatan konstruksi parameter, dan kecepatan eksekusi. Studi ini menguji berbagai model LLM komersial dan open-source dalam berbagai domain tugas seperti rekayasa perangkat lunak, penalaran matematika, dan pemecahan masalah umum. Hasilnya menunjukkan adanya profil kemampuan yang berbeda-beda antar model, serta trade-off nyata antara akurasi, efisiensi, dan kecepatan, yang menantang cara lama dalam melakukan perankingan model. MCP-RADAR tidak hanya membantu dalam menilai kinerja model dalam interaksi dengan tool, tetapi juga memberikan panduan praktis bagi pengembang untuk meningkatkan kompatibilitas dan efektivitas tool dalam ekosistem agen

LLM. Fokus studi ini memang pada MCP sebagai protokol standar, namun pendekatan metodologinya dapat diterapkan pada kerangka integrasi tool lainnya.

Kemudian, penelitian oleh Wang dan Duan (2024) melalui karya berjudul *Agent AI with LangGraph* membahas integrasi kerangka kerja LangGraph dalam sistem agen penerjemahan mesin berbasis LLM. Penelitian ini menunjukkan bagaimana LangGraph—yang dibangun di atas LangChain—memungkinkan pengembangan alur kerja agen secara modular dan otomatis. Dengan memanfaatkan agen-agen spesifik untuk bahasa seperti *TranslateEnAgent*, *TranslateFrenchAgent*, dan *TranslateJpAgent*, sistem ini mampu melakukan penerjemahan antar bahasa dengan mempertahankan konteks dialog secara dinamis. Model sequence-to-sequence yang digunakan dilengkapi dengan attention mechanism dan dilatih pada pasangan data Inggris-Prancis. Eksperimen menunjukkan peningkatan akurasi terjemahan rata-rata sebesar 8,3% dibandingkan baseline GPT-4o solo, dengan latensi proses terjemahan kurang dari 1,2 detik per kalimat. Evaluasi dilakukan menggunakan metrik BLEU1 hingga BLEU4 terhadap 300 pasangan kalimat. Hasil menunjukkan efektivitas pendekatan LangGraph dalam menyusun agen penerjemah berbasis LLM yang modular, efisien, dan dapat diskalakan.

Penelitian berikutnya oleh Su et al. (2024) memperkenalkan *Seesaw*, engine inferensi LLM yang dirancang untuk throughput tinggi melalui pendekatan *dynamic model re-sharding*. Penulis menyoroti bahwa dua tahap utama dalam inferensi LLM—prefill dan decode—memiliki karakteristik komputasi yang sangat berbeda, sehingga strategi paralelisasi statis tidak cukup optimal. Seesaw secara adaptif mengonfigurasi ulang strategi paralelisasi antara dua tahap tersebut, memanfaatkan tiered KV cache buffering dan transition-minimizing scheduling untuk mengurangi overhead saat berpindah antar tahap. Inovasi ini memungkinkan pemrosesan batch yang lebih besar pada tahap decode, sekaligus menghindari bottleneck komunikasi pada tahap prefill. Evaluasi kuantitatif menunjukkan bahwa Seesaw meningkatkan throughput hingga $1.78\times$ (rata-rata $1.36\times$) dibandingkan vLLM, mesin inferensi LLM terbaik saat ini. Kontribusi lainnya termasuk analisis performa per strategi paralelisasi dan efisiensi pemindahan bobot serta cache antar GPU dan CPU. Penelitian ini menekankan pentingnya pendekatan dinamis untuk inferensi LLM offline berskala besar, terutama pada lingkungan dengan keterbatasan sumber daya. Seesaw membuka arah baru untuk optimasi throughput LLM dalam aplikasi industri berbasis informasi seperti ekstraksi data, query basis data, dan sistem reasoning berbasis grafik.

Penelitian yang juga relevan adalah oleh Shiddiqi et al. (2025), "*Resource-Aware Video Streaming (RAViS) Framework for Object Detection System Using Deep Learning Algorithm*" merupakan kontribusi penting dalam bidang edge computing dan computer vision, khususnya dalam konteks efisiensi penggunaan sumber daya pada perangkat terbatas seperti Raspberry Pi. Penelitian ini menawarkan solusi inovatif melalui pengembangan kerangka kerja RAViS yang mampu menjalankan sistem deteksi objek berbasis deep learning secara adaptif, dengan tetap mempertahankan akurasi deteksi tinggi meskipun menghadapi keterbatasan CPU, RAM, dan penyimpanan. Framework ini memantau dan menyesuaikan parameter operasional secara real-time untuk menjaga kinerja sistem deteksi menggunakan algoritma YOLO. Melalui serangkaian eksperimen dalam berbagai skenario, penulis membuktikan efektivitas RAViS dalam menurunkan konsumsi sumber daya secara signifikan tanpa mengorbankan kualitas deteksi. Dengan penggabungan konsep granularitas algoritmik dan pendekatan modular yang efisien, RAViS tidak hanya relevan untuk sistem pengawasan dan manajemen energi berbasis visi komputer, tetapi juga membuka peluang aplikasi luas pada perangkat IoT dan sistem pintar lainnya yang menuntut efisiensi tinggi.

Paper selanjutnya yang berjudul "*Empirical Research on Utilizing LLM-based Agents for Automated Bug Fixing via LangGraph*" memperkenalkan sebuah sistem otomatis untuk perbaikan kode berbasis Large Language Model (LLM) yang terintegrasi dalam kerangka LangGraph. Sistem ini menggabungkan tiga komponen utama—LangGraph sebagai pengatur

alur tugas, GLM-4-Flash untuk pemahaman dan generasi kode, serta ChromaDB sebagai penyimpanan memori kontekstual berbasis vektor. Prosesnya terdiri dari empat tahap: *code generation*, *code execution*, *code repair*, dan *code update*, yang dijalankan secara iteratif hingga kode bebas dari bug. Dengan kemampuan reasoning semantik dan pelacakan status global, sistem ini mampu memperbaiki bug secara otomatis berdasarkan informasi historis dan pola kesalahan yang terekam. Eksperimen menunjukkan bahwa sistem ini efektif dalam menghasilkan dan memperbaiki fungsi Python, seperti kalkulasi luas segitiga dan pembagian angka dengan pengecekan pembagi nol. Framework ini memberikan pendekatan baru yang andal dalam otomatisasi rekayasa perangkat lunak dan pengurangan intervensi manual. Secara keseluruhan, penelitian ini memperluas pemanfaatan LLM dalam pengembangan perangkat lunak cerdas yang adaptif dan efisien.

Paper selanjutnya yang berjudul Paper “*MCP vs Custom Integrations: Comparing the Efficiency and Scalability of Model Context Protocol Servers in AI Development*” (SuperAGI, 2025) membahas perbandingan antara penggunaan Model Context Protocol (MCP) dan pendekatan *custom integrations* dalam pengembangan sistem AI, khususnya dalam hal efisiensi, skalabilitas, dan pengelolaan konteks model. Studi ini menunjukkan bahwa MCP, sebagai protokol standar yang dirancang untuk mengelola alur kerja multi-model dan multi-agent secara modular, mampu menyederhanakan integrasi dan mengurangi kompleksitas teknis yang sering terjadi pada pendekatan integrasi manual. Dengan fitur seperti komunikasi stateful, interoperabilitas lintas framework, dan dukungan plug-and-play untuk berbagai LLM dan tools, MCP menawarkan keunggulan dalam pengembangan cepat dan kolaboratif. Di sisi lain, custom integrations sering kali lebih fleksibel dan dapat dioptimalkan secara spesifik untuk kasus penggunaan tertentu, tetapi membutuhkan waktu pengembangan yang lebih lama dan rentan terhadap kesalahan saat skalabilitas ditingkatkan. Melalui berbagai studi kasus dan eksperimen performa, paper ini menyimpulkan bahwa MCP lebih cocok untuk sistem AI berskala besar dan kolaboratif yang menuntut modularitas dan pengelolaan konteks dinamis, sedangkan custom integrations tetap relevan untuk solusi yang sangat khusus atau ringan.

Penelitian selanjutnya yang berjudul “*Model Context Protocol-based Internet of Experts for Wireless Environment-aware LLM Agents*” oleh Zongxi Liu dan Hongyang Du (2025), menawarkan pendekatan revolusioner untuk menghadirkan kesadaran lingkungan nirkabel pada agen LLM melalui integrasi dengan Model Context Protocol (MCP). Dalam kerangka ini, LLM tidak perlu dilatih ulang atau diprogram ulang untuk memahami kondisi saluran nirkabel, melainkan secara dinamis dapat mengakses serangkaian expert models yang dirancang khusus untuk mendeteksi atribut-atribut fisik seperti kondisi line-of-sight (LoS), efek Doppler, hingga fading. Sistem ini disebut Internet of Experts (IoX), di mana masing-masing model ahli dapat di-query melalui antarmuka JSON-RPC standar yang ditetapkan MCP, dan dijalankan secara modular di edge maupun cloud server. Hasil eksperimen menunjukkan bahwa dengan pendekatan ini, akurasi klasifikasi kondisi saluran meningkat signifikan hingga 40–50% dibandingkan baseline LLM tanpa bantuan expert. Arsitektur ini juga memastikan interpretabilitas dan efisiensi waktu nyata, tanpa perlu mengubah bobot model LLM. Secara keseluruhan, penelitian ini membuka jalur baru bagi integrasi LLM ke dalam sistem komunikasi nirkabel cerdas, serta memperluas aplikasi MCP untuk membangun agen yang adaptif terhadap kondisi lingkungan fisik yang dinamis dan kompleks.

Tabel 2.1 Penelitian Terkait

Judul	Dataset	Metode	Analisis Gap
<i>A real-time image captioning framework using computer vision to help the visually impaired</i> (Sayifa & Pandian, 2023)	MS COCO, Flickr8k, dan dataset pribadi	VGG16-LSTM, ResNet-50. Performa terbaik: akurasi VGG-16 mencapai 83,88%	Belum dioptimalkan untuk edge device tanpa GPU eksternal dan belum mendukung input suara (non-multimodal).
<i>Face Recognition Based Smart Glass for Alzheimer's Patients</i> (Sayifa & Pandian, 2023)	Dataset wajah pasien Alzheimer (pribadi)	Viola-Jones + Local Binary Pattern, klasifikasi SVM dan K-NN. Performa terbaik: akurasi K-NN sebesar 93,36%	Fokus tunggal pada pengenalan wajah; tidak menggunakan data multimodal atau reasoning agent adaptif.
<i>FlexGen: High-Throughput Generative Inference of LLMs with a Single GPU</i> (Ying et al., 2023)	Priority: OPT-175B pada GPU 16 GB	Throughput 1 token/s pada batch 144; quantized 4-bit	GPU-server, bukan edge device
<i>MCP-RADAR: A Multi-Dimensional Benchmark for Evaluating Tool Use Capabilities in Large Language Models</i> (Gao et al., 2025)	MCP-RADA R benchmark: tugas software engineering, mathematical reasoning, dan problem-solving	Evaluasi 6 LLM dalam 5 dimensi performa: - GPT-4: akurasi 93.7%, efisiensi pemilihan tool 94.3%, pemanfaatan resource 72.4%, akurasi parameter 90.2%, eksekusi sukses 95.6% - Claude 3: akurasi 91.2%, efisiensi 92.7%, resource 70.8%, parameter 88.4%, eksekusi 94.1% - Gemini 1.5 Pro: akurasi 90.8%, efisiensi 89.6%, resource 68.2%	Fokus pada evaluasi tool-use berbasis protokol MCP dalam konteks LLM agents; belum mencakup sistem sensor nyata, agen fisik, atau deployment edge secara langsung.
<i>Agent AI with LangGraph: A Modular Framework for Enhancing Machine Translation Using Large Language Models</i> (Wang & Duan, 2024)	Multi-agent MT (Inggris→ Prancis, Jepang); LLM: GPT-4o	Akurasi terjemahan meningkat rata-rata +8.3% dibanding baseline GPT-4o solo; Latency pipeline agent: <1.2s per sentence.	Fokus pada penerjemahan; belum diuji dalam konteks multi-modal atau penggunaan sensor.
<i>Seesaw: High-throughput LLM Inference via Model Re-sharding</i> (Su et al., 2024)	Model distribusi LLM di multi-GPU	Throughput +1.78× (avg. +1.36×) vs vLLM	Tidak mencakup integrasi dengan LLM atau MCP, sehingga aplikasinya dalam sistem AI otonom belum diuji secara langsung.

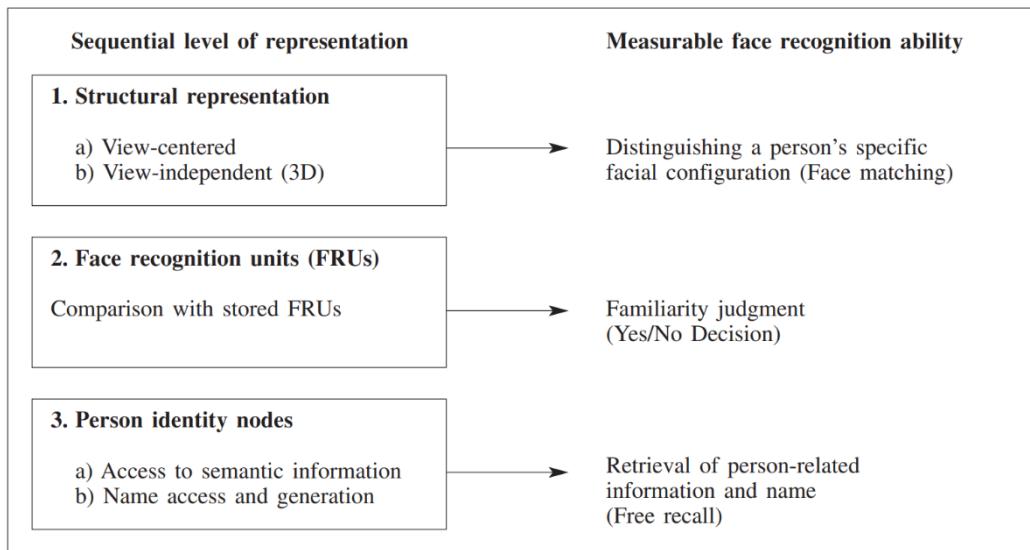
<i>Resource-aware video streaming (RAViS) framework for object detection system using deep learning algorithm</i> (Shiddiqi et al., 2025)	Simulasi video streaming dari kamera pengawas (dataset internal/sintetik)	- Tanpa RAViS (Scenario 2): CPU 84.3%, RAM 36.7%, Storage 57.9%, waktu deteksi rata-rata: 8210s (136 min) - Dengan RAViS (Scenario 3): CPU turun ke 47.9% (\downarrow 36%), RAM naik ke 42.4% namun lebih stabil, Storage turun ke 44.4% (\downarrow 13.5%), deteksi hanya butuh 7.07s rata-rata - Akurasi deteksi: 99.21%, dengan 43.65% deteksi penuh, 56.34% deteksi parsial	Belum mendukung multimodalitas seperti suara atau sensor tambahan. Fokus hanya pada optimasi resource, belum terintegrasi dengan sistem berbasis agen atau AI interaktif.
<i>Empirical Research on Utilizing LLM-based Agents for Automated Bug Fixing via LangGraph</i> (Wang & Duan, 2025)	API-based debugging tasks; LLM: GLM4 Flash + ChromaDB	Tingkat keberhasilan bug fix 78%, rata-rata iterasi debugging: 3.4 langkah, kecepatan per iterasi: $<$ 2s, tool recall: 85%.	Belum disertai metrik resource (CPU/RAM/latency) pada skala besar atau deployment edge.
<i>MCP vs Custom Integrations: Comparing the Efficiency and Scalability of Model Context Protocol Servers in AI Development</i> (SuperAGI, 2025)	Benchmark internal (SuperAGI)	\sim 2.500 req/s throughput (60 % lebih tinggi dari custom), latency \sim 50 ms (30 % lebih rendah), CPU turun 15 %, RAM turun 25%	Evaluasi berbasis server, bukan dalam sistem agent lengkap atau sensor nyata
<i>Model Context Protocol-based Internet of Experts for Wireless Environment-aware LLM Agents</i> (Fei et al., 2025)	Synthetic wireless channel data: LoS, Doppler, Rayleigh, Rician	Akurasi prediksi atribut environment: tanpa MCP 45–59%, dengan MCP 95.5–98.1% across LLMs seperti ChatGPT-3.5, 4, DeepSeek	Belum mencakup pengukuran latency, throughput, atau penggunaan resource; hanya fokus pada akurasi klasifikasi

2.2 Dasar Teori

Sub Bab ini menguraikan tentang landasan teori yang digunakan dalam penelitian ini. Landasan teori yang dibahas meliputi teknik pengumpulan data, teknik pemodelan topik dan perhitungan kinerja untuk mengevaluasi *agentic* AI dengan mengimplementasikan *inference engine* berbasis Gemma C++ dan *CLIP Vision Model* dalam di lingkungan *edge computing* pada Neutrack AI Glove sebagai asisten virtual penyandang tunanetra.

