

**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 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



Disusun Oleh

**I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001**

**PROGRAM STUDI INFORMATIKA
PROGRAM SARJANA
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 PENGEOMBANGAN PERANGKAT LUNAK DI LINUX YANG DITENAGAI LLM MELALUI API
Nama : I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001
Program Studi : Informatika
Program : Sarjana
Semester : Gasal
Tahun Akademik : 2025/2026



Dr. Bambang Purnomasidi 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 Teknologi Informasi
 Universitas Teknologi Digital Indonesia

Yogyakarta, 15 Desember 2025

Dewan Penguji

1. Wagito, S.T., M.T. (Ketua)

2. Dr. Bambang Purnomasidi Dwi Putranto,

S.E., Akt., S.Kom., MMSI (Sekretaris)

3. Ariesta Damayanti, S.Kom., M.Cs. (Anggota)

NIDN

Tandatangan

.....

.....

.....

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, 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 Purnomasidi 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 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

HALAMAN JUDUL	i
HALAMAN PENGESAHAN	iii
PERNYATAAN KEASLIAN TUGAS AKHIR	iv
HALAMAN PERSEMBAHAN	v
PRAKATA	vi
DAFTAR ISI	viii
DAFTAR GAMBAR	xi
DAFTAR TABEL	xii
INTISARI	xiii
ABSTRACT	xiv
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Ruang Lingkup	2
1.4 Tujuan Penelitian	2
1.5 Manfaat Penelitian	3
1.6 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA DAN DASAR TEORI	5
2.1 Tinjauan Pustaka	5
2.1.1 AI Coding Assistant Terintegrasi (IDE-based)	5
2.1.2 CLI-based AI Chat Tools	5
2.1.3 Autonomous Software Engineers	5
2.1.4 Posisi Paicode	5
2.1.5 Perbandingan dengan Penelitian Sebelumnya	6
2.1.6 Posisi Penelitian	6
2.2 Dasar Teori	7

2.2.1	Command Line Interface (CLI)	7
2.2.2	AI Agent	7
2.2.3	Large Language Model (LLM)	8
2.2.4	Perbedaan LLM dan Agen AI	8
2.2.5	Manajemen Dependensi dengan pip dan Virtual Environment	8
3	METODE PENELITIAN	9
3.1	Bahan Penelitian	9
3.2	Peralatan	9
3.3	Metodologi Penelitian	10
3.3.1	Tahapan Penelitian	10
3.4	Analisis Kebutuhan Sistem	11
3.4.1	Kebutuhan Fungsional	11
3.4.2	Kebutuhan Non-Fungsional	11
3.5	Perancangan Sistem	11
3.5.1	Arsitektur Modular	11
3.5.2	Deskripsi Modul	12
3.5.3	Alur Kerja Sistem	13
4	IMPLEMENTASI DAN PEMBAHASAN	15
4.1	Implementasi dan Uji Coba Sistem	15
4.1.1	Lingkungan Implementasi	15
4.1.2	Implementasi Fitur Utama	15
4.1.3	Skenario Pengujian	21
4.1.4	Hasil Uji Coba	22
4.2	Pembahasan	26
4.2.1	Efisiensi Mekanisme Perencanaan Otomatis	26
4.2.2	Analisis Aspek Keamanan	26
4.2.3	Perbandingan dengan Metode Manual	26
4.2.4	Keterbatasan Sistem	28
5	PENUTUP	29
5.1	SIMPULAN	29
5.2	SARAN	30
	DAFTAR PUSTAKA	32
	LAMPIRAN	33

Lampiran A: Manual Penggunaan Aplikasi	33
Lampiran B: Instrumen Pengujian	35

Daftar Gambar

3.1 Flowchart alur eksekusi perintah dalam arsitektur <i>Single-Shot Intelligence</i>	13
4.1 Perbandingan alur kerja Manual vs Paicode. Paicode mengeliminasi <i>context switching</i> dan beban pengetikan.	28

Daftar Tabel

2.1 Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan	6
3.1 Daftar Modul dan Tanggung Jawab Utama	12
4.1 Konfigurasi Lingkungan Implementasi	15
4.2 Skenario Pengujian Fungsional	22
4.3 Hasil Pengukuran Waktu Eksekusi (3 Percobaan)	25
4.4 Ringkasan Indikator Keberhasilan Selepas 3 Percobaan	25
4.5 Perbandingan Jumlah Langkah Kerja (Skenario 1)	27
4.6 Perbandingan Rata-rata Waktu Penyelesaian Tugas	27

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* meningkatkan 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, *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. Sistem eksekusi adaptif dengan 1-3 subfase menunjukkan efisiensi waktu operasional dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang, dengan tetap mempertahankan kualitas hasil yang baik.

