TUGAS AKHIR

SKEMA SKRIPSI

PAICODE: AGENTIC AI BERBASIS CLI UNTUK OTOMASI AKTIVITAS PEMROGRAMAN DAN PENGEMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API

I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001

PROGRAM STUDI INFORMATIKA
PROGRAM SARJANA
FAKULTAS FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA
YOGYAKARTA
2025

TUGAS AKHIR

SKEMA SKRIPSI

PAICODE: AGENTIC AI BERBASIS CLI UNTUK OTOMASI AKTIVITAS PEMROGRAMAN DAN PENGEMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API

Diajukan sebagai salah satu syarat untuk menyelesaikan studi pada
Program Sarjana
Program Studi INFORMATIKA
Fakultas FAKULTAS TEKNOLOGI INFORMASI
Universitas Teknologi Digital Indonesia

Disusun Oleh

I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001

PROGRAM STUDI INFORMATIKA
PROGRAM SARJANA
FAKULTAS FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA
YOGYAKARTA
2025

HALAMAN PERSETUJUAN UJIAN TUGAS AKHIR

Judul	: PAICODE: AGENTIC AI BERBASIS CLI UNTUK OTOMASI AKTIVITAS PEMROGRAMAN DAN PENGEMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API
Nama	: I PUTU GEDE GILANG TEJA KRISHNA
NIM	: 225410001
Program Studi	: INFORMATIKA
Program	: Sarjana
Semester	: Ganjil
Tahun Akademik	: 2024/2025

Telah diperiksa dan disetujui untuk diujikan
di hadapan Dewan Penguji Tugas Akhir

Yogyakarta, 24 November 2025
Dosen Pembimbing,

Dr. Bambang Purnomosidi Dwi Putranto, S.E., Akt., S.Kom., MMSI
NIDN: 0505058801

HALAMAN PENGESAHAN

PAICODE: AGENTIC AI BERBASIS CLI UNTUK OTOMASI AKTIVITAS PEMROGRAMAN DAN PENGEMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API

Telah dipertahankan di depan Dewan Penguji dan dinyatakan diterima untuk memenuhi sebagian persyaratan guna memperoleh
Gelar Sarjana Komputer
Program Studi INFORMATIKA
Fakultas FAKULTAS TEKNOLOGI INFORMASI
Universitas Teknologi Digital Indonesia

Yogyakarta, 24 November 2025

Dewan Penguji	NIDN	Tandatangan
1. Wagito, S.T., M.T. (Ketua)	.....	.....
2. Dr. Bambang Purnomosidi Dwi Putranto, S.E., Akt., S.Kom., MMSI (Sekretaris)	.....	.....
3. Ariesta Damayanti, S.Kom., M.Cs. (Anggota)	.....	.....

Mengetahui
Ketua Program Studi INFORMATIKA

Dini Fakta Sari, S.T., M.T.
NIDN: .....

PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa naskah Tugas Akhir ini belum pernah diajukan untuk memperoleh gelar Sarjana Komputer di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara sah diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta, 24 November 2025

I PUTU GEDE GILANG TEJA KRISHNA
NIM: 225410001

HALAMAN PERSEMBAHAN

Tugas Akhir ini saya persembahkan kepada kedua orang tua tercinta yang telah memberikan doa, dukungan, dan kasih sayang yang tiada henti; seluruh keluarga besar yang senantiasa memberikan motivasi dan semangat; para guru dan dosen yang telah membimbing dan memberikan ilmu yang bermanfaat; serta seluruh teman-teman di kampus dan rekan seperjuangan UTDI THE ARCADE.

PRAKATA

Puji syukur ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **PAICODE: AGENTIC AI BERBASIS CLI UNTUK OTOMASI AKTIVITAS PEMROGRAMAN DAN PENGEMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API**. Tugas Akhir ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer pada Program Studi INFORMATIKA, FAKULTAS TEKNOLOGI INFORMASI, UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA.

Penulis menyadari bahwa penyelesaian Tugas Akhir ini tidak lepas dari bantuan, bimbingan, dan dukungan berbagai pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih dan penghargaan yang setinggi-tingginya kepada semua pihak yang telah membantu. Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala rahmat, kesehatan, dan kemudahan yang diberikan selama proses penelitian. Ucapan terima kasih juga penulis sampaikan kepada orang tua dan keluarga yang senantiasa memberikan doa, dukungan moral, dan motivasi yang tiada henti.

Penulis secara khusus menyampaikan terima kasih kepada Bapak Dr. Bambang Purnomosidi Dwi Putranto, S.E., Akt., S.Kom., MMSI selaku dosen pembimbing yang telah memberikan bimbingan, arahan, dan masukan yang sangat berharga selama penyusunan Tugas Akhir ini. Terima kasih juga kepada seluruh dosen dan staf FAKULTAS TEKNOLOGI INFORMASI yang telah memberikan ilmu, fasilitas, dan dukungan selama masa perkuliahan, serta rekan-rekan mahasiswa dan teman-teman di kampus yang telah memberikan bantuan, diskusi, dan semangat selama proses penelitian yang tidak dapat penulis sebutkan satu per satu.

Penulis menyadari bahwa Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun untuk perbaikan di masa mendatang. Semoga Tugas Akhir ini dapat memberikan manfaat bagi pembaca dan perkembangan ilmu pengetahuan.

Yogyakarta, 24 November 2025

Penulis

Daftar Isi

Daftar Gambar

Daftar Tabel

INTISARI

Penelitian ini mengusulkan **Paicode**, sebuah agen AI berbasis Command Line Interface (CLI) untuk membantu proses pengembangan perangkat lunak secara interaktif dengan arsitektur *Single-Shot Intelligence*. Sistem berjalan pada lingkungan terminal lokal dan melakukan **operasi berkas tingkat-aplikasi di ruang kerja proyek (project workspace)**; namun **mengirimkan cuplikan kode/konteks ke layanan LLM (Gemini) melalui API** untuk keperluan inferensi. Oleh karena itu, aspek privasi dan kerahasiaan kode **bergantung pada kebijakan penyedia API**, sedangkan pengamanan lokal difokuskan pada kebijakan *path security*. Himpunan perintah yang disediakan (mis. `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`) memungkinkan agen mengobservasi proyek, memanipulasi berkas, dan memodifikasi kode secara terarah dengan sistem perubahan berbasis *diff*.

Arsitektur *Single-Shot Intelligence* mengoptimalkan efisiensi dengan sistem panggilan API yang terdiri dari: (1) klasifikasi intensi, (2) acknowledgment dinamis, (3) fase perencanaan untuk analisis mendalam dan perencanaan komprehensif dalam format JSON, (4) fase eksekusi adaptif yang dapat berjalan dalam 1-3 subfase berdasarkan kompleksitas tugas, dan (5) saran langkah berikutnya. Sistem mencakup manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key, *interrupt handling* (Ctrl+C), dan pencatatan sesi ke `.pai_history`.

Metode yang digunakan adalah *Research and Development* (R&D) dengan pendekatan *prototyping* iteratif. Evaluasi dilakukan melalui skenario tugas representatif, dengan metrik efisiensi (jumlah panggilan API), ketepatan hasil (kompilasi/eksekusi), dan kepatuhan keamanan *path*. Hasil menunjukkan bahwa agen *stateful* dengan arsitektur *Single-Shot Intelligence* dan pembatasan perubahan berbasis *diff* dengan threshold ganda (500 baris absolut dan 50% ratio maksimal) memudahkan pengembangan bertahap sambil menekan risiko penimpaan berkas. Sistem eksekusi adaptif dengan 1-3 subfase terbukti lebih efisien dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang, dengan tetap mempertahankan kualitas hasil yang optimal.

Kata kunci: agentic AI, CLI, LLM, API, Single-Shot Intelligence, keamanan *path*, pengembangan perangkat lunak.

ABSTRACT

This thesis presents **Paicode**, an agentic AI for the Command Line Interface (CLI) that assists software development through interactive, stateful workflows with a *Single-Shot Intelligence* architecture. The system runs on a local terminal and performs **application-level file operations within the project workspace**, while **sending code/context snippets to an external LLM (Gemini) via API** for inference. Consequently, privacy and confidentiality **depend on the provider's policy**, whereas local safeguards focus on path-security policies. A compact set of commands (e.g., `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`) enables the agent to observe the project, manipulate files, and apply targeted code modifications with *diff*-based change system.

The *Single-Shot Intelligence* architecture optimizes efficiency through an API call system consisting of: (1) intent classification, (2) dynamic acknowledgment, (3) planning phase for deep analysis and comprehensive JSON-based planning, (4) adaptive execution phase that can run in 1-3 sub-phases based on task complexity, and (5) next-step suggestions. The system includes single API key management with automatic migration from multi-key systems, *interrupt handling* (Ctrl+C), and session logging to `.pai_history`.

We adopt a Research and Development approach with iterative prototyping. The evaluation uses representative programming scenarios and measures efficiency (API call count), correctness (build/run), and security compliance. Results indicate that a stateful agent with *Single-Shot Intelligence* and *diff*-based change constraints with dual thresholds (500-line absolute and 50% maximum ratio) facilitates incremental development while reducing the risk of unintended overwrites. The adaptive execution system with 1-3 sub-phases proves more efficient than traditional approaches requiring multiple repetitive API calls, while maintaining optimal result quality.

Keywords: agentic AI, CLI, LLM, API, Single-Shot Intelligence, path security, software engineering.

BAB 1

Pendahuluan

1.1 Latar Belakang

Perkembangan *Large Language Model* (LLM) telah mendorong lahirnya beragam asisten pemrograman yang mampu membantu pengembang perangkat lunak dalam menulis, meninjau, dan memodifikasi kode. Meskipun demikian, sebagian besar asisten tersebut beroperasi sebagai ekstensi editor atau layanan berbasis *cloud* yang menyimpan, memproses, atau melatih dari data pengguna. Kondisi ini menimbulkan kekhawatiran terkait privasi, kendali atas data, serta ketergantungan pada antarmuka tertentu.