2.2.1 Tantangan bagi Penyandang Tunanetra dan Demensia

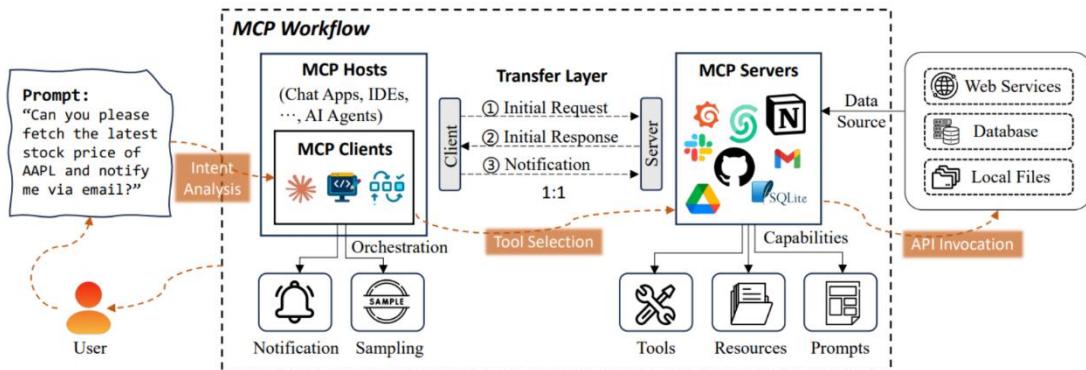
Gangguan penglihatan disebabkan oleh hilangnya ketajaman penglihatan, dimana mata tidak dapat melihat objek sejelas biasanya. Bisa juga disebabkan oleh hilangnya lapang pandang, dimana mata tidak dapat melihat area seluas biasanya tanpa menggerakkan mata atau memutar kepala. *World Health Organization* mendefinisikan “penglihatan rendah/*low vision*” sebagai ketajaman penglihatan antara 20/70 dan 20/400, atau bidang penglihatan 20 derajat atau kurang. “Kebutaan/Tunanetra” didefinisikan sebagai ketajaman penglihatan yang lebih buruk dari 20/400, atau bidang penglihatan 10 derajat atau kurang.



Gambar 2.1 Model Face Perception oleh Bruce dan Young

Demensia merupakan istilah umum yang digunakan untuk menggambarkan penurunan kemampuan kognitif secara signifikan yang mengganggu aktivitas hidup sehari-hari seseorang. Penyakit Alzheimer (AD) adalah jenis demensia yang paling umum, terhitung setidaknya dua pertiga kasus terjadi pada individu berusia 65 tahun ke atas. Model pengenalan wajah (*face perception*) oleh Bruce dan Young (1986) memberikan kerangka kerja untuk memahami bagaimana individu memproses dan mengenali wajah. Menurut model ini, nama adalah informasi yang unik dan tidak memiliki hubungan semantik langsung dengan orang tersebut, hal ini berbeda dengan informasi semantik lainnya yang memiliki jaringan asosiasi yang lebih kuat di otak. *Alzheimer's Disease* (AD) awalnya memengaruhi Level 3, dimana pengidap gejala demensia kesulitan mengingat informasi tentang seseorang misalnya, “dia adalah tetangga saya,” kemudian memengaruhi Level 2, dimana individu mulai kesulitan mengenali wajah yang seharusnya familiar, seperti wajah keluarga atau teman, hingga level 1 dimana mengingat nama adalah tahap yang paling sulit karena memerlukan pemrosesan yang berhasil di semua tahap sebelumnya. Pada demensia, kegagalan di Level 2 dan 3 membuat *recall* nama hampir mustahil. (Werheid et al., 2007)

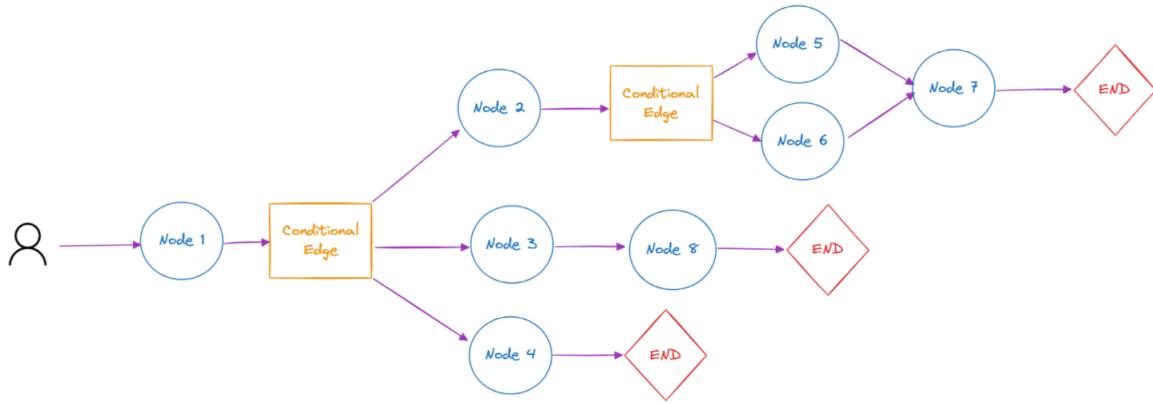
2.2.2 Agentic AI berbasis *Model Context Protocol* (MCP), LangGraph, dan Pydantic



Gambar 2.2 Arsitektur Client dan Server Model Context Protocol

Model Context Protocol (MCP) memperkenalkan standar universal untuk menghubungkan LLM dengan sistem eksternal, menawarkan solusi yang terstruktur untuk integrasi sistem (Anthropic, 2024). Sejak didirikan oleh Anthropic, para pengembang telah membuat ratusan server MCP, menjadikannya kerangka kerja kunci untuk integrasi AI. Protokol ini menyediakan arsitektur yang fleksibel dengan peraturan tertentu di mana data dan kemampuan komputasi dapat diekspos melalui server terstandarisasi, sementara aplikasi AI terhubung sebagai klien untuk mengakses sumber daya tersebut. McGuinness memberikan analisis mendalam tentang pendekatan arsitektural ini dan implikasinya secara praktis. Protokol ini telah diadopsi secara luas oleh perusahaan seperti Block dan Apollo yang mengimplementasikan MCP dalam produksi, serta platform pengembangan seperti Zed, Replit, Codeium, dan Sourcegraph yang mengintegrasikannya ke dalam alur kerja mereka. Anthropic telah mempercepat adopsi dengan menyediakan server MCP bawaan untuk sistem enterprise populer seperti Google Drive, Slack, GitHub, Git, Postgres, dan Puppeteer. Meskipun MCP menunjukkan potensi sebagai standar untuk interaksi sistematis antara pemrosesan *natural language* informal dan sistem komputasi formal, kesuksesannya bergantung pada dukungan industri serta solusi untuk keamanan, skalabilitas, dan kompatibilitas lintas platform (McGuinness, 2024).

Agentic AI merujuk pada sistem kecerdasan buatan yang memiliki kemampuan otonom untuk memahami lingkungan, membuat keputusan, dan menjalankan tindakan secara independen tanpa intervensi manusia terus-menerus. Salah satu tantangan utama dalam pengembangan *Agentic AI* adalah kemampuan untuk berinteraksi dan berintegrasi dengan sistem atau alat eksternal secara aman dan efisien. Di sinilah *Model Context Protocol* (MCP) memainkan peran penting. MCP menyediakan protokol standar yang memungkinkan *Large Language Models* (LLMs) untuk menemukan, mengakses, dan menggunakan berbagai alat komputasi secara dinamis melalui antarmuka universal seperti JSON over HTTP atau stdio (McGuinness, 2024)



Gambar 2.3 Acyclic Graph dari LangGraph

Selain MCP, pengembangan Agentic AI juga didukung oleh LangGraph, sebuah framework berbasis Python yang memungkinkan pembangunan agen AI berbasis graf. LangGraph menggunakan struktur directed acyclic graph (DAG) untuk mengatur alur pemrosesan AI secara modular dan transparan. Hal ini memungkinkan LLM untuk menjalankan tugas yang kompleks secara terstruktur melalui simpul-simpul (nodes) yang dapat dikonfigurasi sebagai fungsi atau pemanggilan API (LangChain, 2024). LangGraph dirancang agar kompatibel dengan ekosistem LangChain dan sangat cocok untuk digunakan dalam pembangunan sistem multi-agent, reasoning loop, dan otomatisasi proses berbasis instruksi.

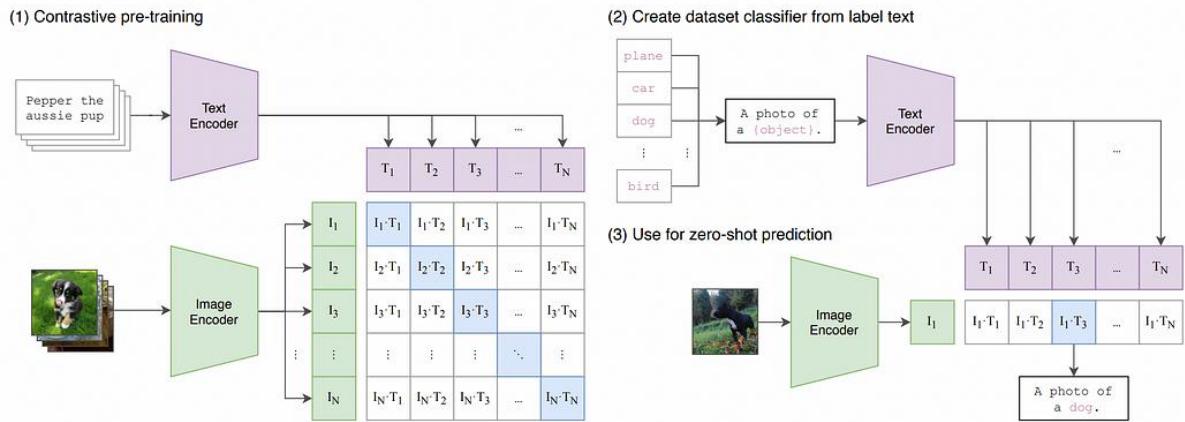
Di sisi lain, Pydantic memainkan peran penting dalam validasi dan serialisasi data dalam sistem Agentic AI. Pydantic memanfaatkan fitur typing dari Python untuk memastikan bahwa struktur data yang dikirim dan diterima antar modul AI, termasuk dalam pemanggilan MCP atau pengolahan LangGraph, tetap konsisten dan aman. Dengan mendefinisikan skema data yang ketat, Pydantic membantu mencegah kesalahan pemrosesan dan meningkatkan reliabilitas sistem AI secara keseluruhan, terutama saat menangani input/output dalam skenario yang kompleks dan berskala besar (Samuel, 2023).

Dengan mengintegrasikan MCP sebagai protokol komunikasi, LangGraph sebagai pengelola alur kerja modular, dan Pydantic sebagai validator data, pengembangan Agentic AI dapat dilakukan secara lebih robust, scalable, dan aman, sekaligus memungkinkan LLM untuk menjadi komponen aktif dalam ekosistem perangkat lunak yang lebih luas

2.2.3 Image Captioning berbasis CLIP Vision Model

CLIP (*Contrastive Language–Image Pretraining*) merupakan model *vision-language* yang terdiri atas dua komponen utama, yaitu *image encoder* dan *text encoder*. Kedua komponen ini dilatih secara bersamaan untuk menghasilkan representasi vektor (*embedding*) dari gambar dan teks dalam ruang embedding yang sama. Hal ini memungkinkan CLIP untuk menilai kesesuaian antara teks dan gambar tanpa memerlukan pelabelan eksplisit selama inferensi, dikenal sebagai *zero-shot learning* (Radford et al., 2021). Komponen *text encoder* pada CLIP menggunakan arsitektur Transformer (Vaswani et al., 2017) dengan konfigurasi 12 lapisan, lebar 512, dan 8 attention heads. Teks mentah diproses melalui mekanisme enkoding pasangan byte (*byte pair encoding/BPE*) dengan ukuran kosa kata (*vocabulary size*) sebesar 49.152 (Sennrich et al., 2016). Panjang maksimum urutan teks dibatasi hingga 76 token, dan setiap token diberikan *positional encoding* sebelum dimasukkan ke dalam encoder teks untuk mendapatkan representasi semantiknya.

CLIP menyediakan beberapa pilihan arsitektur untuk *image encoder*, termasuk jaringan ResNet (He et al., 2016) dan Vision Transformer (ViT) (Dosovitskiy et al., 2020). Mengingat performa superior yang ditunjukkan oleh ViT dalam berbagai penelitian terkini (Zhai et al., 2021), maka pembahasan dalam tulisan ini difokuskan pada penggunaan *image encoder* berbasis Transformer. Dalam arsitektur ViT, gambar dipecah menjadi sejumlah *patch*, yang kemudian dikonversi menjadi token-token vektor. Setiap token diberi tambahan *positional encoding* untuk menjaga informasi lokasi spasial. Selanjutnya, hasil ekstraksi fitur dari seluruh *patch* melalui proses *global pooling*, sehingga menghasilkan satu vektor representatif tunggal untuk seluruh gambar. Demikian halnya dengan teks, hasil dari proses *pooling* pada *image encoder* menghasilkan vektor fitur yang merepresentasikan konten visual gambar tersebut, memungkinkan analisis lebih lanjut dan pencocokan dengan deskripsi teks.

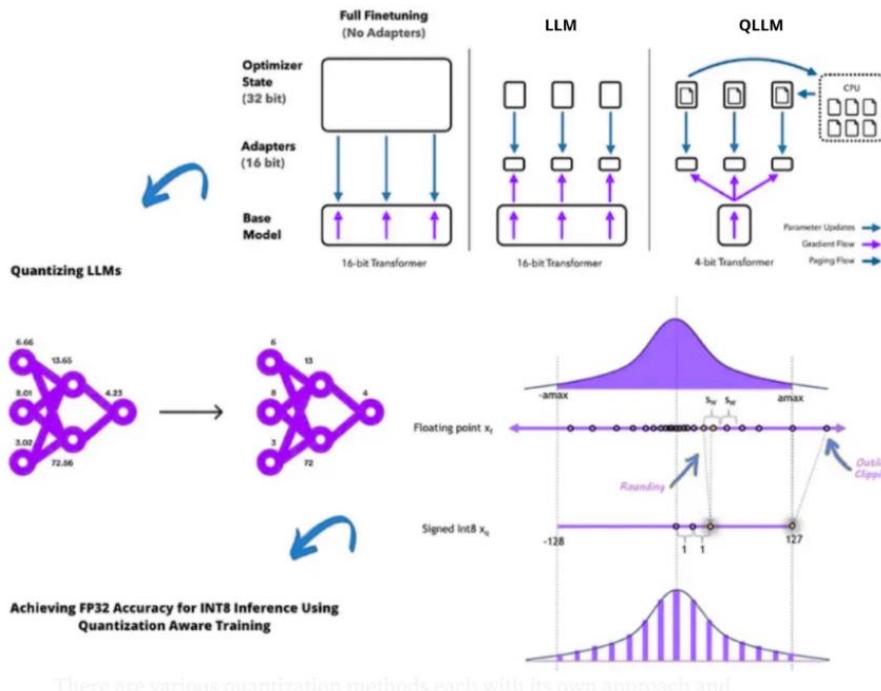


Gambar 2.4 CLIP Vision Model

Pada Gambar 2.4, proses pelatihan CLIP dilakukan menggunakan pasangan data berupa gambar dan teks yang relevan. Gambar diproses oleh *image encoder* (berbasis ResNet atau ViT), menghasilkan vektor embedding yang disimbolkan sebagai "I". Di sisi lain, teks diproses oleh *text encoder* berbasis Transformer, menghasilkan vektor embedding bernama "T". Fungsi kerugian (*loss function*) yang digunakan adalah bentuk *contrastive loss*, yang membantu CLIP belajar untuk membedakan antara pasangan positif dan negatif secara efektif. Dengan demikian, model mampu membangun pemahaman multimodal yang kuat serta kemampuan pencarian informasi silang-modus (*cross-modal retrieval*) yang handal.

2.2.4 Inference Engine menggunakan Gemma C++

Dalam pengembangan sistem AI berbasis multimodal di lingkungan *edge computing*, efisiensi model menjadi faktor krusial karena keterbatasan sumber daya komputasi pada perangkat tepi. Salah satu pendekatan yang menjanjikan adalah pemanfaatan Gemma C++ sebagai *inference engine* yang ringan dan cepat, dikombinasikan dengan *CLIP Vision Model* sebagai ekstraktor fitur visual. Gemma merupakan Large Language Model (LLM) berukuran kecil yang dikembangkan oleh Google, tersedia dalam varian 2B dan 7B parameter (Google, 2024). Berbeda dengan model besar seperti Gemini atau GPT-4, Gemma dirancang untuk menjaga keseimbangan antara efisiensi dan performa dalam berbagai tugas Natural Language Processing (NLP), seperti penerjemahan, pembuatan ringkasan dokumen, dan jawaban berbasis pertanyaan. Sifatnya yang *open source* memungkinkan fleksibilitas tinggi dalam integrasi dan modifikasi, terutama dalam sistem tertanam (*embedded system*).



Gambar 2.5 Kuantisasi Model dengan Inference Engine

Salah satu inovasi penting dalam ekosistem Gemma adalah ketersediaan versi yang dioptimalkan untuk eksekusi lokal, termasuk implementasi dalam bahasa pemrograman C++. Versi ini memungkinkan integrasi langsung ke dalam aplikasi desktop maupun perangkat edge tanpa ketergantungan pada infrastruktur cloud (Santoso, 2024). Salah satu teknik utama yang diterapkan dalam *inference engine* adalah kuantisasi (*quantization*), yang bertujuan untuk mengubah representasi bobot dan aktivasi model dari presisi tinggi (seperti FP32) ke representasi presisi rendah (seperti INT8, INT4, bahkan INT2). Teknik ini dapat secara signifikan mengurangi ukuran model dan mempercepat proses inferensi tanpa kebutuhan sumber daya yang besar.

Secara matematis, proses kuantisasi linear dapat dijelaskan dengan rumus berikut:

$$q = \text{round}\left(\frac{x - \min}{\Delta}\right)$$

dan untuk mendekode nilai asli kembali dari hasil kuantisasi:

$$x \approx \Delta \cdot q + min$$

- x : nilai bobot asli (floating-point)
- q : nilai hasil kuantisasi (integer)
- Δ : skala kuantisasi (quantization scale), dihitung dengan:

$$\Delta = \left(\frac{max - min}{2^b - 1} \right)$$

- b : jumlah bit (contoh: 8 untuk INT8, 4 untuk INT4, 2 untuk INT2)
- **min** dan **max**: rentang nilai floating point dari parameter model

Meskipun terjadi sedikit penurunan akurasi akibat proses kuantisasi, efisiensi inferensi yang diperoleh menjadikan pendekatan ini sangat cocok untuk diimplementasikan pada perangkat *edge* seperti laptop biasa, tablet, atau perangkat IoT.

2.2.5 Metrik Evaluasi

a. Evaluasi Kinerja MCP AI Agent

Untuk mengevaluasi efisiensi sistem *AI Agent* berbasis *Model Context Protocol* (MCP), dilakukan pengukuran waktu eksekusi pada beberapa tahap penting dalam alur kerja, yaitu:

1. Waktu Inisiasi Model LLM

Tahap ini mengukur waktu yang diperlukan untuk memuat model bahasa (*Large Language Model / LLM*) ke dalam memori sebelum siap digunakan.

```
start_time = time.time()
model = get_llms()
```

2. Waktu Inisiasi AI Agent

Tahap ini mengukur waktu yang dibutuhkan untuk membuat agent berbasis ReAct dengan menghubungkannya ke tools yang tersedia melalui MCP server

```
start_time = time.time()
agent = create_react_agent(model, client.get_tools())
```

3. Response Time (Inferensi)

Mengukur durasi dari saat agent menerima *query* hingga memberikan respons akhir kepada pengguna.

```
start_time = time.time()
response = await agent.ainvoke(query)
```

b. Evaluasi Kinerja Image Captioning

BLEU (*BiLingual Evaluation Understudy*) adalah metrik yang digunakan dalam pemrosesan bahasa alami (NLP) dan terjemahan mesin untuk mengevaluasi kualitas teks yang dihasilkan dengan membandingkannya terhadap satu atau lebih terjemahan referensi berkualitas tinggi. BLEU mengukur seberapa mirip teks hasil terjemahan mesin dengan teks referensi yang dibuat manusia dengan cara membandingkan n-gram (urutan n kata berurutan) antara keduanya. Skor BLEU dihitung berdasarkan presisi n-gram yang dimodifikasi dan diberi penalti jika hasil terjemahan terlalu pendek dibanding referensi (Herry, 2014).

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right)$$

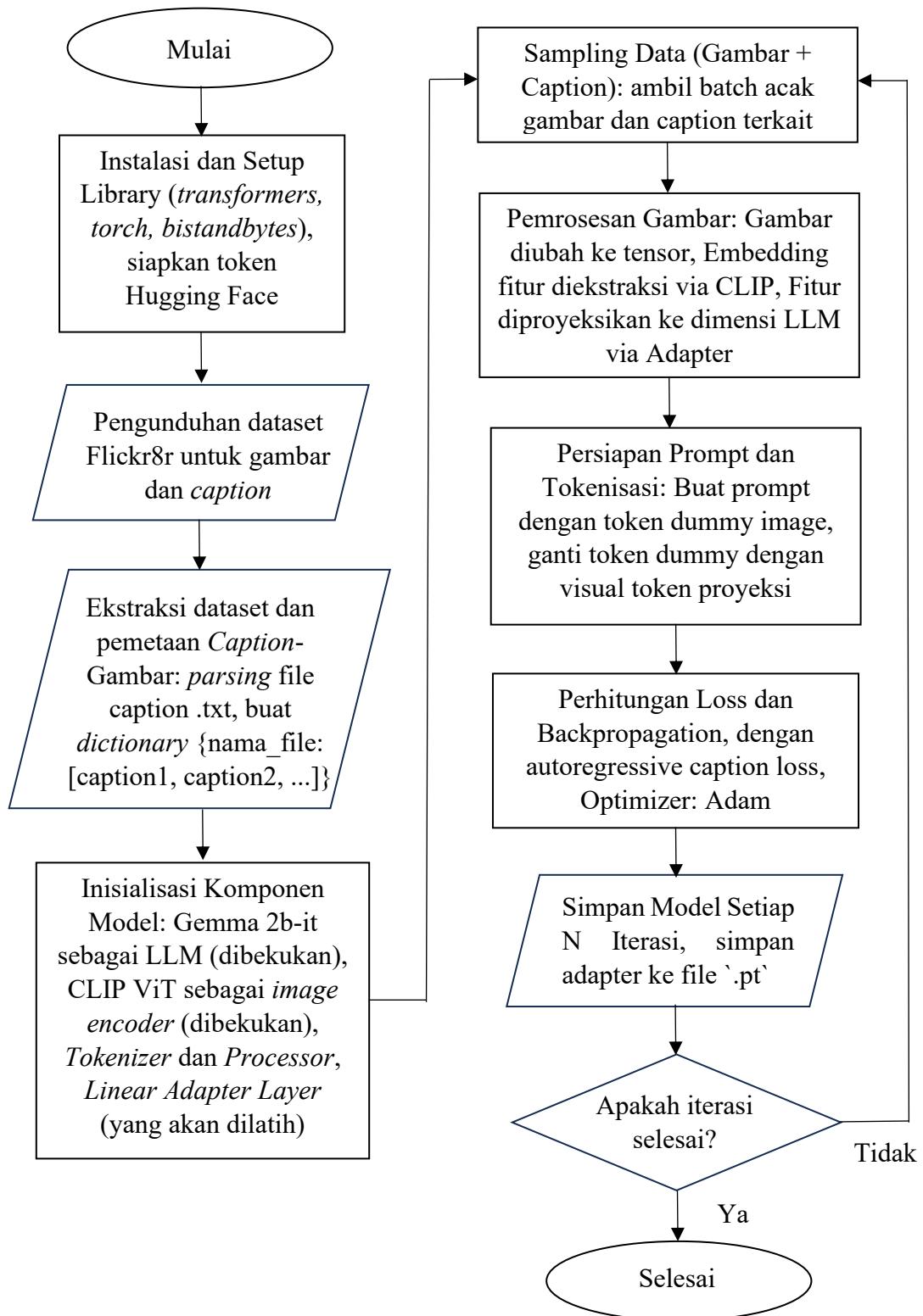
Dimana:

- BP : *brevity penalty* (hukuman untuk hasil terjemahan yang terlalu singkat dibanding referensi)
- N adalah urutan maksimum *n-gram* yang dipertimbangkan (umumnya 4)
- w_n adalah bobot untuk setiap *n-gram* ke- n (biasanya sama besar, yaitu $1/N$)
- p_n adalah *modified precision* untuk *n-gram* ke- n , yaitu jumlah *n-gram* hasil terjemahan yang cocok dengan referensi dibagi jumlah total *n-gram* pada hasil terjemahan

BAB 3 METODOLOGI

3.1 Perancangan Sistem

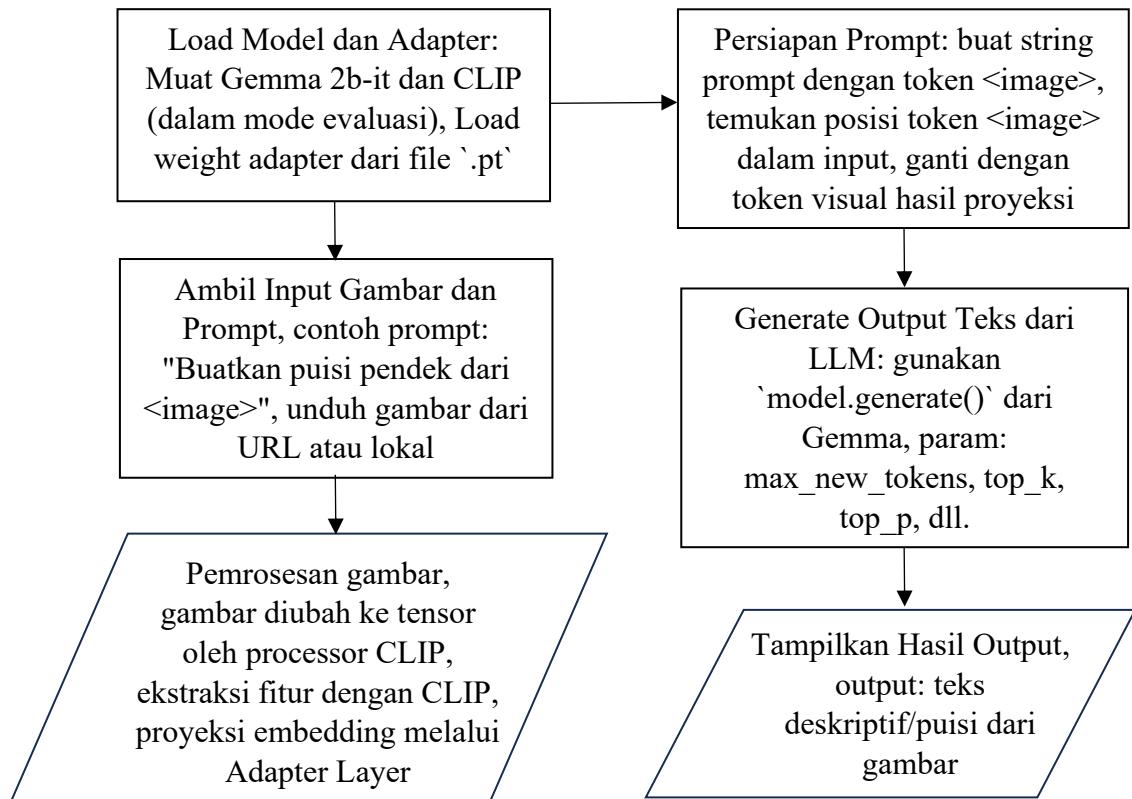
3.1.1. Model Training



Gambar 3.1 Alur Training Model Image Captioning

Proses pelatihan model image captioning ini mengacu langsung pada pendekatan Joan Santoso yang menggabungkan CLIP ViT-L/14 sebagai visual encoder dengan Gemma 2 sebagai text decoder dalam kerangka multimodal. Dataset yang digunakan adalah MS-COCO atau alternatif seperti Flickr8k, yang telah tersedia dalam format anotasi pasangan gambar dan caption. Langkah pertama adalah melakukan *pre-processing dataset* dengan mengonversi setiap caption menjadi token menggunakan *tokenizer* Gemma 2, dan menyiapkan gambar dalam format yang sesuai untuk *input encoder* CLIP. Setiap gambar kemudian diproses melalui CLIP *Vision Transformer* (ViT-L/14), yang menghasilkan representasi embedding visual berdimensi tinggi. Embedding ini kemudian dipasangkan dengan prompt teks berupa pertanyaan terbuka seperti “*What is described in this image?*” yang dijadikan awal input bagi decoder. Prompt dan embedding visual dikombinasikan untuk menghasilkan representasi multimodal yang dapat ditangani oleh Gemma 2. Selanjutnya, decoder Gemma 2 melakukan prediksi urutan teks berdasarkan representasi tersebut, yaitu menghasilkan caption deskriptif untuk setiap gambar. Proses training dilakukan dengan optimisasi loss terhadap perbedaan antara caption yang dihasilkan dan ground truth menggunakan *cross-entropy loss*. Model dilatih menggunakan fine-tuning pada Gemma 2 dan CLIP, dengan konfigurasi hyperparameter seperti learning rate dan batch size yang disesuaikan berdasarkan eksperimen awal. Pelatihan dilakukan pada GPU dengan dukungan memory besar, mengingat ukuran model yang cukup besar (Gemma 2 memiliki miliaran parameter). Output akhir dari proses pelatihan ini adalah model image captioning multimodal berbasis encoder-decoder yang telah mampu memahami konteks visual dan merespons dengan deskripsi tekstual yang relevan, siap digunakan pada tahap inference.