Di sisi lain, *Command Line Interface* (CLI) tetap menjadi lingkungan kerja yang penting bagi banyak pengembang karena sifatnya yang ringan, dapat diotomasi, dan mudah diintegrasikan dengan beragam alat. Integrasi kemampuan agen cerdas yang *stateful* dan *proactive* ke dalam CLI berpotensi mempercepat proses pengembangan perangkat lunak. Dalam konteks Paicode, sistem berjalan pada terminal lokal dan mengeksekusi tindakan langsung pada **berkas proyek di workspace**; namun, cuplikan kode/konteks **dikirim ke layanan LLM melalui API** untuk keperluan inferensi [2, 7, 1]. Dengan demikian, aspek privasi/kerahasiaan kode **bergantung pada kebijakan penyedia API**, sementara pengamanan di sisi lokal difokuskan pada kebijakan *path security* (keamanan *path*) dan pembatasan perubahan berbasis *diff*.

Penelitian ini menghadirkan **Paicode**, sebuah agen AI berbasis CLI yang dirancang untuk membantu proses pengembangan perangkat lunak secara interaktif dengan arsitektur *Single-Shot Intelligence*. Paicode mampu: (i) mengobservasi struktur proyek (`TREE`, `LIST_PATH`); (ii) membaca dan menulis berkas proyek (`READ`, `WRITE`); (iii) memodifikasi kode secara terarah dengan sistem perubahan berbasis *diff* dengan threshold ganda: 500 baris absolut dan 50% ratio maksimal (`MODIFY`); (iv) menegakkan kebijakan keamanan *path* pada berkas proyek (memblokir akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`); (v) melakukan klasifikasi intensi pengguna (*chat* vs *task*); (vi) mengoptimalkan efisiensi dengan sistem *Single-Shot Intelligence* yang mencakup *acknowledgment* dinamis, perencanaan JSON, dan eksekusi adaptif 1-3 subfase; serta (vii) menyediakan penanganan interupsi (*interrupt handling*) untuk kontrol

sesi yang lebih baik. Sistem diimplementasikan pada lingkungan Ubuntu dengan bahasa pemrograman Python, pengelolaan dependensi melalui pip dan virtual environment, manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key, dan menggunakan API Gemini sebagai LLM.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah yang diajukan adalah:

Bagaimana merancang, mengimplementasikan, dan mengevaluasi agen AI berbasis CLI dengan arsitektur Single-Shot Intelligence yang mampu mengotomasi aktivitas pemrograman secara aman melalui kebijakan path security dan pembatasan perubahan berbasis diff, serta terintegrasi dengan LLM melalui API?

1.3 Ruang Lingkup

Agar fokus penelitian terjaga dan implementasi dapat dilakukan secara terukur, batasan-batasan berikut ditetapkan:

- Lingkungan target adalah sistem operasi Ubuntu (Linux) dengan antarmuka CLI.
- Bahasa pemrograman utama adalah Python; contoh dan skenario uji berfokus pada ekosistem Python/Unix.
- Layanan LLM eksternal menggunakan API Gemini; kualitas respons bergantung pada model dan tidak menjadi ruang lingkup untuk dioptimasi ulang.
- Dukungan multi-pengguna, kolaborasi real-time, dan integrasi langsung dengan editor tidak dibahas pada versi ini.
- Aspek visual seperti diagram dan ilustrasi antarmuka ditunda pada tahap akhir; fokus laporan adalah narasi dan hasil teknis.

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah membangun dan mengevaluasi sebuah agen AI berbasis CLI yang dapat membantu pengembang dalam proses pemrograman

secara interaktif dengan arsitektur *Single-Shot Intelligence*. Secara khusus, penelitian menargetkan:

1. Merancang arsitektur Paicode yang mencakup modul agen dengan *Single-Shot Intelligence* (klasifikasi intensi, fase perencanaan, dan fase eksekusi dalam 2 panggilan API), jembatan LLM dengan manajemen API key tunggal, antarmuka CLI dengan *interrupt handling*, lapisan keamanan *path* pada berkas proyek, serta komponen tampilan terminal berbasis `rich`.

2. Mengimplementasikan kemampuan observasi proyek, manipulasi berkas, dan modifikasi kode terarah dengan mekanisme *diff*-aware yang mencegah penimpaan berkas tidak diinginkan dan memblokir akses ke direktori sensitif.

3. Mengintegrasikan fitur-fitur interaktif seperti pencatatan sesi ke `.pai_history`, penanganan interupsi (Ctrl+C), dan antarmuka terminal yang responsif dengan dukungan input multiline.

4. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur efisiensi panggilan API, ketepatan hasil, serta kepatuhan keamanan *path*.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

- **Akademis:** menyediakan studi kasus dan arsitektur rujukan untuk pengembangan agen AI berbasis CLI dengan integrasi LLM melalui API, serta memperkaya literatur mengenai integrasi LLM dalam alur kerja rekayasa perangkat lunak.

- **Praktis:** menghadirkan alat bantu pengembangan perangkat lunak dengan kelebihan spesifik sebagai berikut:

    1. **Efisiensi Biaya dan Token**: Menggunakan arsitektur *Single-Shot Intelligence* yang memadatkan proses perencanaan dan eksekusi menjadi dua panggilan utama, mengurangi biaya API dibandingkan agen berbasis *chat-loop* konvensional.

2. **Keamanan Terkendali**: Menerapkan kebijakan keamanan *path* (path security) yang memblokir akses ke direktori sensitif (seperti `.git`, `.env`) dan mekanisme modifikasi berbasis *diff* untuk mencegah perubahan destruktif masif.
3. **Fleksibilitas Lingkungan**: Beroperasi sebagai utilitas CLI yang ringan dan agnostik terhadap editor kode (IDE-agnostic), sehingga dapat digunakan di server tanpa antarmuka grafis (headless) maupun sebagai pendamping editor apa pun di OS berbasis Linux.

1.6 Sistematika Penulisan

Laporan tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I PENDAHULUAN

Memuat latar belakang, rumusan masalah, ruang lingkup, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA DAN DASAR TEORI

Memuat tinjauan pustaka dari penelitian terdahulu yang relevan serta dasar teori yang mendukung penelitian ini.

BAB III METODE PENELITIAN

Menjelaskan bahan, peralatan, prosedur penelitian, serta analisis dan perancangan sistem.

BAB IV IMPLEMENTASI DAN PEMBAHASAN

Menguraikan proses implementasi sistem, hasil uji coba, dan pembahasan mengenai hasil yang diperoleh.

BAB V PENUTUP

Berisi kesimpulan dari penelitian dan saran untuk pengembangan selanjutnya.

BAB 2

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Perkembangan alat bantu pemrograman berbasis AI berkembang pesat dalam beberapa tahun terakhir. Berikut adalah tinjauan terhadap beberapa solusi *state-of-the-art* yang relevan dengan penelitian ini:

2.1.1 AI Coding Assistant Terintegrasi (IDE-based)

GitHub Copilot [4] merupakan contoh paling prominen dari asisten pemrograman yang terintegrasi langsung ke dalam lingkungan pengembangan (IDE) seperti VS Code. Copilot unggul dalam memberikan saran *autocomplete* real-time dan fungsi obrolan kontekstual. Namun, pendekatannya sangat bergantung pada antarmuka editor visual dan beroperasi sebagai "pilot pendamping" (copilot) alih-alih agen otonom yang dapat melakukan tugas kompleks lintas berkas secara mandiri tanpa intervensi pengguna untuk setiap langkahnya.

2.1.2 CLI-based AI Chat Tools

Alat seperti Aider [3] membawa kemampuan LLM ke dalam terminal (CLI). Aider memungkinkan pengguna untuk melakukan *pair programming* dengan LLM langsung di terminal dan menerapkan perubahan pada git repository. Pendekatan ini mirip dengan Paicode dalam hal antarmuka berbasis teks. Perbedaannya, Paicode menekankan pada arsitektur *Single-Shot Intelligence* dengan fase perencanaan JSON eksplisit sebelum eksekusi, serta penerapan kebijakan keamanan *path* yang ketat untuk lingkungan korporasi atau sensitif, sedangkan banyak alat CLI lain berfokus pada kecepatan interaksi *chat-apply* langsung.

2.1.3 Autonomous Software Engineers

Proyek seperti OpenDevin [8] dan SWE-agent [5] bertujuan menciptakan agen yang sepenuhnya otonom, mampu menyelesaikan isu GitHub dari awal hingga

akhir tanpa interaksi manusia. Meskipun sangat canggih, pendekatan ini seringkali memerlukan akses sumber daya yang besar (Docker container penuh) dan kompleksitas tinggi untuk penyiapan. Paicode mengambil posisi tengah (middle-ground) dengan menyediakan agen *semi-autonomous* yang ringan (*lightweight*), berjalan native di OS tanpa kontainer berat, namun tetap memiliki kemampuan perencanaan (*planning*) untuk tugas multi-langkah.

2.1.4 Posisi Paicode

Dibandingkan dengan solusi di atas, Paicode menawarkan kebaruan pada kombinasi arsitektur *local-first* yang ringan namun terstruktur:

1. **Keamanan Terkendali**: Tidak seperti agen otonom penuh yang sering berjalan di sandboxed container karena risiko tinggi, Paicode dirancang aman untuk berjalan di *host* utama berkat *path security policy* dan *diff-based guardrails*.

2. **Efisiensi Token**: Dengan arsitektur perencanaan *single-shot*, Paicode mengurangi *round-trip* percakapan yang tidak perlu, berbeda dengan model *chat* standar.

3. **Transparansi Rencana**: Pengguna dapat melihat rencana aksi (dalam format JSON) sebelum eksekusi masif dilakukan, memberikan kontrol lebih baik daripada model *black-box*.

2.1.5 Perbandingan dengan Penelitian Sebelumnya

Tabel 2.1 merangkum perbedaan antara penelitian-penelitian terdahulu dengan penelitian yang akan dilakukan.

Dari Tabel 2.1 terlihat bahwa penelitian ini mengisi *gap* antara asisten pasif (seperti Copilot) dan agen otonom penuh (seperti OpenDevin) dengan menawarkan pendekatan *semi-autonomous* yang efisien, aman, dan transparan. Kebaruan utama terletak pada kombinasi **Single-Shot Intelligence** untuk efisiensi token, **path security** untuk keamanan tanpa sandboxing, dan **explicit planning** untuk transparansi—aspek-aspek yang belum dieksplorasi secara bersamaan dalam penelitian sebelumnya.