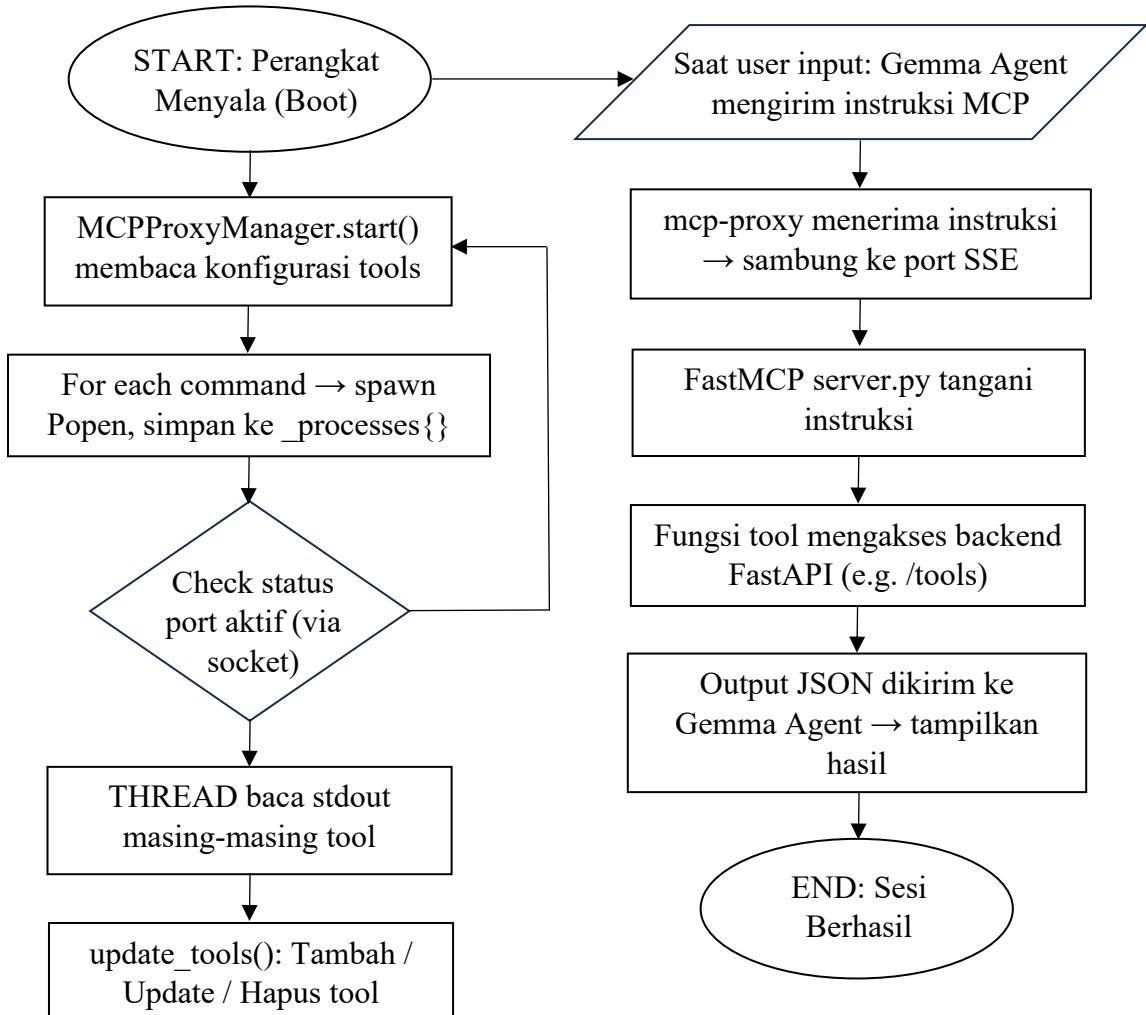
3.1.2 Inference Engine



Gambar 3.2 Alur Inference Engine

Setelah model multimodal berhasil dilatih, proses inference dilakukan untuk menghasilkan deskripsi otomatis dari gambar baru. Pada tahap ini, sistem menerima input berupa citra digital yang akan diproses untuk mendapatkan representasi visual menggunakan CLIP ViT-L/14 sebagai encoder. Gambar diubah menjadi tensor dan dinormalisasi sesuai spesifikasi input CLIP, lalu diekstraksi menjadi embedding visual berukuran tetap. Embedding ini kemudian dipasangkan dengan prompt teks awal seperti “*What is described in this image?*” yang telah ditokenisasi menggunakan tokenizer Gemma 2. Gabungan *embedding visual* dan *prompt text token* disusun dalam format input multimodal yang dimengerti oleh *decoder* Gemma 2. Selama proses inference, Gemma 2 akan memproses input tersebut secara *autoregressive*, menghasilkan token demi token secara berurutan untuk membentuk caption akhir. Proses ini tidak melibatkan pembaruan parameter, melainkan hanya *forward pass* dari model yang telah dilatih sebelumnya. Untuk menghasilkan teks yang natural dan relevan, digunakan teknik *decoding* seperti *greedy decoding* atau *beam search* tergantung kebutuhan akurasi dan efisiensi. Caption yang dihasilkan akan berupa deskripsi *natural language* yang menjelaskan isi gambar. Output ini kemudian dapat digunakan lebih lanjut, proses inference ini dirancang agar berjalan efisien dalam waktu nyata (*real-time*), terutama ketika digunakan pada sistem *edge device* atau asisten virtual berbasis IoT seperti dalam kasus Neutrack Glove.

3.1.3 Model Context Protocol (MCP)



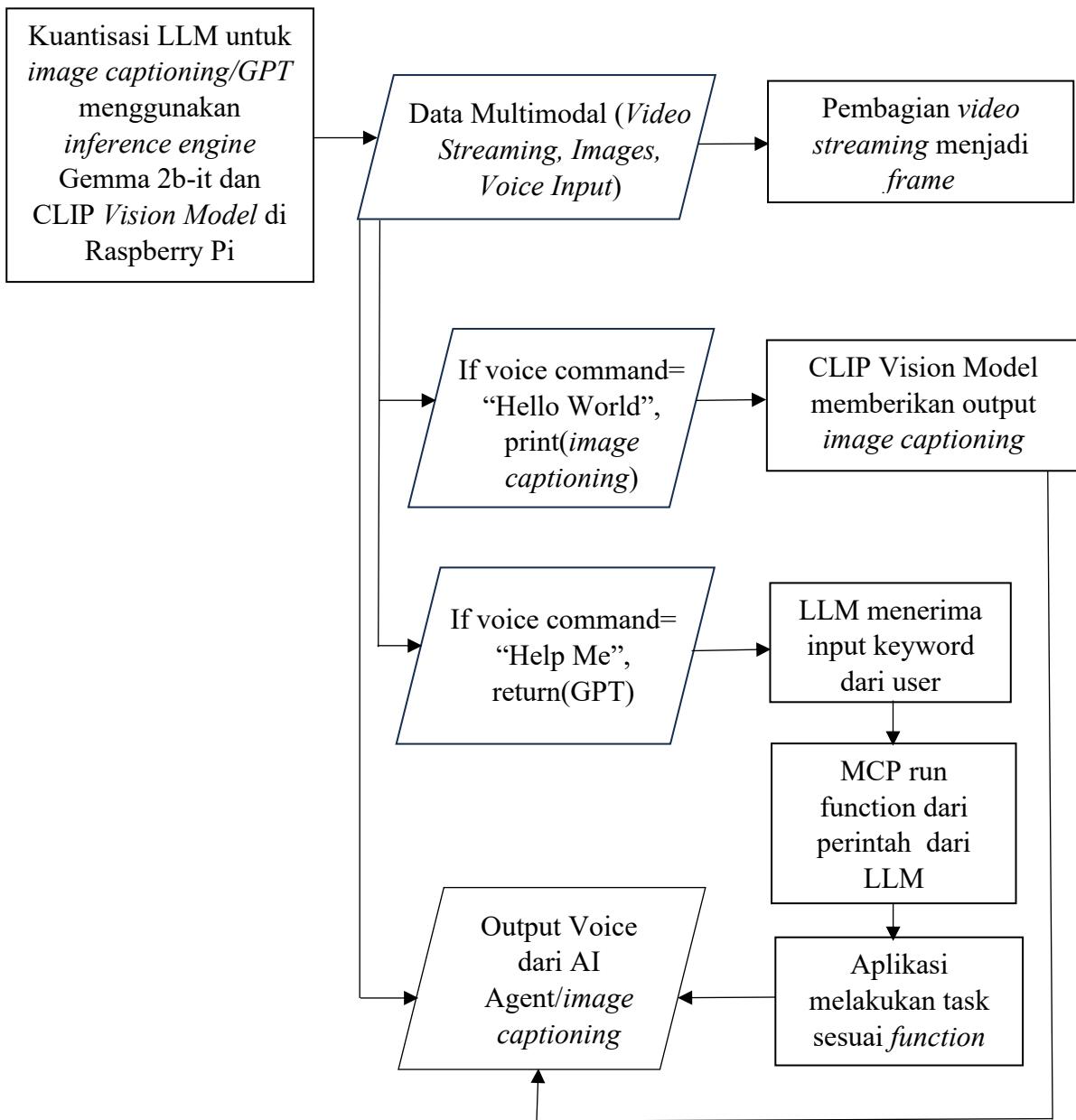
Gambar 3.3 Alur Model Context Protocol Proxy Manager

Agar sistem Neutrack AI Glove mampu melakukan interaksi dengan aplikasi di luar sistem, digunakan pendekatan *Model Context Protocol* (MCP). MCP merupakan protokol standar yang memungkinkan model *Large Language Models* (LLMs) seperti Gemma untuk mengakses dan mengoordinasikan alat eksternal (*tool*) seperti modul segmentasi gambar, *encoder* CLIP, hingga masukan suara melalui antarmuka universal berbasis JSON over HTTP atau stdio. Dengan kata lain, MCP menjembatani komunikasi antara LLM dan berbagai layanan eksternal secara fleksibel dan dinamis. Pada tahap inisialisasi, proses dimulai ketika skrip MCPProxyManager.start() dijalankan oleh sistem untuk membaca daftar perintah dari konfigurasi YAML atau *database*. Setiap *command line* seperti mcp-proxy --sse-port=10021 ... akan diurai menggunakan fungsi _extract_port(), kemudian dijalankan sebagai proses terpisah menggunakan subprocess.Popen() dan disimpan ke dalam struktur dict bernama _processes. Proses-proses ini dikelola dengan pengawasan log melalui thread pembaca stdout, serta pemantauan port aktif melalui check_status() dengan socket checker_is_port_in_use(). Untuk menjamin ketersediaan layanan secara berkelanjutan, metode update_tools() dipanggil berkala guna mendeteksi perubahan pada konfigurasi command yang sedang berjalan: jika ada penambahan maka tool dijalankan, jika ada perubahan maka proses dihentikan lalu dijalankan ulang, dan jika dihapus maka proses dimatikan.

Setelah layanan aktif, MCP memungkinkan agen seperti Gemma untuk mengirimkan perintah dalam bentuk instruksi JSON (misalnya call_tool(get_tools, {...})), yang diterima oleh mcp-proxy dan diteruskan melalui protokol *Server-Sent Event* (SSE) ke *FastMCP Agent Tools Server*. Di sinilah fungsi Python yang telah terdaftar melalui dekorator @mcp.tool() dijalankan—misalnya, get_tools()—yang akan melakukan permintaan ke backend FastAPI (mis. endpoint /tools) menggunakan klien HTTP *asynchronous*, kemudian mengembalikan hasil dalam bentuk JSON ke agen Gemma. Alur ini memastikan bahwa setiap input dari pengguna, seperti menekan tombol “Assist” pada glove, akan diterjemahkan menjadi instruksi yang memicu aktivasi modul tertentu sesuai konteks (misalnya run_rag atau face_recognition) secara dinamis. Flowchart MCP menggambarkan urutan tersebut dengan simbol proses untuk inisialisasi skrip dan spawn tool, simbol keputusan untuk pengecekan status port, dan simbol input/output saat instruksi dikirim maupun hasil dikembalikan. Dengan sistem ini, MCP tidak hanya menjamin modularitas dan skalabilitas, tetapi juga mendukung *hot-reload*, *logging real-time*, keamanan token, dan interoperabilitas antara komponen LLM dan non-LLM, menjadikannya tulang punggung arsitektur inferensi lokal yang adaptif dan andal dalam Neutrack AI Glove.

3.1.4 Pengujian dan Evaluasi

Tahapan ini memfokuskan pada proses pengujian dan evaluasi performa model klasifikasi multilabel yang telah dikembangkan. Untuk memberikan gambaran menyeluruh terkait alur metode penelitian yang digunakan, berikut ditampilkan diagram alur proses pengujian dan evaluasi model:



Gambar 3.4 Alur Tahap Pengujian dengan Perintah Suara

Pada tahap pengujian dan evaluasi, sistem NeutraGem Inference Engine diuji untuk memastikan setiap komponen dari pipeline multimodal berfungsi secara sinergis dalam eksekusi lokal di perangkat edge seperti Raspberry Pi. Berdasarkan *flowchart* pada Gambar 3.4, pengujian dimulai dari proses kuantisasi dan deploy model yang telah dilatih (Gemma 2b-it dan CLIP Vision) ke dalam sistem inference lokal. Sistem kemudian menerima masukan multimodal berupa input suara dan visual secara bersamaan—gambar/video streaming yang dipecah menjadi frame dan masukan suara yang ditangkap dari mikrofon. Setiap input visual diproses oleh CLIP Vision Model untuk menghasilkan embedding yang relevan, lalu diteruskan ke decoder Gemma untuk menghasilkan caption gambar. Sistem mengenali perintah suara seperti “Hello World” sebagai pemicu untuk menampilkan hasil caption, dan “Help Me” sebagai pemicu untuk menjalankan fungsi GPT melalui perantara Model Context Protocol

(MCP). Di sinilah MCP berperan sebagai jembatan antara LLM dengan alat eksternal untuk menjalankan fungsi seperti navigasi, notifikasi, atau pengenalan wajah secara modular. Evaluasi dilakukan dengan mengamati apakah sistem mampu memberikan output suara yang tepat dari AI Agent berdasarkan perintah yang diberikan, serta seberapa cepat respons diberikan tanpa keterlambatan yang signifikan.

Selain itu, sistem diuji dengan berbagai kondisi lingkungan untuk mengevaluasi kehandalan modul TTS, akurasi caption dari input visual, serta konsistensi aktivasi fungsi melalui MCP. Keseluruhan sistem dirancang untuk mendukung interaksi real-time yang privat dan efisien, dengan prinsip edge computing yang mengurangi ketergantungan pada cloud, sehingga aman digunakan oleh penyandang tunanetra maupun demensia. Dari hasil pengujian, dapat disimpulkan bahwa integrasi antara pelatihan model, inference lokal, dan orkestrasi melalui MCP telah membentuk ekosistem AI modular yang tangguh, responsif, dan sesuai untuk kebutuhan asisten virtual berbasis wearable IoT.⁷ Evaluasi performa model ini menjadi landasan untuk integrasi lebih lanjut ke dalam sistem nyata yang akan diimplementasikan. Setelah tahap pengujian model selesai, proses berikutnya adalah merancang dan menyesuaikan perangkat keras dari sistem embedded yang akan menjalankan inference secara langsung di lapangan.