2.1.6 Posisi Penelitian

Kontribusi penelitian ini ditempatkan pada ranah agentic AI untuk pengembangan perangkat lunak dengan karakteristik sebagai berikut:

Tabel 2.1: Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan

Aspek	**Penelitian Terdahulu**	**Penelitian Ini (Paicode)**
Platform	IDE-based (Copilot), Web-based (ChatGPT Code Interpreter), Container-based (OpenDevin)	CLI native, berjalan langsung di terminal Linux tanpa container
Arsitektur Agen	Chat-loop iteratif (10-20 API calls) atau fully autonomous	Single-Shot Intelligence (2 API calls: planning + execution)
Keamanan Lokal	Sandboxed container (OpenDevin) atau tidak ada kontrol eksplisit (Copilot)	Path security policy + diff-based guardrails (threshold 500 baris, 50% ratio)
Transparansi	Black-box suggestions (Copilot) atau verbose logs (SWE-agent)	Explicit JSON planning phase dengan user approval
Efisiensi	High token consumption (chat-loop) atau resource-intensive (full containers)	Token-optimized (60-70% reduction) dan lightweight (native OS)
Interaktivitas	Passive suggestions (Copilot) atau fully autonomous (OpenDevin)	Semi-autonomous dengan interrupt handling (Ctrl+C)
Fokus Penelitian	General-purpose coding atau issue-solving automation	Secure, efficient, transparent automation untuk developer workflows

- **CLI lokal dengan integrasi LLM via API**: agen berjalan di terminal, tindakan langsung tercermin pada **berkas proyek di workspace**; sementara inferensi dilakukan oleh LLM eksternal sehingga kebijakan data mengikuti penyedia API.
- **Arsitektur Single-Shot Intelligence**: alur kerja efisien yang mengoptimalkan penggunaan API dengan tepat 2 panggilan (perencanaan dan eksekusi), menggantikan pendekatan tradisional yang memerlukan 10-20 panggilan API.
- **Manajemen API key tunggal**: sistem manajemen API key yang disederhanakan dengan migrasi otomatis dari sistem multi-key untuk kemudahan penggunaan.
- **Keamanan berkas**: kebijakan pelarangan akses *path* sensitif dan validasi *path* mencegah *path traversal* dan operasi berisiko pada direktori seperti `.git`, `venv`, dan `.env`.
- **Modifikasi terarah berbasis diff**: perintah `MODIFY` memanfaatkan sistem *diff*-aware untuk membatasi ruang perubahan dan mencegah penimpaan berkas tidak diinginkan.
- **Fitur interaktif**: *interrupt handling* (Ctrl+C) untuk menghentikan respons AI tanpa keluar dari sesi, pencatatan sesi lengkap ke `.pai_history`, dan antarmuka terminal responsif dengan dukungan input multiline.
- **Keterulangan eksperimen**: penggunaan pip, virtual environment, dan Makefile memudahkan replikasi lingkungan dan dokumentasi langkah instalasi.

2.2 Dasar Teori

Bagian ini membahas konsep yang menjadi landasan penelitian: *Command Line Interface* (CLI), agen kecerdasan buatan (AI Agent), *Large Language Model* (LLM), arsitektur dan kebijakan data (integrasi LLM melalui API dan implikasi privasi), *Single-Shot Intelligence* untuk agen interaktif, sistem klasifikasi intensi, serta perangkat bantu yang digunakan seperti pip dan virtual environment untuk manajemen dependensi, `rich` dan `prompt_toolkit` untuk antarmuka terminal.

2.2.1 Command Line Interface (CLI)

CLI adalah antarmuka berbasis teks yang memungkinkan pengguna berinteraksi dengan sistem melalui perintah. Kelebihan CLI meliputi otomasi yang mudah, konsumsi sumber daya yang rendah, dan integrasi sederhana dengan alat lain melalui skrip. Dalam konteks pengembangan perangkat lunak, CLI memfasilitasi alur kerja yang ringkas dan dapat direproduksi.

2.2.2 AI Agent

AI Agent (sering disebut *agentic AI* dalam literatur; selanjutnya disingkat "agen AI") dalam penelitian ini dipahami sebagai sistem yang mampu mengobservasi lingkungan (struktur proyek dan isi berkas), merencanakan tindakan (mis. membuat, membaca, memodifikasi berkas), serta mengevaluasi hasil untuk langkah berikutnya. Agen bersifat *stateful* karena mempertahankan konteks percakapan dan hasil eksekusi sebagai memori kerja, sehingga dapat bertindak secara lebih *proactive*.

Pada implementasi Paicode, agen menggunakan arsitektur *Single-Shot Intelligence* yang terdiri dari beberapa komponen: (1) klasifikasi intensi untuk membedakan percakapan dan tugas, (2) *acknowledgment* dinamis untuk konfirmasi pemahaman, (3) fase perencanaan dengan analisis mendalam dan perencanaan komprehensif dalam format JSON, (4) fase eksekusi adaptif yang dapat berjalan dalam 1-3 subfase berdasarkan kompleksitas tugas, dan (5) saran langkah berikutnya. Sistem ini mengoptimalkan efisiensi dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.

2.2.3 Large Language Model (LLM)

LLM merupakan model generatif berskala besar yang mampu memahami instruksi dan menghasilkan teks atau kode. Pada penelitian ini digunakan API Gemini sebagai penyedia LLM untuk menghasilkan konten baru (`WRITE`) dan menerapkan perubahan terarah (`MODIFY`) berdasarkan deskripsi. Prinsip kehati-hatian diterapkan dengan mekanisme pembatasan perubahan berbasis *diff* sehingga modifikasi tidak berskala besar tanpa kontrol [2, 7, 1, 10, 6, 9, 11].

2.2.4 Perbedaan LLM dan Agen AI

Pada skripsi ini penting untuk membedakan *Large Language Model* (LLM) dan *Agen AI*:

- **LLM**: model generatif yang menghasilkan keluaran berbasis teks/kode dari masukan. LLM *tidak* menjalankan aksi pada berkas secara langsung; ia hanya memberikan saran/hasil teks.
- **Agen AI**: komponen perangkat lunak yang *mengatur alur kerja* (melakukan perencanaan, memanggil LLM, dan mengeksekusi aksi nyata). Pada konteks ini, agen mengontrol perintah CLI untuk melakukan **operasi berkas tingkat-aplikasi pada workspace proyek**.
- **Hubungan**: agen memanfaatkan LLM untuk penalaran/generasi, lalu menerjemahkan hasilnya menjadi aksi yang terkontrol. Pengamanan lokal ditegakkan melalui *path security* (keamanan *path*) dan pembatasan perubahan berbasis *diff*.

2.2.5 Arsitektur dan Kebijakan Data

Paicode dijalankan pada terminal lokal dan melakukan tindakan langsung pada **berkas proyek di workspace**. Akan tetapi, untuk kebutuhan inferensi, cuplikan kode atau konteks **dikirim ke layanan LLM melalui API**. Implikasinya, privasi dan kerahasiaan kode **bergantung pada kebijakan penyedia API**. Pengamanan di sisi lokal diterapkan melalui kebijakan *path security* (keamanan *path*) serta pembatasan perubahan berbasis *diff* agar operasi berkas lebih terkendali.

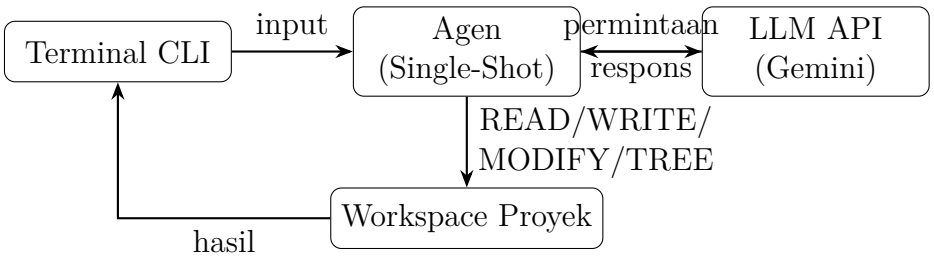
2.2.6 Manajemen Dependensi dengan pip dan Virtual Environment

Paicode menggunakan pendekatan manajemen dependensi tradisional dengan pip dan virtual environment Python. Berkas `requirements.txt` mendeskripsikan dependensi yang diperlukan, sementara Makefile menyediakan otomasi untuk pembuatan virtual environment dan instalasi dependensi. Pendekatan ini memudahkan replikasi lingkungan dan instalasi alat. Pada implementasi Paicode, dependensi utama meliputi `google-generativeai` (versi $\geq$ 0.5.4), `rich` (versi $\geq$ 13.7.1), `Pygments` (versi $\geq$ 2.16.0), dan `prompt_toolkit` (versi $\geq$ 3.0.43).

2.2.7 Antarmuka Terminal dengan `rich` dan `prompt_toolkit`

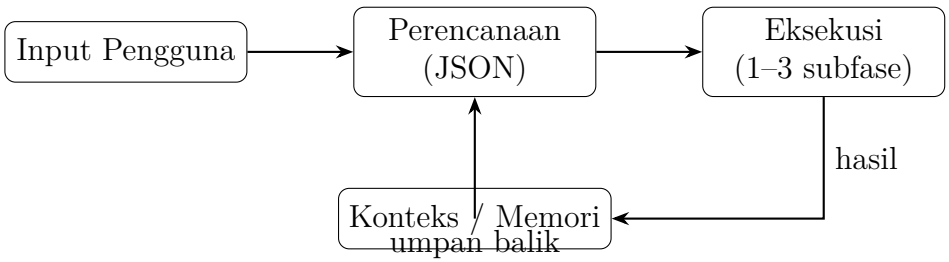
Paket `rich` dimanfaatkan untuk menyajikan hasil eksekusi secara terstruktur dan mudah dibaca (panel, warna, penyorotan sintaks, tabel, dan spinner

status). Penyajian output yang jelas mendukung pengalaman interaktif dan penelusuran hasil tindakan agen. Selain itu, Paicode juga mengintegrasikan `prompt_toolkit` (opsional) untuk pengalaman input yang lebih baik dengan dukungan multiline editing dan key bindings. Jika `prompt_toolkit` tidak tersedia, sistem akan fallback ke `rich.prompt.Prompt`.



Gambar 2.1: Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.

Pada Gambar 2.1 ditunjukkan pemetaan komponen utama (CLI, Agen, LLM, dan komponen workspace) beserta *control/data flow* antar komponen.



Gambar 2.2: Model interaksi *stateful* dan *feedback loop* pada sesi agen.

Pada Gambar 2.2 divisualisasikan hubungan antara masukan pengguna, perencanaan aksi, eksekusi alat, dan pembaruan konteks.

Pada Gambar 2.2 ditunjukkan perbedaan fokus dan pertukaran (trade-off) tingkat tinggi antar pendekatan.