Perancangan perangkat keras dari *embedded systems* sebagai *virtual assistant* bagi penyandang tunanetra dan demensia meliputi personalisasi dan arsitektur sistem, desain tiga dimensi piranti, dan prinsip kerja alat.

Bagaimana cara mempersonalisasi fitur IoT Neutrack?



Gambar 3.5 Konfigurasi Hardware Neutrack AI Glove

Berdasarkan kebutuhan, tunanetra atau individu dengan gangguan penglihatan perlu menggunakan semua modul sistem benam. Sementara itu, pengguna dengan tingkat demensia tertentu hanya memerlukan Raspberry Pi 4B+, kamera, GPS, dan headset. Personalisasi ini memungkinkan solusi yang tepat guna dan sesuai anggaran. Arsitektur hardware wearable IoT mencakup modul GPS untuk menangkap koordinat, Firebase dan Google Cloud Storage sebagai database real-time, serta antarmuka frontend yang terintegrasi dengan Google Maps API. Untuk pemrosesan gambar, digunakan Gemini API (online) atau TensorFlow Lite (lokal) di Raspberry Pi, menghasilkan output teks-ke-suara.

3.1.2 Prinsip Kerja alat

Software dari wearable IoT diprogram secara modular sehingga pengguna dapat mengubah mode dengan memasukkan perintah suara setelah menekan tombol untuk menghentikan loop "while".

1. Perceive Efficiently

Perintah suara “Hello World/Halo Dunia” akan mengaktifkan image captioning berbasis cloud, sensor ultrasonik untuk mengenali lingkungan sekitar bagi tunanetra, serta object detection menggunakan TensorFlow Lite secara lokal. Sensor ultrasonik juga memicu buzzer atau motor getar saat mendekripsi objek dalam jarak ≤ 70 cm.

2. Re-track Where You Are

Lalu, perintah suara “Tunjukkan Jalan” mengaktifkan modul pemrograman yang berfungsi untuk GPS Tracking dan pencarian jalan tercepat, sehingga penyandang tunanetra maupun demensia tidak perlu khawatir dalam berjalan kaki maupun bernavigasi. Harapannya, keluarga dapat memantau lokasi melalui website GPS Tracking yang terhubung secara real-time

3. Recall-Your Loved Ones

Setelah itu, Neutrack dapat diarahkan kepada wajah lawan bicara dengan memasukkan perintah suara “Siapakah Dia?” mengaktifkan fungsi train_model yang dapat menangkap wajah yang ingin di-training secara langsung dengan model DeepFace Face Recognition. Setelah pencocokan data berhasil, pengguna dapat melakukan face recognition.

Selain itu, pengguna dapat menekan tombol untuk menghentikan loop program. Lalu, perintah suara “Help Me!” mengaktifkan fitur jarvis berbasis *Generative Pre-Trained Transformer* (GPT) AI Agent sebagai asisten virtual yang siap membantu untuk chat, juga menggunakan aplikasi-aplikasi hanya dengan berbicara dengan chatbot.

3.1.1. Perancangan Software Visual Assistant

Perancangan perangkat lunak dari *embedded systems* sebagai virtual assistant bagi penyandang tunanetra dan demensia meliputi arsitektur website GPS Tracking, aplikasi talkback untuk tunanetra, aplikasi chatbot untuk demensia, dan NeutraGem *AI Inference Engine*. Situs web GPS tracking dikembangkan dengan Bootstrap, HTML, CSS, dan JavaScript di sisi *frontend*, serta Flask di *backend*. Tampilan peta menggunakan Google Maps API yang terintegrasi dengan modul GPS Ublox Neo 6M, dengan koordinat lokasi disimpan secara real-time di Firebase dan Google Cloud. Sistem *Visual Assistant* ini dirancang dengan pendekatan *multimodal image captioning* berbasis *Retrieval-Augmented Generation* (RAG) yang dilatih secara lokal menggunakan kombinasi CLIP ViT sebagai *encoder* visual dan Gemma 2 2B-it sebagai LLM, dengan adaptor linear sebagai jembatan antara representasi visual dan teks; setelah pelatihan, engine yang sama digunakan untuk inferensi deskriptif secara real-time melalui NeutraGem Inference Engine, sedangkan manajemen dan eksekusi dinamis seluruh proses LLM, termasuk pemanggilan agent dan modularisasi, dikendalikan secara otomatis oleh MCPProxyManager berbasis port.

3.2 Bahan dan Peralatan yang Digunakan

Untuk mendukung penelitian ini, sejumlah peralatan dan bahan telah disiapkan yang dapat dilihat sebagai berikut :

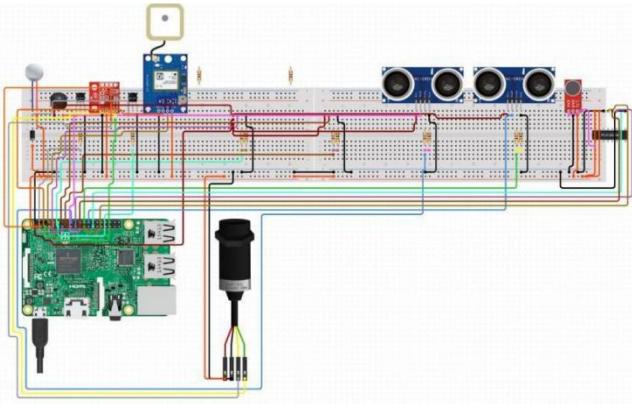
3.2.1 Perangkat Keras

4. Processor 13th Gen Intel(R) Core (TM) i9-13900H 2.60 GHz
5. Random Access Memory (RAM) 16 GB
6. NVIDIA® GeForce RTX™ 4070 with 8GB GDDR6
7. SSD 1 TB
8. Mouse dan Keyboard

3.2.2 Perangkat Lunak

1. Sistem Operasi Ubuntu 24.04.2 LTS
2. *Visual Studio Code*
3. *Library* pendukung di Python

3.2.3 Komponen *Neutrack AI Glove*



1. *Raspberry Pi 4B+*
2. *USB Camera 8MP* Kabel Tarik
3. *GPS Module Ublox Neo 6M*
4. Sensor Ultrasonik
5. *Buzzer 5V*
6. *Powerbank 5V 10.000 mAh*
7. *Fan DC 5V 0.30A 6x6 cm plus USB*
8. *Vibration Motor*
9. Mikrofon Mini USB U01
10. PCB

3.3 Rencana Implementasi Uji Coba

3.1.1. *Model Training*

File `training_model.py` berfungsi sebagai fondasi utama dalam proses pelatihan model multimodal berbasis Gemma 2, yang mampu memahami dan menghasilkan teks berdasarkan masukan visual berupa gambar. Model ini menggabungkan fitur visual dari gambar menggunakan encoder CLIP ViT dan meneruskan representasinya ke dalam Linear Adapter Layer, sebelum dikirimkan ke Gemma 2 2B-it, sebuah large language model (LLM) yang telah dilatih sebelumnya oleh Google.

Selama proses pelatihan, fitur gambar dikonversi menjadi vektor berdimensi 768, lalu diproyeksikan ke ruang embedding teks berdimensi 2304. Tujuan utama pelatihan adalah melatih adaptor linear agar mampu menjembatani representasi visual ke dalam konteks bahasa yang dapat dipahami oleh Gemma. Dengan membekukan parameter dari model CLIP dan Gemma 2, hanya lapisan adaptor yang mengalami pembaruan bobot menggunakan loss autoregresif, sehingga model tetap ringan dan efisien untuk digunakan di perangkat edge seperti Raspberry Pi.

Proses training menggunakan dataset Flickr8k, dengan input berupa pasangan gambar dan caption. Ukuran gambar disesuaikan dengan kebutuhan CLIP ViT, yaitu (3, 224, 224) piksel. Setiap caption dipasangkan dengan token khusus `<start_image>` untuk menyisipkan token

gambar, dan kemudian diikuti oleh instruksi yang diformat dalam gaya *chat template* seperti yang digunakan oleh Gemma.

Tabel 3.1 Pseudocode Pelatihan Model Berbasis Gemma 2 dan CLIP ViT

```
# Deskripsi: Model multimodal untuk pelatihan captioning berbasis Gemma 2  
dan CLIP ViT  
# Input: Gambar berukuran (3, 224, 224) dan teks (caption)  
# Output: Loss training autoregresif untuk adaptasi visual ke teks  
  
01| IMPORT torch, torch.nn, torch.nn.functional as F  
02| FROM transformers IMPORT AutoProcessor, AutoTokenizer,  
AutoModelForCausalLM, CLIPVisionModel  
03| FROM PIL IMPORT Image  
04| IMPORT numpy as np  
05|  
06| CLASS MyAdaptor(nn.Module):  
07|     FUNCTION __init__(self, vis_dim: int, word_dim: int):  
08|         self.adapter_linear = nn.Linear(vis_dim, word_dim)  
09|  
10|     FUNCTION forward(self, img_output):  
11|         RETURN self.adapter_linear(img_output)  
12|  
13|  
14| CLASS MyModel(nn.Module):  
15|     FUNCTION __init__(self):  
16|             self.model_language =  
AutoModelForCausalLM.from_pretrained("google/gemma-2-2b-it")  
17|             self.tokenizer = AutoTokenizer.from_pretrained("google/gemma-  
2-2b-it", padding_side="right")  
18|             self.image_processor =  
AutoProcessor.from_pretrained("openai/clip-vit-base-  
patch32").image_processor  
19|             self.model_image =  
CLIPVisionModel.from_pretrained("openai/clip-vit-base-patch32")  
20|  
21|             self.word_dim = 2304  
22|             self.vis_dim = 768  
23|             self.num_vis_token = 50  
24|             self.trigger_str = "<start_image>"  
25|             self.adaptor = MyAdaptor(self.vis_dim, self.word_dim)  
26|             self.dummy_img_token = (" ".join(["the"] *  
self.num_vis_token)).strip()  
27|  
28|     FUNCTION forward_loss(self, images: list[Image], captions:  
list[str]) -> float:  
29|         # Format instruksi prompt untuk Gemma  
30|         instruction = "<start_of_turn>user\n"  
31|                         instruction += f"<start_image>  
{self.dummy_img_token}\n<end_image>\n"
```

```

32| # Preprocessing input gambar
33|         # Asumsikan input 'images' adalah list path string dari dataset
(misal: Flickr8k)
34|         image_list = [Image.open(img_path).convert("RGB").resize((224,
224)) for img_path in images]
35|         # Gunakan processor bawaan CLIP ViT untuk mengubah ke tensor
(B, 3, 224, 224)
36|             pixel_values = self.image_processor(image_list,
return_tensors="pt")['pixel_values']
37|             pixel_values = pixel_values.to(device)
38|             image_embed = self.get_image_embed(pixel_values)38|     #
Menyiapkan prompt multimodal untuk pelatihan caption
39|                 prompt = "<start_of_turn>user\n<start_image> " +
self.dummy_img_token + "\n<end_image>\n..."
40|                 full_inputs = [prompt + c for c in captions]
41|                 input_tokenized = self.tokenizer_language(full_inputs,
return_tensors="pt", padding=True)
42|                 # Dapatkan token embedding
43|                 token_ids = input_tokenized['input_ids']
44|                 attention_mask = input_tokenized['attention_mask']
45|                     token_embeddings = self.model_language.model.embed_tokens(token_ids)
46|                     # Ganti bagian <start_image> dengan fitur visual yang telah
diadaptasi
47|                     replaced_embedding = replace_visual_embedding(token_embeddings,
image_embed)
48|                     # Hitung prediksi dan loss
49|                     logits = self.model_language(inputs_embeds=replaced_embedding,
attention_mask=attention_mask)['logits']
50|                     loss = autoregressive_loss(logits, token_ids)
51|                     RETURN loss
52|
53| FUNCTION train_model():
54|     model = MyModel()
55|     FREEZE(model.model_language)
56|     FREEZE(model.model_image)
57|     optimizer = Adam(model.adaptor.parameters(), lr=1e-4)
58|     FOR step IN range(NUM_ITERATION):
59|         images, captions = sample_data(Flickr8k, n=BATCH_SIZE)
60|         loss = model.forward_loss(images, captions)
61|         optimizer.zero_grad()
62|         loss.backward()
63|         optimizer.step()
64|         IF step % SAVE_INTERVAL == 0:
65|             SAVE(model.adaptor, SAVED_PATH)

```

3.1.2 Inference Engine

File inference_engine.py merupakan inti dari sistem prediksi multimodal berbasis *Vision-to-Text* pada NeutraGem, yang mengadaptasi pendekatan artikel Joan Santoso (2024) dengan menggunakan *Gemma 2* sebagai LLM decoder dan *CLIP ViT* sebagai image encoder. Sistem ini memungkinkan pengguna untuk memberikan input berupa gambar berukuran tetap (3, 224, 224)—yang akan dikonversi menjadi representasi vektor visual oleh CLIP—and menghasilkan deskripsi naratif atau caption yang relevan secara semantik menggunakan *Gemma 2 2B-it*.

Tabel 3.2 Pseudocode Inference Engine

```
# Deskripsi: Modul inferensi multimodal untuk menghasilkan caption dari
# gambar menggunakan CLIP dan Gemma 2
# Input: Gambar berukuran (3, 224, 224)
# Output: Teks caption hasil decoding dari model Gemma 2

01| IMPORT torch, torch.nn AS nn
02| FROM transformers IMPORT AutoProcessor, AutoTokenizer,
AutoModelForCausalLM, CLIPVisionModel
03| FROM PIL IMPORT Image
04| IMPORT numpy AS np
05|
06| CLASS MyAdaptor(nn.Module):
07|     FUNCTION __init__(self, vis_dim: int, word_dim: int):
08|         self.adapter_linear = nn.Linear(vis_dim, word_dim)
09|
10|     FUNCTION forward(self, img_output):
11|         RETURN self.adapter_linear(img_output)
12|
13|
14| CLASS VisionToTextInference:
15|     FUNCTION __init__(self):
16|         # Load LLM Gemma 2
17|         self.model_language
AutoModelForCausalLM.from_pretrained("google/gemma-2-2b-it")
18|         self.tokenizer =
AutoTokenizer.from_pretrained("google/gemma-2-2b-it",
padding_side="right")
19|
20|         # Load CLIP encoder
21|         self.image_processor =
AutoProcessor.from_pretrained("openai/clip-vit-base-
patch32").image_processor
22|         self.model_image =
CLIPVisionModel.from_pretrained("openai/clip-vit-base-patch32")
23|
```

```

24|         # Load adaptor
25|         self.word_dim = 2304
26|         self.vis_dim = 768
27|         self.num_vis_token = 50
28|         self.trigger_str = "<start_image>"
29|         self.adaptor = MyAdaptor(self.vis_dim, self.word_dim)
30|             self.dummy_img_token = (" ".join(["the"] * self.num_vis_token)).strip()
31|
32|     # Fungsi untuk menyandikan gambar ke dalam embedding visual
33|     FUNCTION get_image_embed(self, image: Image) -> torch.Tensor:
34|         pixel_values = self.image_processor(image,
35|                                             return_tensors="pt")['pixel_values'] # shape: (1, 3, 224, 224)
36|         clip_output = self.model_image(pixel_values)['last_hidden_state'] # shape: (1, T, 768)
37|         adapted = self.adaptor(clip_output) # shape: (1, T, 2304)
38|         RETURN adapted
39|
40|     # Fungsi utama untuk menghasilkan caption dari gambar input
41|     FUNCTION generate_caption(self, image: Image) -> str:
42|         image_embed = self.get_image_embed(image) # vektor embedding
43|         prompt_text = "<start_of_turn>user\n<start_image> " +
44|                         self.dummy_img_token + "\n<end_image>\nDescribe this image."
45|         tokenized_input = self.tokenizer(prompt_text,
46|                                         return_tensors="pt")
47|         token_ids = tokenized_input["input_ids"]
48|         attention_mask = tokenized_input["attention_mask"]
49|         token_embeddings = self.model_language.model.embed_tokens
50|         (token_ids)
51|         outputs = self.model_language.generate(
52|             inputs_embeds=final_embeddings,
53|             attention_mask=attention_mask,
54|             max_new_tokens=50,
55|             do_sample=False # greedy decoding)
56|         decoded = self.tokenizer.decode(outputs[0], skip_special_tokens=True)
57|         RETURN decoded
58| # Contoh pemanggilan untuk gambar lokal
59| IF __name__ == "__main__":
60|     model = VisionToTextInference()
61|     img=Image.open("dataset/flickr8k/test_image.jpg").convert("RGB")
62|     .resize((224, 224))
63|     caption = model.generate_caption(img)
64|     PRINT("Hasil Caption:", caption)

```

3.1.3 Model Context Protocol (MCP)

File embedding_utils.py berfungsi sebagai pembungkus (*wrapper*) untuk layanan embedding yang dijalankan secara lokal. Embedding ini penting dalam *pipeline retrieval-augmented generation* (RAG) karena akan mengubah input teks menjadi representasi vektor berdimensi tinggi. Representasi ini akan digunakan dalam pencocokan dokumen berbasis kemiripan (cosine similarity). Komponen ini juga digunakan oleh fungsi retrieval_with_rerank() dalam file rag_utils.py.