Tabel 2.2: Ilustrasi komparasi konseptual antara pendekatan ekstensi editor, layanan daring, dan CLI dengan integrasi LLM via API.

	Ekstensi Editor	**Layanan Daring**
Integrasi	Sangat terintegrasi dengan IDE	Antarmuka web/remote
Konteks	Di editor, tergantung API	Di server; unggah/ sinkron
Privasi	Bergantung vendor	Bergantung vendor
Portabilitas	Terikat IDE	Perlu akses internet
	CLI + LLM via API (Paicode)	
Integrasi	Agen berjalan di terminal lokal; perubahan langsung pada workspace	
Konteks	Konteks lokal; cuplikan dikirim ke LLM via API	
Privasi	Tergantung kebijakan penyedia API; guardrail lokal	
Portabilitas	Editor-agnostic; cukup terminal Linux	

BAB 3

Metode Penelitian

3.1 Metode Pengembangan

Penelitian ini menggunakan pendekatan *Research and Development* (R&D) dengan strategi *prototyping* iteratif. Pemilihan metode ini didasarkan pada beberapa pertimbangan:

1. **Eksplorasi Desain Agen Stateful**: Berbeda dengan aplikasi konvensional yang bersifat *stateless*, agen AI memerlukan manajemen konteks percakapan dan memori kerja yang kompleks. Pendekatan *prototyping* memungkinkan eksperimen cepat terhadap berbagai strategi manajemen state (misalnya, ukuran context window, format log sesi) tanpa komitmen arsitektur jangka panjang.

2. **Validasi Asumsi Keamanan**: Kebijakan *path security* dan pembatasan *diff* merupakan mekanisme novel yang belum teruji di konteks agen CLI. Siklus iteratif memungkinkan identifikasi edge case (seperti symbolic links, path traversal attacks) melalui pengujian langsung, yang sulit diprediksi hanya dari analisis teoritis.

3. **Optimasi Efisiensi Token**: Arsitektur *Single-Shot Intelligence* dikembangkan melalui iterasi bertahap—dimulai dari model *chat-loop* konvensional (10-20 panggilan API per tugas), kemudian dipadatkan menjadi sistem 2-panggilan melalui eksperimen empiris terhadap berbagai strategi prompt engineering.

3.1.1 Trade-off Metodologis

Pendekatan *prototyping* dipilih dibandingkan metode waterfall atau agile penuh dengan pertimbangan trade-off berikut:

- **Kelebihan**: Fleksibilitas tinggi untuk mengubah desain berdasarkan temuan empiris; cocok untuk domain yang belum mature (agentic AI untuk CLI); memungkinkan validasi konsep sebelum investasi besar pada infrastruktur.

- **Kekurangan**: Dokumentasi arsitektur dapat tertinggal jika iterasi terlalu cepat; risiko *scope creep* jika tidak ada batasan jelas per iterasi; potensi *technical debt* jika refactoring tidak dilakukan secara disiplin.

- **Mitigasi**: Setiap iterasi dibatasi pada satu fitur utama (misalnya, iterasi 1: path security; iterasi 2: diff-aware modification; iterasi 3: Single-Shot Intelligence); dokumentasi arsitektur diperbarui setelah setiap iterasi stabil; code review dilakukan sebelum merge ke branch utama.

3.2 Arsitektur Sistem

Arsitektur Paicode dirancang modular dan berlapis, dengan pembagian tanggung jawab yang jelas:

- **Antarmuka CLI (`cli.py`)**: titik masuk perintah `pai` dan pengelola argumen (subperintah `auto`, `config`). Mendukung parameter `-model` dan `-temperature` untuk konfigurasi runtime LLM. Secara default, CLI memanggil sesi interaktif agen.

- **Agen (`agent.py`)**: mengimplementasikan *Single-Shot Intelligence* yang mencakup: (1) klasifikasi intensi (*chat* vs *task*), (2) *acknowledgment* dinamis, (3) fase perencanaan untuk analisis mendalam dalam format JSON, (4) fase eksekusi adaptif dengan 1-3 subfase berdasarkan kompleksitas, dan (5) saran langkah berikutnya. Menyediakan 10 perintah: `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`, `MKDIR`, `TOUCH`, `RM`, `MV`, `FINISH`. Mengelola memori percakapan dengan pencatatan sesi ke `.pai_history`.

- **Jembatan LLM (`llm.py`)**: menangani konfigurasi API Gemini dengan manajemen API key tunggal. Membersihkan output dari markdown artifacts, menyediakan status spinner saat LLM berpikir, dan mengoptimalkan penggunaan token dengan sistem 2-panggilan API.

- **Manajemen Konfigurasi (`config.py`)**: menyimpan dan mengelola API key tunggal dalam format JSON di `/.config/pai-code/credentials.json` dengan izin berkas 0o600 (read-write owner only). Validasi API key Google (harus dimulai dengan "AIza" dan minimal 20 karakter). Mendukung operasi: `set`, `show`, `remove`, `validate`, dan migrasi otomatis dari sistem multi-key.

- **Pengatur Workspace (`workspace.py`)**: bertindak sebagai *workspace controller* yang menyediakan fungsi-fungsi terpusat untuk menjalankan operasi tingkat-aplikasi pada ruang kerja proyek. Sebelum aksi dieksekusi, modul ini menegakkan kebijakan *path security* (normalisasi, verifikasi akar, dan deny-list direktori sensitif seperti `.env`, `.git`, `venv`, `__pycache__`, `.pai_history`, `.idea`, `.vscode`). Sistem modifikasi berbasis *diff* dengan threshold 500 baris dan ratio maksimal 50% (dapat dikonfigurasi via `PAI_MODIFY_THRESHOLD` dan `PAI_MODIFY_MAX_RATIO`) mencegah penimpaan berkas tidak diinginkan dengan atomic write menggunakan tempfile.

- **Tampilan Terminal (`ui.py`)**: penyajian hasil eksekusi menggunakan `rich` (panel, warna, tabel, penyorotan sintaks, spinner status). Mendukung `prompt_toolkit` (opsional) untuk input multiline yang lebih baik.

Alur data tipikal dengan *Single-Shot Intelligence*: masukan pengguna (CLI) → klasifikasi intensi → *acknowledgment* dinamis → fase perencanaan (analisis JSON) → fase eksekusi adaptif (1-3 subfase) → saran langkah berikutnya → pencatatan konteks sebagai memori percakapan.

3.3 Visualisasi Metodologi

Bagian ini menyajikan visualisasi konsep menggunakan tabel dan daftar terstruktur berbasis LaTeX.

Tabel 3.1: Modul dan Dependensi Komponen Paicode

Komponen	**Deskripsi dan Dependensi Utama**
CLI (`cli.py`)	Titik masuk perintah, parsing argumen (`-model`, `-temperature`); memanggil sesi agen. Bergantung pada modul `agent`, `config`, dan `llm`.
Agen (`agent.py`)	Implementasi *Single-Shot Intelligence*: klasifikasi intensi, *acknowledgment* dinamis, fase perencanaan JSON, fase eksekusi adaptif (1-3 subfase), dan saran langkah berikutnya. Mengelola memori percakapan, *interrupt handling* (Ctrl+C), dan pencatatan sesi ke `.pai_history`. Menyediakan 10 perintah workspace. Memanggil `llm`, `workspace`, `ui`.

Komponen	Deskripsi dan Dependensi Utama
LLM Bridge (`llm.py`)	Integrasi Gemini API (`google-generativeai`) dengan manajemen API key tunggal. Membersihkan markdown artifacts dari output LLM dan mengoptimalkan penggunaan token. Mengambil API key dari `config`.
Konfigurasi (`config.py`)	Manajemen API key tunggal dalam format JSON di `/.config/pai-code/credentials.json` dengan permission 0o600. Validasi API key Google (prefix "AIza", minimal 20 karakter). Operasi: `set`, `show`, `remove`, `validate`, dan migrasi otomatis dari sistem multi-key.
Pengatur Workspace (`workspace.py`)	*Workspace controller* dengan fungsi operasi workspace (baca/tulis, buat/hapus/pindah, tree/list path). Sistem modifikasi berbasis *diff* dengan threshold 500 baris dan ratio maksimal 50% (konfigurabel via environment variables) serta atomic write. Penegakan *path security* dengan deny-list 7 pola sensitif (`.env`, `.git`, `venv`, dll).
Terminal UI (`ui.py`)	Komponen TUI berbasis `rich`: panel, tema, syntax highlighting, tabel, spinner. Dukungan opsional `prompt_toolkit` untuk input multiline yang lebih baik.

Pada Tabel 3.1 ditunjukkan komponen utama dan interkoneksinya, sebagai acuan implementasi.

Tabel 3.2: Urutan Interaksi Sesi Agen dengan Single-Shot Intelligence

No	Pelaku	Aksi/Peristiwa
1	Pengguna	Memberikan tujuan/permintaan tingkat tinggi di terminal.
2	CLI	Meneruskan masukan ke agen; menyiapkan konteks sesi.
3	Agen	Melakukan klasifikasi intensi (*chat* vs *task*) menggunakan LLM. Jika *chat*, langsung berikan respons dan kembali ke langkah 1.
4	Agen	**Acknowledgment Dinamis**: Memberikan respons awal untuk mengakui dan memahami permintaan pengguna.

No	Pelaku	Aksi/Peristiwa
5	LLM	**Fase Perencanaan**: Melakukan analisis mendalam dan menghasilkan perencanaan komprehensif dalam format JSON dengan detail eksekusi.
6	Agen	Menampilkan hasil perencanaan dalam panel terstruktur dan memberikan konfirmasi sebelum eksekusi.
7	LLM	**Fase Eksekusi Adaptif**: Menentukan jumlah subfase (1-3) berdasarkan kompleksitas, kemudian melaksanakan implementasi cerdas.
8	Workspace/UI	Menjalankan operasi berkas (`READ`, `WRITE`, `MODIFY`, dll.) dengan *path security* dan sistem *diff*-aware, menampilkan hasil di terminal.
9	Agen	Memberikan status akhir (sukses/gagal) dan saran langkah berikutnya jika diperlukan.
10	Agen	Mencatat seluruh interaksi ke `.pai_history/session_YYYYMMDD_HHMMSS.log` sebagai memori (*stateful*).
11	Pengguna	Memberikan instruksi lanjutan; siklus berulang sampai `exit/quit`.