Tabel 3.3 Pseudocode Embedding Pipeline Retrieval Augmented Generation

```
# Deskripsi: Pembungkus layanan embedding lokal untuk sistem RAG
# Input: list[str] (dokumen teks atau query string tunggal)
# Output: list[float] (vektor embedding berdimensi 1536)
01| CLASS EmbedderService:
02|     # Inisialisasi objek embedding saat kelas dipanggil
03|     FUNCTION __init__(self):
04|         self._init_embedding_model()
05|
06|     # Memuat model embedding lokal dengan dimensi tetap
07|     FUNCTION _init_embedding_model(self):
08|         self.embedding_model = CustomLocalEmbeddingModel(dimensions=1536)
09|
10|     # Fungsi untuk mengubah list dokumen menjadi list vektor embedding
11|     FUNCTION embed_documents(self, texts: list[str]) -> list[list[float]]:
12|         RETURN self.embedding_model.embed_documents(texts)
13|
14|     # Fungsi untuk mengubah 1 query (string) menjadi vektor embedding
15|     FUNCTION embed_query(self, query: str) -> list[float]:
16|         RETURN self.embedding_model.embed_query(query)
17|
18|     # Properti untuk mendapatkan panjang vektor hasil embedding
19|     PROPERTY embedding_dim(self) -> int:
20|         RETURN len(self.embed_query("test"))
21|
22|
23| # Contoh pemanggilan fungsi saat dijalankan langsung
24| IF __name__ == "__main__":
25|     embedder = EmbedderService()
26|
27|     documents = [
28|         "Sistem multimodal digunakan untuk bantu tunanetra.",
29|         "Embedding lokal digunakan dalam NeutraGem."
30|     ]
31| 
```

```

32|     # Embedding dokumen
33|     doc_embeddings = embedder.embed_documents(documents)
34|     PRINT("Embeddings dokumen:", doc_embeddings)
35|
36|     # Embedding query tunggal
37|     query = "Apa itu NeutraGem?"
38|     query_embedding = embedder.embed_query(query)
39|     PRINT("Embedding query:", query_embedding)
40|
41|     # Cek panjang vektor embedding
42|     PRINT("Dimensi vektor:", embedder.embedding_dim)

```

File `rag_utils.py` ini bertugas menyatukan proses pencarian berbasis vektor (`retrieval_with_rerank`) dan generasi jawaban (`generate_response`) dengan menggunakan model LLM lokal. Sistem akan mengambil dokumen yang paling relevan terhadap pertanyaan pengguna dari basis data vektor embedding, lalu memberi perintah ke LLM untuk menjawab berdasarkan konteks tersebut. File ini bergantung pada `embedding_utils.py` dan `get_llms.py`.

Tabel 3.4 Pseudocode Retrieval with Rerank dan Generate Response Berbasis Vektor

```

# Deskripsi: Utilitas RAG untuk retrieval dan generation jawaban dalam
sistem NeutraGem
# Input:
#   - query: str (pertanyaan dari pengguna)
#   - documents: list[str] (kumpulan dokumen embedding)
# Output:
#   - top_k_docs: list[str] (dokumen paling relevan)
#   - answer: str (jawaban dari model LLM berdasarkan dokumen)
01| IMPORT numpy AS np
02| FROM embedding_utils IMPORT EmbedderService
03| FROM get_llms IMPORT get_llms
04| # Inisialisasi model embedding dan LLM lokal
05| embedder = EmbedderService()
06| llm = get_llms(model_name="gemma-2b")
07| # Menghitung cosine similarity antar dua vektor
08| FUNCTION cosine_similarity(vec1: list[float], vec2: list[float]) ->
float:
09|         RETURN np.dot(vec1, vec2) / (np.linalg.norm(vec1) *
np.linalg.norm(vec2))

10| # Mengambil dokumen paling relevan dari kumpulan dokumen berdasarkan
query
11| FUNCTION retrieval_with_rerank(query: str, documents: list[str],
top_k=3) -> list[str]:

```

```

15|     similarity_scores = [cosine_similarity(query_vector, doc_vec)
16|                           FOR doc_vec IN document_vectors]
17|
18|     sorted_indices = np.argsort(similarity_scores)[::-1]
19|     top_docs = [documents[i] FOR i IN sorted_indices[:top_k]]
20|
21|     RETURN top_docs

22| # Menghasilkan jawaban dari LLM berdasarkan konteks dokumen
23| FUNCTION generate_response(query: str, documents: list[str]) -> str:
24|     context = "\n".join(documents)
25|
26|     prompt = f"""
27|         Berdasarkan informasi berikut, jawablah pertanyaan di bawah:
28|         ---\n{context}\n---
29|         Pertanyaan: {query}
30|         Jawaban:
31|         """
32|
33|     response = llm.invoke(prompt)
34|     RETURN response

35| # Contoh penggunaan saat file dijalankan langsung
36| IF __name__ == "__main__":
37|     query = "Apa itu NeutraGem?"
38|     docs = [
39|         "NeutraGem adalah sistem multimodal berbasis LLM lokal.",
40|         "Sistem ini menggunakan Raspberry Pi dan TensorFlow Lite.",
41|         "Embedding digunakan untuk pencarian konteks."
42|     ]
43|
44|     relevant_docs = retrieval_with_rerank(query, docs)
45|     answer = generate_response(query, relevant_docs)
46|
47|     PRINT("Dokumen relevan:", relevant_docs)
48|     PRINT("Jawaban:", answer)

```

Untuk menghasilkan deskripsi teks dari konteks yang diperoleh melalui proses retrieval, sistem ini memanfaatkan decoder Large Language Model (LLM) lokal berbasis Gemma 2. File get_llms.py bertugas menginisialisasi model tersebut dengan konfigurasi seperti nama model dan suhu (temperature) untuk kontrol kreativitas output. Fungsi get_llms() akan mengembalikan instansi model yang siap digunakan untuk inference. Model ini dikembangkan agar dapat berjalan secara lokal dalam sistem NeutraGem dan mendukung interaksi real-time pada perangkat edge seperti Raspberry Pi.

Tabel 3.5 Pseudocode Inisialisasi Model LLM Lokal untuk Inference RAG

```

# Deskripsi: Inisialisasi model LLM lokal berbasis Gemma 2 untuk inference
# Input: nama model (str), suhu/temperature (float)
# Output: objek model LLM siap digunakan untuk generate_response()
# =====

01| IMPORT os
02| FROM local_llm_engine import LocalGemmaChatModel # LLM lokal (misal:
Gemma 2 wrapper)

03|
04| FUNCTION get_llms(model_name: str = "gemma-2b-instruct", temperature:
float = 0.0):
05|     """
06|         Mengembalikan instansi model LLM lokal dengan parameter
konfigurasi.
07|     Args:
08|         model_name: Nama model lokal yang akan digunakan (misal:
"gemma-2b-instruct")
09|         temperature: Nilai untuk mengatur kreativitas output (semakin
tinggi, semakin kreatif)
10|     Returns:
11|         Objek LLM lokal yang dapat digunakan untuk inference
12|     """
13|
14|     model = LocalGemmaChatModel(
15|         model_name = model_name,
16|         temperature = temperature,
17|         streaming = True # Dukung interaksi real-time
18|     )
19|
20|     RETURN model

21|
22| # Contoh penggunaan jika dijalankan langsung
23| IF __name__ == "__main__":
24|     llm = get_llms("gemma-2b-instruct", temperature=0.2)
25|     response = llm.invoke("Apa itu Gemma?")
26|     PRINT(f"Jawaban LLM:\n{response.content}")

```

File app_mcp_agent.py merupakan inti dari manajemen agent dan tools dalam sistem MCP. Di dalamnya terdapat sejumlah fungsi yang berfungsi untuk membuat, membaca, memperbarui, dan menghapus (CRUD) entitas agent serta relasinya dengan tools tertentu. Fungsi create_new_agent() memungkinkan sistem untuk mendaftarkan agent baru, sedangkan fungsi seperti get_tools() dan assign_tool_to_agent() digunakan untuk memetakan tools ke agent secara dinamis. Modul ini menghubungkan sistem ke backend FastAPI untuk penyimpanan dan manajemen data agent, dan bertindak sebagai API Client dalam arsitektur MCP.

Tabel 3.6 Pseudocode CRUD AI Agent dalam Ekosistem MCP

```
# Deskripsi: Modul untuk manajemen agent dalam ekosistem MCP
# Input: JSON payload agent baru, ID tools, ID agent
# Output: Status keberhasilan operasi create/read/update/delete agent

01| IMPORT requests
02| IMPORT json
03| FROM dotenv IMPORT load_dotenv
04| IMPORT os

05| load_dotenv() # Memuat variabel lingkungan dari file .env

06| # Base URL dari backend FastAPI yang menangani data agent dan tool
07| BASE_URL = os.getenv("AGENT_BACKEND_URL", "http://localhost:8000")

08|
09| # Membuat agent baru ke backend
10| FUNCTION create_new_agent(agent_data: dict) -> dict:
11|     """
12|         Mengirim request POST ke backend untuk membuat agent baru
13|         Args:
14|             agent_data: Dictionary berisi data agent (nama, deskripsi,
tools, dsb.)
15|         Returns:
16|             Response dari backend dalam format JSON
17|             """
18|         url = f"{BASE_URL}/agents/"
19|         response = requests.post(url, json=agent_data)
20|         RETURN response.json()

21|
22| # Mengambil daftar semua agent dari backend
23| FUNCTION get_all_agents() -> list:
24|     """
25|         Mengambil seluruh data agent dari backend
26|         Returns:
27|             List berisi semua agent dalam sistem
28|             """
29|         url = f"{BASE_URL}/agents/"
30|         response = requests.get(url)
31|         RETURN response.json()

32|
33| # Mengambil daftar tools yang tersedia
34| FUNCTION get_tools() -> list:
```

```

35|     """
36|     Mendapatkan daftar tools dari backend yang dapat dikaitkan ke
agent
37|     Returns:
38|         List berisi tools beserta metadatanya
39|     """
40|     url = f"{BASE_URL}/tools/"
41|     response = requests.get(url)
42|     RETURN response.json()

43|
44| # Mengaitkan tool ke agent (assign)
45| FUNCTION assign_tool_to_agent(agent_id: str, tool_id: str) -> dict:
46|     """
47|     Menghubungkan tool ke agent tertentu
48|     Args:
49|         agent_id: ID unik dari agent
50|         tool_id: ID unik dari tool yang ingin dihubungkan
51|     Returns:
52|         Status keberhasilan assign tool
53|     """
54|     url = f"{BASE_URL}/agents/{agent_id}/assign_tool/"
55|     payload = {"tool_id": tool_id}
56|     response = requests.post(url, json=payload)
57|     RETURN response.json()

58|
59| # Contoh penggunaan
60| IF __name__ == "__main__":
61|     agent_data = {
62|         "name": "caption_agent",
63|         "description": "Agent untuk image captioning",
64|         "status": "active"
65|     }
66|     new_agent = create_new_agent(agent_data)
67|     PRINT(f"Agent berhasil dibuat: {new_agent}")

```

File mcp_proxy_manager.py bertugas sebagai pengelola lifecycle tools-agent dalam ekosistem MCP. Komponen ini mengecek status setiap tool berdasarkan port yang dikonfigurasi, melakukan reload atau restart jika terjadi kegagalan, dan menyinkronkan informasi dengan backend. Alur utama program dimulai dari MCPProxyManager.start() yang menerima parameter perintah untuk menjalankan berbagai tools berdasarkan metadata yang sudah didefinisikan. File ini berperan sebagai jembatan antara status fisik tools di lapangan dan logika digital manajemen agent, sehingga inference engine selalu memiliki akses ke tools yang aktif dan berjalan dengan baik.

Tabel 3.7 Pseudocode Model Context Protocol Proxy Manager

```
# Deskripsi: Manajer eksekusi dan pemantauan tools MCP
# Input: Perintah eksekusi tools (cmdline list)
# Output: Status tools, proses logging, daftar aktif tools

01| IMPORT os, subprocess, threading, json, logging, time
02| IMPORT yaml

03| # Inisialisasi logging
04| logger = logging.getLogger(__name__)
05| logging.basicConfig(level=logging.INFO)

06| CLASS MCPProxyManager:
07|     # Konstruktor untuk inisialisasi internal state
08|     FUNCTION __init__(self):
09|         self.running_processes = {} # Mapping: port -> subprocess.Popen
object

10|     # Fungsi utama untuk menjalankan perintah
11|     FUNCTION start(self, arr_full_cmd: list):
12|         """
13|             Menjalankan semua tool dari daftar perintah konfigurasi
14|         """
15|         FOR cmd_path IN arr_full_cmd:
16|             self._start_process(cmd_path)

17|     # Menjalankan satu proses tool berdasarkan path YAML
18|     FUNCTION _start_process(self, yaml_path: str):
19|         WITH open(yaml_path, 'r') AS file:
20|             config = yaml.safe_load(file)

21|             tool_name = config["name"]
22|             port = config["port"]
23|             exec_cmd = config["exec"]

24|             logger.info(f"Menjalankan tool: {tool_name} di port: {port}")
25|             process = subprocess.Popen(
26|                 exec_cmd,
27|                 shell=True,
28|                 stdout=subprocess.PIPE,
29|                 stderr=subprocess.STDOUT,
30|                 text=True,
31|                 bufsize=1
32|             )
33|             self.running_processes[port] = process
```

```

34|         # Jalankan thread terpisah untuk membaca log
35|     FUNCTION read_log():
36|         FOR line IN iter(process.stdout.readline, ''):
37|             logger.info(f"[{tool_name}] {line.strip()}")
38|
39|         thread = threading.Thread(target=read_log)
40|         thread.daemon = True
41|         thread.start()
42|
43|     # Fungsi untuk memeriksa apakah proses masih berjalan
44|     FUNCTION check_status(self):
45|         FOR port, proc IN self.running_processes.items():
46|             IF proc.poll() IS NOT None:
47|                 logger.warning(f"Tool di port {port} telah berhenti.")
48|
49|     # Menghentikan semua proses saat MCP dimatikan
50|     FUNCTION stop_all(self):
51|         FOR port, proc IN self.running_processes.items():
52|             logger.info(f"Menghentikan tool di port {port}")
53|             proc.terminate()
54|
55|     # Eksekusi saat script dijalankan langsung
56| IF __name__ == "__main__":
57|     manager = MCPProxyManager()
58|
59|     # Contoh path file konfigurasi YAML untuk tools
60|     cmd = [
61|         "tools/image_captioning.yaml",
62|         "tools/navigation.yaml"
63|     ]
64|
65|     # Menjalankan semua tools
66|     manager.start(cmd)
67|
68|     # Monitoring setiap 10 detik
69|     TRY:
70|         WHILE True:
71|             manager.check_status()
72|             time.sleep(10)

```

File `check_tools_status.py` bertanggung jawab untuk melakukan monitoring berkala terhadap status semua tools dalam sistem. Menggunakan library seperti psutil, file ini mendeteksi apakah suatu tool sedang berjalan (aktif), tidak aktif, atau perlu di-restart. Fungsi `check_tools_status()` akan dipanggil setiap menit secara otomatis menggunakan schedule, dan akan memperbarui status tools ke database Supabase. Jika tool tidak ditemukan aktif, sistem ini juga akan mencoba me-restart tool tersebut secara otomatis di background. Dengan pendekatan ini, sistem inference MCP dapat memelihara ketersediaan tools dengan cara otonom dan tangguh terhadap gangguan.

Tabel 3.8 Pseudocode Monitoring Status Tools MCP

```
# Deskripsi: Monitoring periodik status tools MCP dan restart otomatis
# Input: Informasi konfigurasi tools dari database
# Output: Update status tool ke database dan eksekusi ulang jika mati

01| IMPORT os, time, re, copy, threading, subprocess, logging
02| IMPORT psutil, schedule
03| FROM datetime IMPORT datetime
04| FROM dotenv IMPORT load_dotenv
05| FROM supabase IMPORT create_client, Client

06| # Setup logging
07| logger = logging.getLogger(__name__)
08| logging.basicConfig(level=logging.INFO)

09| # Load variabel lingkungan
10| load_dotenv()

11| # Inisialisasi Supabase
12| SUPABASE_URL = os.getenv("SUPABASE_URL")
13| SUPABASE_KEY = os.getenv("SUPABASE_KEY")
14| supabase: Client = create_client(SUPABASE_URL, SUPABASE_KEY)

15| PREDEFINED_COMPANY_ID = "95901eaa-c08d-4b0a-a5d6-3063a622cb98"
16| _tools_cache = None
17| _last_cache_time = 0
18| _cache_ttl = 60
19| _tool_restart_attempts = {}
20| _restart_cooldown = 15 * 60

21| # Ambil info tools dari database (dengan cache)
22| FUNCTION get_tools_info():
23|     current_time = time.time()
24|     IF _tools_cache IS NOT None AND (current_time - _last_cache_time) < _cache_ttl:
25|         RETURN _tools_cache
26|     response =
27|     supabase.table('tools_with_decrypted_keys').select("*").execute()
28|     _tools_cache = response.data
29|     _last_cache_time = current_time
30|     RETURN _tools_cache
31| # Cek port aktif
32| FUNCTION get_ports_info(ports: list[int]) -> dict:
33|     result = {}
34|     FOR conn IN psutil.net_connections(kind='inet'):
35|         IF conn.status == psutil.CONN_LISTEN AND conn.laddr.port IN ports:
```

```

36|             cmdline = psutil.Process(pid).cmdline() IF pid ELSE []
37|             result[conn.laddr.port] = cmdline
38|         RETURN result

39| # Bersihkan spasi ganda
40| FUNCTION remove_double_space(text: str) -> str:
41|     RETURN re.sub(r'\s+', ' ', text).strip()

42| # Ambil port dari struktur nested
43| FUNCTION safe_get_port(row: dict) -> int OR None:
44|     TRY:
45|         RETURN int(row['versions'][0]['released']['port'])
46|     EXCEPT:
47|         RETURN None

48| # Update tools ke Supabase (batch)
49| FUNCTION update_tools_info_batch(updates: list[dict]):
50|     batch_size = 5
51|     FOR i IN range(0, len(updates), batch_size):
52|         batch = updates[i:i + batch_size]
53|                     supabase.table('tools').upsert(batch,
on_conflict="tool_id").execute()

54| # Start ulang tool jika tidak aktif
55| FUNCTION start_tool(tool: dict):
56|     version = tool['versions'][0]
57|     port = version['released']['port']
58|     args = version['released']['args']
59|     env = version['released'].get('env', {})
60|     env_str = " ".join(f"-{k} {v}" for k, v IN env.items())
61|     cmd = f"mcp-proxy --sse-port={port} {env_str}-- {args}"
62|             process = subprocess.Popen(cmd, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.STDOUT, text=True)
63|     FUNCTION read_output():
64|         FOR line IN iter(process.stdout.readline, ''):
65|             logger.info(f"[{tool['name']}] {line.strip()}")
66|     threading.Thread(target=read_output, daemon=True).start()

67| # Fungsi utama pengecekan status tools
68| FUNCTION check_tools_status():
69|     tools_info = get_tools_info()
70|     ports = [safe_get_port(t) for t IN tools_info IF safe_get_port(t)
IS NOT None]
71|     ports_info = get_ports_info(ports)
72|     updates, tools_to_start = [], []
73|     FOR row IN tools_info:

```

```

74|         tool_id = row['tool_id']
75|         port = safe_get_port(row)
76|         args = row['versions'][0]['released']['args']
77|         current_status = row.get("on_status", "")
78|         company_id = row['company_id']
79|         is_predefined = company_id == PREDEFINED_COMPANY_ID
80|         status = "Inactive"
81|         IF port IN ports_info:
82|             cmdline = remove_double_space(" ".join(ports_info[port]))
83|             status = "Online" IF remove_double_space(args) IN cmdline
84| ELSE f"Inactive, Port used by: {cmdline}"
85| ELSE:
86|     now = time.time()
87|     last = _tool_restart_attempts.get(tool_id, 0)
88|     IF now - last > _restart_coldown:
89|         tools_to_start.append(row)
90|         _tool_restart_attempts[tool_id] = now
91|     updates.append({
92|         "tool_id": tool_id,
93|         "on_status": status,
94|         "name": row["name"],
95|         "versions": row.get("versions")
96|     })
97|     update_tools_info_batch(updates)
98| FOR tool IN tools_to_start:
99|     start_tool(tool)
100| # Jadwalkan pemeriksaan tiap 1 menit
101| schedule.every(1).minutes.do(check_tools_status)
102| # Saat dijalankan langsung
103| IF __name__ == "__main__":
104|     threading.Thread(target=check_tools_status, daemon=True).start()
105|     WHILE True:
106|         schedule.run_pending()
107|         time.sleep(1)

```

3.1.4 Pengujian dan Evaluasi

Pada tahap akhir sistem, dilakukan proses pengujian dan evaluasi terhadap performa *NeutraGem Inference Engine* yang telah terintegrasi dalam ekosistem *Model Context Protocol (MCP)*. Evaluasi ini bertujuan untuk menguji seberapa responsif dan adaptif sistem terhadap permintaan pengguna, termasuk waktu inisialisasi model, jumlah langkah reasoning (rekursi), serta waktu respons keseluruhan. Dalam implementasinya, sistem menggunakan agent berbasis `create_react_agent` dari LangGraph yang dikombinasikan dengan model LLM lokal (Gemma 2) dan pengelolaan tools multimodal dari server MCP menggunakan MultiServerMCPCClient. Proses ini meniru perilaku *real-time multimodal decision-making* yang penting untuk aplikasi AI pada pengguna dengan keterbatasan penglihatan atau kognitif seperti tunanetra dan demensia.

Table 3.9 Pseudocode Pengujian MCP dan AI Agent

```

# Deskripsi: Pengujian akhir dan evaluasi inference engine multimodal berbasis MCP
# Input: Prompt pengguna untuk reasoning, tool MCP yang aktif, dan model Gemma lokal
# Output: Waktu inisialisasi, jumlah rekursi, waktu respons, dan respons akhir dari LLM

01| import asyncio
02| import os, time
03| import json
04| from dotenv import load_dotenv
05| from langchain_mcp_adapters.client import MultiServerMCPClient
06| from langgraph.prebuilt import create_react_agent
07| from langchain_community.llms import LlamaCpp
08| from langchain_core.runnables import RunnableConfig
09| from langchain_core.callbacks import BaseCallbackHandler

10| # Muat variabel lingkungan
11| load_dotenv()

12| # Fungsi untuk mengambil instance LLM lokal (Gemma 2)
13| def get_local_llm():
14|     return LlamaCpp(
15|         model_path=os.getenv("LLM_LOCAL_MODEL_PATH", "./models/gemma-
2b-instruct-q4.gguf"),
16|         temperature=0.1,
17|         max_tokens=512,
18|         n_ctx=2048,
19|         n_threads=6,
20|         verbose=True,
21|     )

22| # Fungsi utama evaluasi agent
23| async def main():
24|     start_model = time.time()
25|     model = get_local_llm()
26|     print(f"Model Gemma siap dalam: {time.time() - start_model:.4f} detik")

27|     async with MultiServerMCPClient(
28|         {
29|             "captioner": {
30|                 "url": "http://localhost:8080/sse",
31|                 "transport": "sse"
32|             }
33|         }
34|     ) as client:
35|         start_agent = time.time()
36|         tools = client.get_tools()