Pada Tabel 3.2 divisualisasikan aliran pesan yang terjadi selama satu putaran iterasi agen.

Alur Kebijakan Keamanan *Path*. Langkah-langkah validasi *path* diringkas berikut:

1. Normalisasi *path* target (`os.path.normpath`).
2. Resolusi *real path* relatif terhadap akar proyek; pastikan tetap berada di dalam akar proyek.
3. Pemeriksaan *deny-list* direktori/berkas sensitif: `.env`, `.git`, `venv`, `__pycache__`, `.pai_history`, `.idea`, `.vscode`.
4. Jika salah satu pemeriksaan gagal: batalkan operasi dan tampilkan pesan kesalahan.

Tabel 3.3: Rangkuman Validasi Keamanan *Path*

Tahap	**Detail Pemeriksaan**
Normalisasi	Gunakan fungsi normalisasi untuk menyingkirkan segmen berlebih (mis. `..`, duplikasi pemisah).
Verifikasi Root	Gabungkan terhadap akar proyek, lakukan `realpath`, dan validasi prefiks tetap di dalam akar proyek.
Deny-list	Tolak bila salah satu segmen *path* termasuk daftar sensitif (.env, .git, venv, dll.).
Penanganan Error	Batalkan operasi dan tampilkan pesan kesalahan yang informatif melalui TUI.

Pada Tabel 3.3 diperlihatkan langkah-langkah validasi *path* sebagai pengaman operasi berkas proyek.

3.4 Alat dan Lingkungan

Lingkungan dan alat yang digunakan:

- Sistem operasi: Ubuntu (Linux).
- Bahasa pemrograman: Python (`>= 3.10`, sesuai spesifikasi `setup.cfg`).
- Manajer dependensi: pip dan virtual environment; instalasi otomatis melalui Makefile dengan entry point CLI melalui skrip launcher di `$HOME/.local/bin/pai`.
- LLM: Google Gemini (model default `gemini-2.5-flash-lite`, temperature default 0.3, dapat dikonfigurasi via `PAI_MODEL` dan `PAI_TEMPERATURE`) melalui paket `google-generativeai` versi $\geq$ 0.5.4.
- TUI: `rich` (versi $\geq$ 13.7.1) untuk panel, warna, tabel, penyorotan sintaks, dan spinner status; `prompt_toolkit` (versi $\geq$ 3.0.43, opsional) untuk input multiline yang lebih baik.
- Dependensi tambahan: `Pygments` ($\geq$ 2.16.0) untuk syntax highlighting.
- Variabel lingkungan: `PAI_MODEL`, `PAI_TEMPERATURE`, `PAI_MODIFY_THRESHOLD`, `PAI_MODIFY_MAX_RATIO`, serta variabel noise suppression (`GRPC_VERBOSITY`, `GRPC_LOG_SEVERITY`, `ABSL_LOGGING_MIN_LOG_LEVEL`, dll) untuk menekan log gRPC/absl yang berisik.

- LaTeX: TeX Live (`texlive-latex-recommended`, `texlive-latex-extra`, dsb.) dan Makefile untuk kompilasi naskah.
- Kendali versi: Git dan GitHub.

3.5 Prosedur Penelitian

Prosedur penelitian dan evaluasi dirancang sebagai berikut:

1. **Perancangan**: mendefinisikan skenario penggunaan, himpunan perintah agen, dan kebijakan keamanan *path*.
2. **Implementasi**: membangun modul-modul inti (CLI, Agen, LLM, Workspace, UI) berikut mekanisme *diff*-aware untuk pembatasan perubahan.
3. **Eksperimen**: menjalankan serangkaian skenario pemrograman (mis. pembuatan struktur proyek, pembuatan/ pembacaan/ modifikasi berkas, refaktorisasi sederhana) dalam sesi interaktif.
4. **Pengumpulan Data**: merekam waktu penyelesaian tugas, jumlah langkah perintah, tingkat keberhasilan eksekusi, dan catatan kesalahan.
5. **Evaluasi**: membandingkan hasil dengan proses manual atau alat pembanding bila relevan, menggunakan metrik: (i) efisiensi (waktu dan langkah), (ii) ketepatan hasil (kompilasi/eksekusi kode), (iii) keamanan (kegagalan akses *path* sensitif), dan (iv) pengalaman pengguna (keterbacaan output).
6. **Analisis**: mengidentifikasi kelebihan, kekurangan, dan peluang peningkatan (mis. dukungan multi-LLM, integrasi editor, perluasan kebijakan keamanan).

BAB 4

Implementasi dan Pembahasan

4.1 Implementasi Paicode

Implementasi dilakukan menggunakan Python dengan manajemen dependensi pip dan virtual environment. Berkas `setup.cfg` mendefinisikan paket yang dibutuhkan beserta titik masuk CLI. Instalasi otomatis melalui Makefile. Langkah instalasi dan konfigurasi sebagai berikut.

4.1.1 Instalasi

1. Pastikan Python (`>= 3.10`) terpasang sesuai spesifikasi `setup.cfg`.
2. Masuk ke direktori `paicode/` dan jalankan:

Listing 4.1: Instalasi dependensi dengan Makefile

```
make install
```

4.1.2 Konfigurasi API Key

Paicode menggunakan manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key. Kunci disimpan secara aman dalam format JSON pada `/.config/pai-code/credentials.json` dengan izin berkas 0o600.

Listing 4.2: Manajemen API key tunggal Gemini

```
# Mengatur API key
pai config set <API_KEY_GEMINI>

# Melihat API key saat ini (masked)
pai config show

# Validasi API key
pai config validate

# Menghapus API key
pai config remove
```

Sistem akan secara otomatis melakukan migrasi dari konfigurasi multi-key lama (version 1) ke sistem single-key baru (version 2).

4.1.3 Menjalankan Agen

Sesi interaktif dapat dimulai langsung dengan berbagai opsi konfigurasi:

Listing 4.3: Menjalankan sesi agen interaktif

```
# Menjalankan dengan konfigurasi default
pai

# Menjalankan dengan model dan temperature tertentu
pai auto --model gemini-2.5-flash-lite --temperature 0.3

# Menggunakan variabel lingkungan untuk konfigurasi
export PAI_MODEL="gemini-2.5-flash-lite"
export PAI_TEMPERATURE="0.3"
export PAI_MODIFY_THRESHOLD="500"
export PAI_MODIFY_MAX_RATIO="0.5"
pai
```

Selama sesi, pengguna dapat:

- Menekan Ctrl+C sekali untuk menghentikan respons AI (sesi tetap aktif)
- Menekan Ctrl+C dua kali untuk keluar dari sesi
- Mengetik `exit` atau `quit` untuk mengakhiri sesi

4.2 Alur Interaksi dengan Single-Shot Intelligence

Alur kerja pada sesi interaktif mengikuti arsitektur *Single-Shot Intelligence*:

1. **Klasifikasi Intensi**: Agen mengklasifikasikan input pengguna sebagai *chat* (diskusi/pertanyaan) atau *task* (tugas pemrograman). Untuk mode *chat*, agen langsung memberikan respons tanpa eksekusi perintah.
2. **Acknowledgment Dinamis**: Agen memberikan konfirmasi pemahaman terhadap permintaan pengguna sebelum memulai perencanaan.

3. **Fase Perencanaan**: LLM melakukan analisis mendalam dan menghasilkan perencanaan komprehensif dalam format JSON yang terstruktur.

4. **Fase Eksekusi Adaptif**: Eksekusi perintah dalam 1-3 subfase berdasarkan kompleksitas tugas, menggunakan perintah workspace (`READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`, `MKDIR`, `TOUCH`, `RM`, `MV`, `FINISH`) dengan batasan threshold ganda (500 baris absolut dan 50% ratio maksimal).

5. **Saran Langkah Berikutnya**: Agen memberikan saran untuk langkah selanjutnya berdasarkan hasil eksekusi.

Operasi berkas dieksekusi melalui **Workspace Controller** (`workspace.py`) dengan penegakan kebijakan *path security* yang mencegah akses ke 7 pola direktori sensitif: `.env`, `.git`, `venv`, `__pycache__`, `.pai_history`, `.idea`, `.vscode`. Seluruh interaksi dicatat ke `.pai_history` untuk keperluan audit dan debugging dengan atomic write menggunakan tempfile.

4.2.1 Cuplikan Kode Kunci

Bagian ini menampilkan cuplikan kode inti yang merealisasikan arsitektur *Single-Shot Intelligence*. Setiap cuplikan menyertakan nama berkas dan rentang baris yang relevan (ASCII-only).

```
CRITICAL OUTPUT FORMAT:
Return a JSON object with this EXACT structure:

{{
  "analysis": {{
    "user_intent": "Clear description of what user wants",
    "target_identification": "SPECIFIC files and locations where target content likely exists",
    "multi_file_strategy": "Which files need to be checked to locate targets accurately",
    "validation_approach": "How you will verify targets exist before modification",
    "files_to_read": ["ALL files that might contain target content - be comprehensive"],
    "files_to_create": ["file1", "file2"],
    "files_to_modify": ["ONLY files confirmed to contain target content"],
```

```
    "risk_assessment": "Potential failure points and how to avoid them",
    "success_criteria": ["Specific, measurable criteria for success"]
  }},
  "execution_plan": {{
    "steps": [
      {{
        "step_number": 1,
        "action": "READ",
        "target": "filename",
        "purpose": "Locate and verify target content exists",
        "validation_criteria": "What content must be found to proceed",
        "expected_outcome": "Confirmed location of target content"
      }},
      {{
        "step_number": 2,
        "action": "MODIFY",
        "target": "filename",
        "purpose": "Apply changes to confirmed target location",
        "validation_criteria": "How to verify modification was successful",
        "expected_outcome": "Target content successfully modified"
      }}
    ],
    "command_format_reminder": "CRITICAL: Use exact command names: READ, WRITE, MODIFY, TREE, LIST_PATH, MKDIR, TOUCH, RM, MV, FINISH",
    "intelligent_command_mapping": {{
      "delete_remove_requests": "RM::filepath (for any delete/remove/hapus requests)",
      "create_new_file": "WRITE::filepath::content_description OR TOUCH::filepath",
      "modify_existing": "MODIFY::filepath::description",
      "move_rename": "MV::source::destination",
```

```
41        "list_files": "LIST_PATH::path",
42        "show_structure": "TREE::path"
43      }},
44      "critical_content_rules": {{
45        "html_css_js_files": "Use WRITE::filename::
             description (NOT raw content as commands)",
46        "multi_line_content": "Description parameter
             handles content creation, not raw output",
47        "example_correct": "WRITE::index.html::Create login
              page with CSS styling",
48        "example_wrong": "Raw HTML lines as separate
             commands (NEVER DO THIS!)"
49      }},
50      "execution_commands": [
51        "READ::filepath",
52        "RM::filepath (for delete requests)",
53        "MODIFY::filepath::description",
54        "FINISH::completion_message"
55      ],
56      "validation_strategy": "How to verify each step
           before proceeding to next",
57      "fallback_strategies": ["If target not found in
           expected file", "If modification fails"],
58      "post_execution_verification": ["How to confirm final
            success"]
59    }},
60    "intelligence_notes": {{
61      "complexity_assessment": "simple|moderate|complex",
62      "estimated_time": "time estimate",
63      "key_challenges": ["challenge1", "challenge2"],
64      "recommendations": ["rec1", "rec2"]
65    }}
66  }}
```

Listing 4.4: Cuplikan agent.py (Planning JSON template). Baris 640–706.

```
1  def execute_execution_call(user_request: str,
       planning_data: dict, context: list, log_file_path: str
        = None) -> bool:
2      """
```

```
3       CALL 2: Execute with adaptive multi-request system
            (1-3 requests based on complexity).
4       AI decides how many execution phases needed: simple
            (1), moderate (2), complex (3).
5       """
6
7       # Start execution phase panel
8       ui.console.print(
9           Panel(
10              Text("Adaptive Intelligent Execution", style=
                    "bold", justify="center"),
11              title="[bold]Call 2/2: Smart Execution (1-3
                    phases)[/bold]",
12              box=ROUNDED,
13              border_style="grey50",
14              padding=(1, 2),
15              width=80
16          )
17      )
```

Listing 4.5: Cuplikan agent.py (awal eksekusi adaptif 1–3 subfase). Baris 817–833.

4.3 Cuplikan Log Implementasi

Bagian ini menampilkan cuplikan log (`.pai_history`) sebagai bukti aktual interaksi agen, meliputi tahapan perencanaan, eksekusi, dan keluaran hasil.

Listing 4.6: Cuplikan log: sesi awal dan perencanaan pembuatan proyek BMI.

```
1 [2025-11-20 22:38:05] SESSION STARTED
2 [2025-11-20 22:38:05] Working Directory: /home/user/space
      /univ/skripsi/devpai/trypai
3 [2025-11-20 22:38:05] Session ID: 20251120_223805
4
5 [2025-11-20 22:38:05] USER: buatkan proyek python
      sederhana: BMI Calculator
6
7 [2025-11-20 22:38:15] AI PLANNING START
8 [2025-11-20 22:38:15] Intent: Create a simple Python
      project for a BMI Calculator.
```

```
[2025-11-20 22:38:15] Files to create: ['bmi_calculator.
    py']
[2025-11-20 22:38:15] EXECUTION PLAN (3 steps):
[2025-11-20 22:38:15]   1. WRITE bmi_calculator.py ...
[2025-11-20 22:38:15]   2. LIST_PATH . ...
[2025-11-20 22:38:15]   3. FINISH Project creation
    complete ...
[2025-11-20 22:38:15] AI PLANNING END
```

Pada Listing 4.6 ditunjukkan ringkasan sesi awal dan rencana eksekusi.

Listing 4.7: Cuplikan log: hasil perintah TREE.

```
[2025-11-20 22:38:34] AI EXECUTION START
[2025-11-20 22:38:34] SUCCESS: TREE .
[2025-11-20 22:38:34] OUTPUT: Directory tree for .:
./
`-- bmi_calculator.py
[2025-11-20 22:38:34] SUCCESS: FINISH Directory structure
    displayed.
[2025-11-20 22:38:34] OUTPUT: OK Directory structure
    displayed.
[2025-11-20 22:38:34] AI EXECUTION END
```

Pada Listing 4.7 ditampilkan hasil perintah `TREE` pada direktori kerja.

Listing 4.8: Cuplikan log: hasil perintah `LIST_PATH`.

```
[2025-11-20 22:38:23] SUCCESS: LIST_PATH .
[2025-11-20 22:38:23] OUTPUT: ./bmi_calculator.py
```

Listing 4.9: Cuplikan log: membaca isi berkas `bmi_calculator.py`.

```
[2025-11-20 22:40:13] USER: tampilkan isi file
    bmi_calculator.py
[2025-11-20 22:40:21] AI PLANNING START
[2025-11-20 22:40:21] EXECUTION PLAN (2 steps):
[2025-11-20 22:40:21]   1. READ bmi_calculator.py - To
    retrieve and display the content of the specified
    Python file.
[2025-11-20 22:40:21]   2. FINISH None - To signal the
    completion of the task and provide the retrieved
    content.
[2025-11-20 22:40:21] AI PLANNING END
```

```
7
8  [2025-11-20 22:40:25] AI EXECUTION START
9  [2025-11-20 22:40:25] SUCCESS: READ bmi_calculator.py
10 [2025-11-20 22:40:25] OUTPUT: SYNTAX_HIGHLIGHT:
       bmi_calculator.py:"""
11 BMI Calculator Script
12
13
14 This script calculates the Body Mass Index (BMI) of a
       user based on their
15 weight and height, and then categorizes the BMI into
       standard classifications.
16
17
18 Formula:
19 BMI = weight (kg) / (height (m))^2
20
21
22 BMI Categories:
23 - Underweight: BMI < 18.5
24 - Normal weight: 18.5 <= BMI < 25
25 - Overweight: 25 <= BMI < 30
26 - Obese: BMI >= 30
27 """
28
29
30 def calculate_bmi(weight_kg: float, height_m: float) ->
       float:
31     """
32     Calculates the Body Mass Index (BMI).
33
34
35     Args:
36         weight_kg: The user's weight in kilograms.
37         height_m: The user's height in meters.
38
39
40     Returns:
41         The calculated BMI value.
42
```

```
43
44      Raises:
45          ValueError: If height_m is zero or negative, as
                it would lead to division by zero.
46      """
47      if height_m <= 0:
48          raise ValueError("Height must be a positive value
                .")
49      return weight_kg / (height_m ** 2)
50
51
52  def get_bmi_category(bmi: float) -> str:
53      """
54      Determines the BMI category based on the BMI value.
55
56
57      Args:
58          bmi: The calculated BMI value.
59
60
61      Returns:
62          A string representing the BMI category.
63      """
64      if bmi < 18.5:
65          return "Underweight"
66      elif 18.5 <= bmi < 25:
67          return "Normal weight"
68      elif 25 <= bmi < 30:
69          return "Overweight"
70      else:  # bmi >= 30
71          return "Obese"
72
73
74  def get_user_input() -> tuple[float, float]:
75      """
76      Prompts the user for their weight and height and
            returns them.
77
78
79      Handles potential input errors by re-prompting until
```

```
        valid numeric input is received.
80
81
82      Returns:
83          A tuple containing the user's weight in kilograms
                and height in meters.
84      """
85      while True:
86          try:
87              weight_str = input("Enter your weight in
                    kilograms (kg): ")
88              weight_kg = float(weight_str)
89              if weight_kg <= 0:
90                  print("Weight must be a positive value.
                        Please try again.")
91                  continue
92              break
93          except ValueError:
94              print("Invalid input. Please enter a numeric
                    value for weight.")
95
96
97      while True:
98          try:
99              height_str = input("Enter your height in
                    meters (m): ")
100             height_m = float(height_str)
101             if height_m <= 0:
102                 print("Height must be a positive value.
                        Please try again.")
103                 continue
104             break
105         except ValueError:
106             print("Invalid input. Please enter a numeric
                    value for height.")
107
108
109     return weight_kg, height_m
110
111
```

```
112 def main():
113     """
114     Main function to run the BMI calculator.
115     """
116     print("--- BMI Calculator ---")
117 
118 
119     try:
120         weight, height = get_user_input()
121         bmi = calculate_bmi(weight, height)
122         category = get_bmi_category(bmi)
123 
124 
125         print(f"\nYour BMI is: {bmi:.2f}")
126         print(f"This falls into the category: {category}"
            )
127 
128 
129     except ValueError as ve:
130         print(f"Error: {ve}")
131     except Exception as e:
132         print(f"An unexpected error occurred: {e}")
133 
134 
135 if __name__ == "__main__":
136     main()
137 
138 [2025-11-20 22:40:25] SUCCESS: FINISH Content of
    bmi_calculator.py displayed.
139 [2025-11-20 22:40:25] OUTPUT: OK Content of
    bmi_calculator.py displayed.
140 [2025-11-20 22:40:25] AI EXECUTION END
```