```

```

37|         agent = create_react_agent(model, tools)
38|         print(f"Agent MCP siap dalam: {time.time() - start_agent:.4f} detik")
39|
40|         # Ambil masukan pengguna
41|         user_query = input("\nMasukkan pertanyaan atau perintah Anda:\n> ")
42|         query = {"messages": user_query}
43|
44|         # Handler untuk rekursi (opsional)
45|         class RecursionTracker(BaseCallbackHandler):
46|             def __init__(self):
47|                 super().__init__()
48|                 self.count = 0
49|             def on_llm_start(self, *args, **kwargs):
50|                 self.count += 1
51|                 print(f"Langkah rekursi ke-{self.count}")
52|
53|             tracker = RecursionTracker()
54|             config = RunnableConfig(recursion_limit=25,
55|                                       callbacks=[tracker])
56|
57|             start_inference = time.time()
58|             response = await agent.ainvoke(query, config)
59|             print(f"\nWaktu respons: {time.time() - start_inference:.4f} detik")
60|             print(f"Total rekursi: {tracker.count}")
61|
62|             # Cetak hasil respons
63|             messages = response.get("messages", [])
64|             for message in messages:
65|                 if hasattr(message, "content"):
66|                     print("\nRespons Gemma:")
67|                     print("-" * 50)
68|                     print(message.content)
69|                     print("-" * 50)
70|
71|
72|
73|
74|
75|
76|
77|
78|
79|
80|
81|
82|
83|
84|
85|
86|
87|
88|
89|
90|
91|
92|
93|
94|
95|
96|
97|
98|
99|
100|
101|
102|
103|
104|
105|
106|
107|
108|
109|
110|
111|
112|
113|
114|
115|
116|
117|
118|
119|
120|
121|
122|
123|
124|
125|
126|
127|
128|
129|
130|
131|
132|
133|
134|
135|
136|
137|
138|
139|
140|
141|
142|
143|
144|
145|
146|
147|
148|
149|
150|
151|
152|
153|
154|
155|
156|
157|
158|
159|
160|
161|
162|
163|
164|
165|
166|
167|
168|
169|
170|
171|
172|
173|
174|
175|
176|
177|
178|
179|
180|
181|
182|
183|
184|
185|
186|
187|
188|
189|
190|
191|
192|
193|
194|
195|
196|
197|
198|
199|
200|
201|
202|
203|
204|
205|
206|
207|
208|
209|
210|
211|
212|
213|
214|
215|
216|
217|
218|
219|
220|
221|
222|
223|
224|
225|
226|
227|
228|
229|
230|
231|
232|
233|
234|
235|
236|
237|
238|
239|
240|
241|
242|
243|
244|
245|
246|
247|
248|
249|
250|
251|
252|
253|
254|
255|
256|
257|
258|
259|
260|
261|
262|
263|
264|
265|
266|
267|
268|
269|
270|
271|
272|
273|
274|
275|
276|
277|
278|
279|
280|
281|
282|
283|
284|
285|
286|
287|
288|
289|
290|
291|
292|
293|
294|
295|
296|
297|
298|
299|
300|
301|
302|
303|
304|
305|
306|
307|
308|
309|
310|
311|
312|
313|
314|
315|
316|
317|
318|
319|
320|
321|
322|
323|
324|
325|
326|
327|
328|
329|
330|
331|
332|
333|
334|
335|
336|
337|
338|
339|
340|
341|
342|
343|
344|
345|
346|
347|
348|
349|
350|
351|
352|
353|
354|
355|
356|
357|
358|
359|
360|
361|
362|
363|
364|
365|
366|
367|
368|
369|
370|
371|
372|
373|
374|
375|
376|
377|
378|
379|
380|
381|
382|
383|
384|
385|
386|
387|
388|
389|
390|
391|
392|
393|
394|
395|
396|
397|
398|
399|
400|
401|
402|
403|
404|
405|
406|
407|
408|
409|
410|
411|
412|
413|
414|
415|
416|
417|
418|
419|
420|
421|
422|
423|
424|
425|
426|
427|
428|
429|
430|
431|
432|
433|
434|
435|
436|
437|
438|
439|
440|
441|
442|
443|
444|
445|
446|
447|
448|
449|
450|
451|
452|
453|
454|
455|
456|
457|
458|
459|
460|
461|
462|
463|
464|
465|
466|
467|
468|
469|
470|
471|
472|
473|
474|
475|
476|
477|
478|
479|
480|
481|
482|
483|
484|
485|
486|
487|
488|
489|
490|
491|
492|
493|
494|
495|
496|
497|
498|
499|
500|
501|
502|
503|
504|
505|
506|
507|
508|
509|
510|
511|
512|
513|
514|
515|
516|
517|
518|
519|
520|
521|
522|
523|
524|
525|
526|
527|
528|
529|
530|
531|
532|
533|
534|
535|
536|
537|
538|
539|
540|
541|
542|
543|
544|
545|
546|
547|
548|
549|
550|
551|
552|
553|
554|
555|
556|
557|
558|
559|
5510|
5511|
5512|
5513|
5514|
5515|
5516|
5517|
5518|
5519|
5520|
5521|
5522|
5523|
5524|
5525|
5526|
5527|
5528|
5529|
5530|
5531|
5532|
5533|
5534|
5535|
5536|
5537|
5538|
5539|
5540|
5541|
5542|
5543|
5544|
5545|
5546|
5547|
5548|
5549|
5550|
5551|
5552|
5553|
5554|
5555|
5556|
5557|
5558|
5559|
55510|
55511|
55512|
55513|
55514|
55515|
55516|
55517|
55518|
55519|
55520|
55521|
55522|
55523|
55524|
55525|
55526|
55527|
55528|
55529|
55530|
55531|
55532|
55533|
55534|
55535|
55536|
55537|
55538|
55539|
55540|
55541|
55542|
55543|
55544|
55545|
55546|
55547|
55548|
55549|
555410|
555411|
555412|
555413|
555414|
555415|
555416|
555417|
555418|
555419|
555420|
555421|
555422|
555423|
555424|
555425|
555426|
555427|
555428|
555429|
555430|
555431|
555432|
555433|
555434|
555435|
555436|
555437|
555438|
555439|
555440|
555441|
555442|
555443|
555444|
555445|
555446|
555447|
555448|
555449|
555450|
555451|
555452|
555453|
555454|
555455|
555456|
555457|
555458|
555459|
555460|
555461|
555462|
555463|
555464|
555465|
555466|
555467|
555468|
555469|
555470|
555471|
555472|
555473|
555474|
555475|
555476|
555477|
555478|
555479|
555480|
555481|
555482|
555483|
555484|
555485|
555486|
555487|
555488|
555489|
555490|
555491|
555492|
555493|
555494|
555495|
555496|
555497|
555498|
555499|
5554100|
5554101|
5554102|
5554103|
5554104|
5554105|
5554106|
5554107|
5554108|
5554109|
5554110|
5554111|
5554112|
5554113|
5554114|
5554115|
5554116|
5554117|
5554118|
5554119|
55541100|
55541101|
55541102|
55541103|
55541104|
55541105|
55541106|
55541107|
55541108|
55541109|
55541110|
55541111|
55541112|
55541113|
55541114|
55541115|
55541116|
55541117|
55541118|
55541119|
555411100|
555411101|
555411102|
555411103|
555411104|
555411105|
555411106|
555411107|
555411108|
555411109|
555411110|
555411111|
555411112|
555411113|
555411114|
555411115|
555411116|
555411117|
555411118|
555411119|
5554111100|
5554111101|
5554111102|
5554111103|
5554111104|
5554111105|
5554111106|
5554111107|
5554111108|
5554111109|
5554111110|
5554111111|
5554111112|
5554111113|
5554111114|
5554111115|
5554111116|
5554111117|
5554111118|
5554111119|
55541111100|
55541111101|
55541111102|
55541111103|
55541111104|
55541111105|
55541111106|
55541111107|
55541111108|
55541111109|
55541111110|
55541111111|
55541111112|
55541111113|
55541111114|
55541111115|
55541111116|
55541111117|
55541111118|
55541111119|
555411111100|
555411111101|
555411111102|
555411111103|
555411111104|
555411111105|
555411111106|
555411111107|
555411111108|
555411111109|
555411111110|
555411111111|
555411111112|
555411111113|
555411111114|
555411111115|
555411111116|
555411111117|
555411111118|
555411111119|
5554111111100|
5554111111101|
5554111111102|
5554111111103|
5554111111104|
5554111111105|
5554111111106|
5554111111107|
5554111111108|
5554111111109|
5554111111110|
5554111111111|
5554111111112|
5554111111113|
5554111111114|
5554111111115|
5554111111116|
5554111111117|
5554111111118|
5554111111119|
55541111111100|
55541111111101|
55541111111102|
55541111111103|
55541111111104|
55541111111105|
55541111111106|
55541111111107|
55541111111108|
55541111111109|
55541111111110|
55541111111111|
55541111111112|
55541111111113|
55541111111114|
55541111111115|
55541111111116|
55541111111117|
55541111111118|
55541111111119|
555411111111100|
555411111111101|
555411111111102|
555411111111103|
555411111111104|
555411111111105|
555411111111106|
555411111111107|
555411111111108|
555411111111109|
555411111111110|
555411111111111|
555411111111112|
555411111111113|
555411111111114|
555411111111115|
555411111111116|
555411111111117|
555411111111118|
555411111111119|
5554111111111100|
5554111111111101|
5554111111111102|
5554111111111103|
5554111111111104|
5554111111111105|
5554111111111106|
5554111111111107|
5554111111111108|
5554111111111109|
5554111111111110|
5554111111111111|
5554111111111112|
5554111111111113|
5554111111111114|
5554111111111115|
5554111111111116|
5554111111111117|
5554111111111118|
5554111111111119|
55541111111111100|
55541111111111101|
55541111111111102|
55541111111111103|
55541111111111104|
55541111111111105|
55541111111111106|
55541111111111107|
55541111111111108|
55541111111111109|
55541111111111110|
55541111111111111|
55541111111111112|
55541111111111113|
55541111111111114|
55541111111111115|
55541111111111116|
55541111111111117|
55541111111111118|
55541111111111119|
555411111111111100|
555411111111111101|
555411111111111102|
555411111111111103|
555411111111111104|
555411111111111105|
555411111111111106|
555411111111111107|
555411111111111108|
555411111111111109|
555411111111111110|
555411111111111111|
555411111111111112|
555411111111111113|
555411111111111114|
555411111111111115|
555411111111111116|
555411111111111117|
555411111111111118|
555411111111111119|
5554111111111111100|
5554111111111111101|
5554111111111111102|
5554111111111111103|
5554111111111111104|
5554111111111111105|
5554111111111111106|
5554111111111111107|
5554111111111111108|
5554111111111111109|
5554111111111111110|
5554111111111111111|
5554111111111111112|
5554111111111111113|
5554111111111111114|
5554111111111111115|
5554111111111111116|
5554111111111111117|
5554111111111111118|
5554111111111111119|
55541111111111111100|
55541111111111111101|
55541111111111111102|
55541111111111111103|
55541111111111111104|
55541111111111111105|
55541111111111111106|
55541111111111111107|
55541111111111111108|
55541111111111111109|
55541111111111111110|
55541111111111111111|
55541111111111111112|
55541111111111111113|
55541111111111111114|
55541111111111111115|
55541111111111111116|
55541111111111111117|
55541111111111111118|
55541111111111111119|
555411111111111111100|
555411111111111111101|
555411111111111111102|
555411111111111111103|
555411111111111111104|
555411111111111111105|
555411111111111111106|
555411111111111111107|
555411111111111111108|
555411111111111111109|
555411111111111111110|
555411111111111111111|
555411111111111111112|
555411111111111111113|
555411111111111111114|
555411111111111111115|
555411111111111111116|
555411111111111111117|
555411111111111111118|
555411111111111111119|
5554111111111111111100|
5554111111111111111101|
5554111111111111111102|
5554111111111111111103|
5554111111111111111104|
5554111111111111111105|
5554111111111111111106|
5554111111111111111107|
5554111111111111111108|
5554111111111111111109|
5554111111111111111110|
5554111111111111111111|
5554111111111111111112|
5554111111111111111113|
5554111111111111111114|
5554111111111111111115|
5554111111111111111116|
5554111111111111111117|
5554111111111111111118|
5554111111111111111119|
55541111111111111111100|
55541111111111111111101|
55541111111111111111102|
55541111111111111111103|
55541111111111111111104|
55541111111111111111105|
55541111111111111111106|
55541111111111111111107|
55541111111111111111108|
55541111111111111111109|
55541111111111111111110|
55541111111111111111111|
55541111111111111111112|
55541111111111111111113|
55541111111111111111114|
55541111111111111111115|
55541111111111111111116|
55541111111111111111117|
55541111111111111111118|
55541111111111111111119|
555411111111111111111100|
555411111111111111111101|
555411111111111111111102|
555411111111111111111103|
555411111111111111111104|
555411111111111111111105|
555411111111111111111106|
555411111111111111111107|
555411111111111111111108|
555411111111111111111109|
555411111111111111111110|
555411111111111111111111|
555411111111111111111112|
555411111111111111111113|
555411111111111111111114|
555411111111111111111115|
555411111111111111111116|
555411111111111111111117|
555411111111111111111118|
555411111111111111111119|
5554111111111111111111100|
5554111111111111111111101|
5554111111111111111111102|
5554111111111111111111103|
5554111111111111111111104|
5554111111111111111111105|
5554111111111111111111106|
5554111111111111111111107|
5554111111111111111111108|
5554111111111111111111109|
5554111111111111111111110|
5554111111111111111111111|
5554111111111111111111112|
5554111111111111111111113|
5554111111111111111111114|
5554111111111111111111115|
5554111111111111111111116|
5554111111111111111111117|
5554111111111111111111118|
5554111111111111111111119|
55541111111111111111111100|
55541111111111111111111101|
55541111111111111111111102|
55541111111111111111111103|
55541111111111111111111104|
55541111111111111111111105|
55541111111111111111111106|
55541111111111111111111107|
55541111111111111111111108|
55541111111111111111111109|
55541111111111111111111110|
55541111111111111111111111|
55541111111111111111111112|
55541111111111111111111113|
55541111111111111111111114|
55541111111111111111111115|
55541111111111111111111116|
55541111111111111111111117|
55541111111111111111111118|
55541111111111111111111119|
555411111111111111111111100|
555411111111111111111111101|
555411111111111111111111102|
555411111111111111111111103|
555411111111111111111111104|
555411111111111111111111105|
555411111111111111111111106|
555411111111111111111111107|
555411111111111111111111108|
555411111111111111111111109|
555411111111111111111111110|
555411111111111111111111111|
555411111111111111111111112|
555411111111111111111111113|
555411111111111111111111114|
555411111111111111111111115|
555411111111111111111111116|
555411111111111111111111117|
555411111111111111111111118|
555411111111111111111111119|
5554111111111111111111111100|
5554111111111111111111111101|
5554111111111111111111111102|
5554111111111111111111111103|
5554111111111111111111111104|
5554111111
```

Table 3.10 Pseudocode Pengujian BLEU Score Image Captioning

```

# Deskripsi: Evaluasi hasil caption model dengan BLEU Score terhadap
referensi
# Input: Caption hasil inference dan ground truth dari Flickr8k
# Output: BLEU-1 sampai BLEU-4

01| IMPORT json, nltk
02|     FROM     nltk.translate.bleu_score      IMPORT      sentence_bleu,
SmoothingFunction
03| smoother = SmoothingFunction().method4
04|
05| # Baca file caption hasil inference
06| WITH open("results/generated_captions.json", "r") AS f:
07|     gen = json.load(f)
08| # Baca file referensi caption
09| WITH open("results/ground_truth_captions.json", "r") AS f:
10|     ref = json.load(f)
11|
12| # List untuk BLEU score semua gambar
13| b1, b2, b3, b4 = [], [], [], []
14|
15| FOR img_id IN gen:
16|     hyp = gen[img_id].lower().split()
17|     refs = [r.lower().split() FOR r IN ref.get(img_id, [])]
18|     IF refs:
19|         b1.append(sentence_bleu(refs, hyp, weights=(1,0,0,0),
smoothing_function=smoother))
20|         b2.append(sentence_bleu(refs, hyp, weights=(0.5,0.5,0,0),
smoothing_function=smoother))
21|         b3.append(sentence_bleu(refs, hyp,
weights=(0.33,0.33,0.33,0), smoothing_function=smoother))
22|         b4.append(sentence_bleu(refs, hyp,
weights=(0.25,0.25,0.25,0.25), smoothing_function=smoother))
23|
24| # Tampilkan skor rata-rata
25| PRINT("BLEU Evaluation Results")
26| PRINT("BLEU-1: ", sum(b1)/len(b1))
27| PRINT("BLEU-2: ", sum(b2)/len(b2))
28| PRINT("BLEU-3: ", sum(b3)/len(b3))
29| PRINT("BLEU-4: ", sum(b4)/len(b4))

```

JADWAL KEGIATAN

Dalam penyusunan rencana penelitian diperlukan jadwal yang terperinci untuk mengatur setiap tahapan, guna memastikan kelancaran dan penyelesaian tepat waktu sesuai dengan target yang telah ditetapkan. Berikut merupakan rencana kegiatan penelitian yang akan dilakukan :

NO	Nama Kegiatan	Minggu ke-																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	Studi Pustaka																		
2	Eksplorasi Topik yang diajukan																		
3	Eksplorasi Dataset yang akan digunakan																		
4	Analisis topik dengan dataset																		
5	Eksplorasi Metode																		
6	Pelaporan Kemajuan																		
7	Eksperimen Metode																		
8	Penyusunan Proposal																		
9	Proposal ACC																		
10	Pengumpulan Proposal																		

DAFTAR PUSTAKA

- Anthropic. (2023). Model context protocol: A standard for AI system integration. <https://modelcontextprotocol.io>
- Betz, G., Richardson, K., & Musen, M. A. (2021). Procedural reasoning networks for understanding mechanisms in physical systems. arXiv. <https://doi.org/10.48550/arXiv.2110.00088>
- Cahyawijaya, S., Lovenia, H., Koto, F., Putri, R. A., Dave, E., Lee, J., Shadieq, N., Cenggoro, W., Akbar, S. M., Mahendra, M. I., Putri, D. A., Wilie, B., Winata, G. I., Aji, A. F., Purwarianti, A., & Fung, P. (2024). Cendol: Open instruction-tuned generative large language models for Indonesian languages. arXiv preprint. <https://doi.org/10.48550/arXiv.2404.06138>
- Codelabs Academy. (2024, September 6). Memahami Skor BLEU di NLP: Mengevaluasi Kualitas Terjemahan. Diakses dari <https://codelabsacademy.com/id/blog/understanding-bleu-score-in-nlp-evaluating-translation-quality>
- Drori, I., Kates, L., Katz, Y., Sinha, K., Zou, J., Shen, Y., Guo, Y., Lepert, J., Cheng, X., Bruss, C. B., et al. (2023). Integrating large language models with scientific computing libraries. arXiv. <https://doi.org/10.48550/arXiv.2306.08999>
- Habler, I., Huang, K., & Kulkarni, P. (2025). Building a secure agentic AI application leveraging Google's A2A protocol. arXiv preprint. arXiv:2504.16902
- Herry. (2014, April). Mengukur Hasil Terjemahan dengan BLEU (Bilingual Evaluation Understudy). Diakses dari <http://www.herry.blog.untan.ac.id/2014/04/mengukur-hasil-terjemahan-dengan-bleu.html>
- Lee, J., Le, T., Chen, J., & Lee, D. (2023). Do language models plagiarize? In Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, & G. Houben (Eds.), Proceedings of the ACM Web Conference 2023 (WWW '23) (pp. 3637–3647). ACM.
- Liu, C. & Zhao, J. 2024. Resource Allocation for Stable LLM Training in Mobile Edge Computing. Proceedings of the 2024 International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc), 14-17 Oktober 2024, Athena, Yunani. pp. 1-10.
- Navastara, D.A., Ansori, D.B., Suciati, N. & Akbar, Z.F. 2023. Combination of DenseNet and BiLSTM Model for Indonesian Image Captioning. Proceedings of the 2023 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA), 28-30 November 2023, Surabaya, Indonesia. pp. 994-999.
- Safiya, K.M., Pandian, R. 2024. A real-time image captioning framework using computer vision to help the visually impaired. Multimed Tools Appl 83, 59413–59438
- Sharma, S., & Sodhi, B. (2023). Calculating originality of LLM-assisted source code. arXiv. <https://doi.org/10.48550/arXiv.2307.04492>
- Shiddiqi, A., Yogatama, E. & Navastara, D.A. 2023. Resource-Aware Video Streaming (RAViS) Framework for Object Detection System using Deep Learning Algorithm. MethodsX, 11.
- Szeider, S. (2024). MCP-Solver: Integrating Language Models with Constraint Programming Systems. Stefan Szeider Algorithms and Complexity Group.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824-24837. <https://doi.org/10.48550/arXiv.2201.11903>
- Yamauchi, R., Sonoda, S., Sannai, A., & Kumagai, W. (2023). LPML: LLM-prompting markup language for mathematical reasoning.

Yang, C., Wang, Z., Yan, R., Zhang, M., Guo, Y., & Lin, X. (2025). *A Survey of AI Agent Protocols*. arXiv preprint arXiv:2404.10697.
<https://doi.org/10.48550/arXiv.2404.10697>