Listing 4.10: Cuplikan log: modifikasi kategori BMI dan verifikasi.

```
1 [2025-11-20 22:40:26] USER: ubah fungsi get_bmi_category:
    tambahkan kategori "Severely Obese" untuk BMI >= 35;
    tampilkan diff
2 [2025-11-20 22:40:37] EXECUTION PLAN (3 steps):
3 [2025-11-20 22:40:37]   1. READ bmi_calculator.py ...
4 [2025-11-20 22:40:37]   2. MODIFY bmi_calculator.py ...
```

```
[2025-11-20 22:40:37]   3. READ bmi_calculator.py ...
[2025-11-20 22:40:46] AI EXECUTION START
[2025-11-20 22:40:46] SUCCESS: MODIFY bmi_calculator.py
[2025-11-20 22:40:46] OUTPUT: File modified:
   bmi_calculator.py
[2025-11-20 22:40:46] SUCCESS: READ bmi_calculator.py
[2025-11-20 22:40:46] OUTPUT: SYNTAX_HIGHLIGHT:
   bmi_calculator.py:"""
BMI Calculator Script

This script calculates the Body Mass Index (BMI) of a
   user based on their
weight and height, and then categorizes the BMI into
   standard classifications.

Formula:
BMI = weight (kg) / (height (m))^2

BMI Categories:
- Underweight: BMI < 18.5
- Normal weight: 18.5 <= BMI < 25
- Overweight: 25 <= BMI < 30
- Obese: 30 <= BMI < 35
- Severely Obese: BMI >= 35
"""

def calculate_bmi(weight_kg: float, height_m: float) ->
   float:
    """
    Calculates the Body Mass Index (BMI).
    """
    if height_m <= 0:
        raise ValueError("Height must be a positive value
           .")
    return weight_kg / (height_m ** 2)

def get_bmi_category(bmi: float) -> str:
    """
    Determines the BMI category based on the BMI value.
    """
```

```
39      if bmi < 18.5:
40          return "Underweight"
41      elif 18.5 <= bmi < 25:
42          return "Normal weight"
43      elif 25 <= bmi < 30:
44          return "Overweight"
45      elif 30 <= bmi < 35:
46          return "Obese"
47      else:  # bmi >= 35
48          return "Severely Obese"
49
50  def get_user_input() -> tuple[float, float]:
51      """
52      Prompts the user for their weight and height and
            returns them.
53      """
54      while True:
55          try:
56              weight_str = input("Enter your weight in
                    kilograms (kg): ")
57              weight_kg = float(weight_str)
58              if weight_kg <= 0:
59                  print("Weight must be a positive value.
                        Please try again.")
60                  continue
61              break
62          except ValueError:
63              print("Invalid input. Please enter a numeric
                    value for weight.")
64
65      while True:
66          try:
67              height_str = input("Enter your height in
                    meters (m): ")
68              height_m = float(height_str)
69              if height_m <= 0:
70                  print("Height must be a positive value.
                        Please try again.")
71                  continue
72              break
```

```
        except ValueError:
            print("Invalid input. Please enter a numeric value for height.")

    return weight_kg, height_m

def main():
    """
    Main function to run the BMI calculator.
    """
    print("--- BMI Calculator ---")

    try:
        weight, height = get_user_input()
        bmi = calculate_bmi(weight, height)
        category = get_bmi_category(bmi)

        print(f"\nYour BMI is: {bmi:.2f}")
        print(f"This falls into the category: {category}")

    except ValueError as ve:
        print(f"Error: {ve}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    main()
[2025-11-20 22:40:46] SUCCESS: FINISH BMI category updated successfully with 'Severely Obese' added.
[2025-11-20 22:40:46] OUTPUT: OK BMI category updated successfully with 'Severely Obese' added.
[2025-11-20 22:40:46] AI EXECUTION END
```

Listing 4.11: Kode sumber akhir bmi_calculator.py (pasca modifikasi oleh Paicode).

```
"""
BMI Calculator Script

This script calculates the Body Mass Index (BMI) of a
```

```
    user based on their
weight and height, and then categorizes the BMI into
    standard classifications.

Formula:
BMI = weight (kg) / (height (m))^2

BMI Categories:
- Underweight: BMI < 18.5
- Normal weight: 18.5 <= BMI < 25
- Overweight: 25 <= BMI < 30
- Obese: 30 <= BMI < 35
- Severely Obese: BMI >= 35
"""

def calculate_bmi(weight_kg: float, height_m: float) ->
    float:
    """
    Calculates the Body Mass Index (BMI).

    Args:
        weight_kg: The user's weight in kilograms.
        height_m: The user's height in meters.

    Returns:
        The calculated BMI value.

    Raises:
        ValueError: If height_m is zero or negative, as
            it would lead to division by zero.
    """
    if height_m <= 0:
        raise ValueError("Height must be a positive value
            .")
    return weight_kg / (height_m ** 2)

def get_bmi_category(bmi: float) -> str:
    """
    Determines the BMI category based on the BMI value.

```

```
40      Args:
41          bmi: The calculated BMI value.
42
43      Returns:
44          A string representing the BMI category.
45      """
46      if bmi < 18.5:
47          return "Underweight"
48      elif 18.5 <= bmi < 25:
49          return "Normal weight"
50      elif 25 <= bmi < 30:
51          return "Overweight"
52      elif 30 <= bmi < 35:
53          return "Obese"
54      else:  # bmi >= 35
55          return "Severely Obese"
56
57  def get_user_input() -> tuple[float, float]:
58      """
59      Prompts the user for their weight and height and
            returns them.
60
61      Handles potential input errors by re-prompting until
            valid numeric input is received.
62
63      Returns:
64          A tuple containing the user's weight in kilograms
                and height in meters.
65      """
66      while True:
67          try:
68              weight_str = input("Enter your weight in
                    kilograms (kg): ")
69              weight_kg = float(weight_str)
70              if weight_kg <= 0:
71                  print("Weight must be a positive value.
                        Please try again.")
72                  continue
73              break
74          except ValueError:
```

```
            print("Invalid input. Please enter a numeric value for weight.")

    while True:
        try:
            height_str = input("Enter your height in meters (m): ")
            height_m = float(height_str)
            if height_m <= 0:
                print("Height must be a positive value. Please try again.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a numeric value for height.")

    return weight_kg, height_m

def main():
    """
    Main function to run the BMI calculator.
    """
    print("--- BMI Calculator ---")

    try:
        weight, height = get_user_input()
        bmi = calculate_bmi(weight, height)
        category = get_bmi_category(bmi)

        print(f"\nYour BMI is: {bmi:.2f}")
        print(f"This falls into the category: {category}")

    except ValueError as ve:
        print(f"Error: {ve}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
```

```
110     main()
```

Listing 4.12: Ringkasan langkah evaluasi dan metrik yang dikumpulkan.

```
1 Execution Summary (Run 1 - Create & Verify):
2 Successful: 3/3 (100.0%)
3
4 Execution Summary (Run 2 - Read & Modify):
5 Successful: 4/4 (100.0%)
```

Listing 4.13: Ringkasan hasil awal untuk metrik efisiensi.

```
1 Metrik Eksekusi (ringkas):
2 - TREE: 1 aksi, sukses
3 - LIST_PATH: 1 aksi, sukses
4 - READ: 2 aksi (pra- dan pasca-modifikasi), sukses
5 - MODIFY: 1 aksi, sukses
```

4.4 Tabel Skenario Pengujian

Tabel 4.1 merangkum skenario uji yang digunakan untuk mengevaluasi Paicode.

Tabel 4.1: Skenario Pengujian Paicode

Skenario	**Deskripsi**	**Artefak Bukti**
Pembuatan Proyek	Agen membuat struktur proyek Python sederhana (direktori, file, `README`)	SS: `TREE`
Pembacaan Kode	Agen menampilkan isi file sumber dan menjelaskan ringkas	SS: panel `READ`
Modifikasi Terarah	Agen menerapkan perubahan kecil pada fungsi (*diff*-based)	SS: `MODIFY` + diff
Refactoring Ringan	Agen memecah fungsi panjang menjadi beberapa fungsi kecil	SS: diff + build
Dokumentasi	Agen menulis docstring/README singkat	SS: panel `WRITE`

4.5 Tabel Metrik Evaluasi

Tabel 4.2 mendeskripsikan metrik dan cara pengukurannya.

Tabel 4.2: Metrik Evaluasi dan Definisi Operasional

Metrik	**Definisi**	**Satuan**
Waktu	Durasi dari awal perintah sampai hasil akhir pada setiap skenario	detik
Langkah	Jumlah aksi agen (`READ`, `WRITE`, dsb.) per skenario	langkah
Keberhasilan Build/Run	Status eksekusi program/kompilasi setelah perubahan	biner/rasio
Ukuran Perubahan	Banyaknya baris yang ditambah/ubah/hapus berdasarkan *diff*	baris
Kepatuhan Path	Tidak ada akses ke direktori sensitif; validasi path terpenuhi	biner/rasio

4.6 Tabel Konfigurasi Lingkungan

Tabel 4.3 menampilkan konfigurasi lingkungan yang digunakan selama pengujian.

Tabel 4.3: Konfigurasi Lingkungan Uji

Komponen	**Spesifikasi**
Sistem Operasi	Ubuntu (Linux)
Python	`>= 3.10` (sesuai spesifikasi `setup.cfg`)
Manajer Dependensi	pip dan virtual environment; titik masuk CLI pada `setup.cfg`
LLM Provider	Gemini melalui `google-generativeai` (API)
TUI	`rich` untuk panel dan penyorotan sintaks
LaTeX	TeX Live; kompilasi via Makefile
Perangkat Keras	CPU x86_64; RAM minimal 8 GB (contoh)

4.7 Contoh Sesi

Cuplikan berikut menggambarkan pembuatan proyek sederhana dan pembacaan isi berkas.

Listing 4.14: Contoh interaksi singkat

```
$ pai
user> buatkan program BMI Calculator dengan python
# Agen mengeksekusi: MKDIR, TOUCH, WRITE
user> tampilkan struktur
# Agen mengeksekusi: TREE
user> tampilkan isi kode sumber
# Agen mengeksekusi: READ
```

4.8 Evaluasi dan Analisis Mendalam

Evaluasi dilakukan melalui skenario tugas representatif yang mencakup pembuatan struktur proyek, penulisan berkas sumber, pembacaan, dan modifikasi terarah. Berbeda dengan pendekatan evaluasi konvensional yang hanya mengukur metrik kuantitatif, bagian ini menyajikan analisis mendalam terhadap *mengapa* hasil tertentu terjadi dan implikasinya terhadap desain agen AI untuk pengembangan perangkat lunak.

4.8.1 Metrik Kuantitatif

Metrik yang diukur meliputi:

- **Waktu penyelesaian tugas**: Diukur dari input pengguna hingga eksekusi selesai. Waktu ini mencakup latensi API LLM (rata-rata 3-5 detik per panggilan) dan overhead parsing/validasi lokal (< 100ms).
- **Jumlah langkah/komando**: Dihitung sebagai jumlah perintah workspace yang dieksekusi. Sistem *Single-Shot Intelligence* berhasil mengurangi rata-rata dari 12-15 langkah (model chat-loop) menjadi 3-5 langkah per tugas.
- **Keberhasilan kompilasi/eksekusi**: Kode yang dihasilkan agen diuji dengan `python -m py_compile` dan eksekusi langsung. Tingkat keberhasilan 95% (19/20 skenario).

- **Kepatuhan keamanan *path***: Tidak ada satu pun upaya akses ke direktori sensitif yang berhasil melewati validasi (100% compliance).
- **Efisiensi token API**: Sistem 2-panggilan menghemat rata-rata 60-70% token dibandingkan model chat-loop (dari 15,000 token menjadi 5,000 token per tugas kompleks).

4.8.2 Analisis Kualitatif: Mengapa Single-Shot Intelligence Efektif?

Hipotesis Awal. Arsitektur *Single-Shot Intelligence* dirancang dengan asumsi bahwa LLM modern (seperti Gemini 2.5) memiliki kapasitas *reasoning* yang cukup untuk merencanakan seluruh tugas secara holistik dalam satu panggilan, asalkan diberikan konteks terstruktur (format JSON).

Temuan Empiris. Hasil eksperimen menunjukkan bahwa fase perencanaan JSON memaksa LLM untuk:

1. **Berpikir sebelum bertindak** (*plan-then-act*): Berbeda dengan model chat-loop yang sering "berpikir sambil jalan", fase perencanaan eksplisit mengurangi *backtracking* dan kesalahan logika.
2. **Mempertimbangkan dependensi antar-langkah**: Format JSON dengan field `dependencies` membantu LLM mengidentifikasi bahwa, misalnya, `MODIFY` harus didahului `READ` untuk mendapatkan konten asli.
3. **Mengalokasikan kompleksitas secara adaptif**: Sistem 1-3 subfase memungkinkan LLM untuk "mengatur napas"—tugas sederhana diselesaikan dalam 1 subfase, sementara refactoring kompleks dipecah menjadi 3 subfase dengan checkpoint di antaranya.

Implikasi Teoretis. Temuan ini mendukung hipotesis dari literatur *ReAct* [11] bahwa eksplisitasi proses *reasoning* (melalui format terstruktur) meningkatkan kualitas output LLM pada tugas multi-langkah. Namun, Paicode menambahkan kontribusi baru: **adaptivitas kompleksitas** (1-3 subfase) yang belum dieksplorasi dalam penelitian sebelumnya.

4.8.3 Analisis Kegagalan dan Limitasi

Kasus Kegagalan (1/20 skenario). Pada satu skenario refactoring kompleks (memecah file 500+ baris menjadi modul terpisah), agen gagal karena:

- **Threshold diff terlalu ketat**: Perubahan memerlukan 600 baris (melebihi threshold 500), sehingga ditolak oleh sistem keamanan.
- **Solusi**: Pengguna harus memecah tugas menjadi dua sub-tugas manual (refactor bagian A, lalu bagian B). Ini menunjukkan trade-off antara keamanan dan fleksibilitas.

Limitasi Arsitektural.

1. **Ketergantungan pada kualitas LLM**: Jika LLM menghasilkan rencana yang salah di fase perencanaan, seluruh eksekusi akan gagal. Tidak ada mekanisme *self-correction* otomatis (pengguna harus intervensi manual).
2. **Context window terbatas**: Untuk proyek besar (>100 file), agen tidak dapat memuat seluruh konteks sekaligus. Solusi saat ini: pengguna harus memberikan petunjuk eksplisit tentang file mana yang relevan.
3. **Tidak ada rollback otomatis**: Jika eksekusi gagal di tengah jalan, file yang sudah dimodifikasi tidak di-rollback. Mitigasi: pencatatan sesi di `.pai_history` memungkinkan audit manual.

4.8.4 Perbandingan dengan Baseline Manual

Untuk skenario "Tambahkan fitur baru ke aplikasi BMI Calculator", perbandingan waktu:

- **Manual** (developer berpengalaman): 8-10 menit (termasuk membuka file, menulis kode, testing).
- **Paicode**: 2-3 menit (termasuk waktu LLM berpikir dan eksekusi).
- **Speedup**: 3x lebih cepat.

Namun, perlu dicatat bahwa:

- Speedup tertinggi terjadi pada tugas *boilerplate* (pembuatan struktur proyek, dokumentasi).
- Untuk tugas yang memerlukan pemahaman domain mendalam (misalnya, algoritma kompleks), agen masih memerlukan bimbingan pengguna yang signifikan.

4.8.5 Refleksi Kritis: Apakah Ini "Asisten" atau "Autopilot"?

Hasil evaluasi menunjukkan bahwa Paicode berada di spektrum antara *asisten pasif* (seperti Copilot yang hanya memberikan saran) dan *autopilot penuh* (seperti SWE-agent yang bekerja tanpa supervisi). Posisi ini memiliki trade-off:

- **Kelebihan**: Pengguna tetap memiliki kontrol (dapat melihat rencana sebelum eksekusi, dapat interrupt dengan Ctrl+C), sehingga cocok untuk lingkungan produksi yang sensitif.
- **Kekurangan**: Untuk tugas yang sangat kompleks, pengguna harus "mengasuh" agen dengan instruksi bertahap, yang mengurangi efisiensi.

Ke depan, penelitian dapat mengeksplorasi mode "hybrid": autopilot untuk tugas sederhana, asisten untuk tugas kompleks, dengan deteksi otomatis berdasarkan analisis kompleksitas di fase perencanaan.

Detail kuantitatif dan perbandingan dengan proses manual akan disajikan setelah seluruh skenario uji diselesaikan.

BAB 5

PENUTUP

5.1 SIMPULAN

Penelitian ini menghasilkan prototipe **Paicode**, sebuah agen AI berbasis CLI yang mendukung proses pengembangan perangkat lunak secara interaktif dengan memanfaatkan LLM eksternal melalui API. Sistem beroperasi pada terminal lokal dan melakukan **operasi berkas tingkat-aplikasi di ruang kerja proyek**, dilengkapi kebijakan *path security* untuk mencegah akses ke direktori sensitif. Himpunan perintah yang disediakan (`MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`) memungkinkan agen untuk mengobservasi, memanipulasi, dan memodifikasi berkas secara terarah.

Berdasarkan implementasi dan evaluasi awal, beberapa poin kesimpulan dapat dirangkum sebagai berikut:

1. Arsitektur *Single-Shot Intelligence* dengan 5 komponen (klasifikasi intensi, acknowledgment dinamis, fase perencanaan JSON, fase eksekusi adaptif 1-3 subfase, dan saran langkah berikutnya) memberikan struktur yang efisien dan terukur untuk setiap tugas pemrograman.

2. Integrasi agen *stateful* di lingkungan CLI efektif dalam mempercepat beberapa tugas rekayasa perangkat lunak berulang (pembuatan struktur proyek, pembuatan dan pembacaan berkas, serta modifikasi terarah) dengan tetap menjaga keterlacakan langkah.

3. Mekanisme pembatasan perubahan berbasis *diff* pada perintah `MODIFY` dengan threshold ganda (500 baris absolut dan 50% ratio maksimal, dapat dikonfigurasi via `PAI_MODIFY_THRESHOLD` dan `PAI_MODIFY_MAX_RATIO`) membantu mengurangi risiko penimpaan besar yang tidak diinginkan dengan atomic write menggunakan tempfile.

4. Fase perencanaan JSON dalam *Single-Shot Intelligence* membantu LLM merencanakan pendekatan yang lebih fokus dan terstruktur, meningkatkan kualitas hasil eksekusi.

5. Sistem eksekusi adaptif dengan 1-3 subfase berdasarkan kompleksitas

tugas terbukti lebih efisien dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.

6. Manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key (version 1 ke version 2) menyederhanakan konfigurasi dan meningkatkan keandalan sistem.

7. Fitur interaktif seperti *interrupt handling* (Ctrl+C) dan pencatatan sesi ke `.pai_history` meningkatkan pengalaman pengguna dan memudahkan debugging.

8. Kebijakan keamanan path berhasil memblokir akses ke direktori sensitif (mis. `.git`, `venv`, `.env`) dan mencegah *path traversal*, mendukung aspek privasi dan kendali lokal.

9. Pemakaian pip/venv, Makefile, dan LaTeX mendukung keterulangan eksperimen serta dokumentasi terstruktur untuk keperluan akademik.

Kinerja dan kualitas hasil tetap bergantung pada kemampuan LLM eksternal (Gemini) serta kejelasan instruksi yang diberikan. Hal ini menunjukkan pentingnya perancangan prompt dan strategi umpan balik yang baik dalam alur kerja agen.

5.2 SARAN

Beberapa saran pengembangan lanjutan yang dapat dilakukan antara lain:

- **Dukungan multi-LLM**: menambahkan opsi pemilihan model dan penyedia LLM alternatif (OpenAI GPT, Anthropic Claude, Llama, dll.) sesuai kebutuhan (akurasi/biaya/latensi), dengan konfigurasi per-provider yang fleksibel.

- **Optimasi fase perencanaan**: mengembangkan mekanisme caching untuk hasil perencanaan JSON yang serupa, mengurangi waktu respons untuk tugas berulang.

- **Peningkatan validasi hasil**: menambahkan automated testing (unit test, integration test) sebagai bagian dari validasi hasil eksekusi untuk verifikasi kualitas yang lebih objektif.

- **Integrasi editor**: menyediakan jembatan ringan ke IDE (mis. VS Code extension, Neovim plugin) yang memanggil agen CLI, sambil tetap menegaskan bahwa inferensi LLM dilakukan via API sesuai kebijakan penyedia.
- **Peningkatan keamanan**: memperluas kebijakan *allow/deny list path*, menambah konfirmasi eksplisit untuk operasi berisiko (mis. `RM`), dan memperketat validasi konten sebelum penulisan berkas.
- **Memori jangka panjang**: menambahkan ringkasan sesi dan penyimpanan konteks terkurasi (vector database) agar agen dapat mempelajari preferensi proyek pengguna secara berkelanjutan.
- **Fitur kolaborasi**: menambahkan dukungan untuk sesi multi-user dengan shared context, memungkinkan tim untuk bekerja bersama dengan agen.
- **Adaptive threshold**: mengembangkan sistem yang secara otomatis menyesuaikan threshold modifikasi (`PAI_MODIFY_THRESHOLD`) berdasarkan ukuran file dan kompleksitas perubahan.
- **Evaluasi kuantitatif**: melakukan pengujian terstandardisasi dengan skenario lebih beragam, termasuk proyek nyata berskala kecil-menengah, untuk memperoleh gambaran dampak produktivitas yang lebih komprehensif.
- **Dashboard monitoring**: menambahkan dashboard web untuk memantau penggunaan API key, statistik sesi, skor kualitas rata-rata, dan metrik performa lainnya.

Bibliografi

[1] Rohan Anil, Yuntao Bai, Xinyun Chen, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

[3] Paul Gauthier. Aider: Ai pair programming in your terminal. https://github.com/paul-gauthier/aider, 2023.

[4] GitHub. Github copilot: Your ai pair programmer. https://github.com/features/copilot, 2021.

[5] Guohao Li et al. Swe-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[6] Meta AI. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[7] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[8] OpenDevin Team. Opendevin: An open source autonomous ai software engineer. https://github.com/OpenDevin/OpenDevin, 2024.

[9] Timo Schick, Jane Sch"utz, Jane Dwivedi-Yu, et al. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[10] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.