Kata kunci: AI, agen, CLI, LLM, API, keamanan, 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 improves efficiency through an API call system consisting of: (1) intent classification, (2) dynamic acknowledgement, (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, *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 high result quality.

Keywords: AI, agent, CLI, LLM, API, security, software, development.

BAB I

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 (Brown et al., 2020; OpenAI, 2023; Anil et al., 2023). 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) meningkatkan 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, 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:

1. Lingkungan target adalah sistem operasi Ubuntu (Linux) dengan antarmuka CLI.
2. Bahasa pemrograman utama adalah Python; contoh dan skenario uji berfokus pada ekosistem Python/Unix.
3. Layanan LLM eksternal menggunakan API Gemini; kualitas respons bergantung pada model dan tidak menjadi ruang lingkup untuk dioptimasi ulang.
4. Dukungan multi-pengguna, kolaborasi real-time, dan integrasi langsung dengan editor tidak dibahas pada versi ini.
5. 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 menguji fungsionalitas 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 fase utama), 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:

1. **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.
2. **Praktis:** menghadirkan alat bantu pengembangan perangkat lunak dengan kelebihan spesifik sebagai berikut:
 - (a) **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.
 - (b) **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.
 - (c) **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 dalam lima bab yang saling berkaitan. Bab I (Pendahuluan) memaparkan latar belakang masalah, rumusan masalah yang akan diselesaikan, ruang lingkup penelitian, tujuan yang ingin dicapai, manfaat penelitian bagi aspek akademis maupun praktis, serta sistematika penulisan laporan ini.

Bab II (Tinjauan Pustaka dan Dasar Teori) menguraikan tinjauan pustaka dari penelitian-penelitian terdahulu yang relevan dengan pengembangan agen

AI dan CLI, serta landasan teori yang mendukung penelitian, meliputi konsep *Command Line Interface* (CLI), *Artificial Intelligence* (AI) Agent, *Large Language Model* (LLM), dan arsitektur perangkat lunak terkait.

Bab III (Metode Penelitian) menjelaskan objek dan bahan penelitian yang digunakan, peralatan pendukung baik perangkat keras maupun lunak, prosedur penelitian yang dilakukan selama penelitian, serta analisis dan perancangan sistem Paicode secara rinci.

Bab IV (Implementasi dan Pembahasan) menjabarkan proses lingkungan implementasi sistem, realisasi fitur-fitur utama Paicode, skenario pengujian yang dilakukan, serta pembahasan mendalam mengenai hasil uji coba dan evaluasi kinerja sistem.

Bab V (Penutup) berisi simpulan yang diperoleh dari seluruh rangkaian penelitian serta saran-saran konstruktif untuk pengembangan sistem Paicode di masa mendatang.

BAB II

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 (2021) menghadirkan GitHub Copilot sebagai 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.

2.1.2 CLI-based AI Chat Tools

Gauthier (2023) mengembangkan Aider untuk 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.

2.1.3 Autonomous Software Engineers

OpenDevin Team (2024) dan Li et al. (2024) masing-masing meluncurkan proyek OpenDevin dan SWE-agent yang bertujuan menciptakan agen yang sepenuhnya otonom, mampu menyelesaikan isu GitHub dari awal hingga akhir tanpa interaksi manusia. Meskipun sangat canggih, pendekatan ini sering kali memerlukan akses sumber daya yang besar (Docker container penuh). Paicode mengambil posisi tengah (*middle-ground*) dengan menyediakan agen *semi-autonomous* yang ringan.

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.

Tabel 2.1: Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan

Pene-litian	Plat-form	Arsitektur	Kea-manan	Trans-paransi	Efisiensi	Inter-aktivitas
Copilot	IDE-based	Chat-loop iteratif	Tidak eksplisit	Black-box	High token	Passive
ChatGPT	Web-based	Chat-loop	Tidak eksplisit	Black-box	High token	Passive
OpenDevinContainer	Fully autonomous	Sand-boxed	Verbose logs	Resource-intensive	Auto-nomous	
SWE-agent	General	Autonomous	Sand-boxed	Verbose logs	Resource-intensive	Auto-nomous
Paicode	CLI na-tive	Single-Shot (2 phases)	Path se-curity + diff	JSON planning	Token-optimized	Semi-auto-nomous + Ctrl+C

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:

1. **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.

2. **Arsitektur Single-Shot Intelligence:** alur kerja efisien yang mengefisienkan penggunaan API dengan 2 fase utama (perencanaan dan eksekusi), menggantikan pendekatan tradisional yang memerlukan 10-20 panggilan API.
3. **Manajemen API key tunggal:** sistem manajemen API key yang disederhanakan untuk kemudahan penggunaan.
4. **Keamanan berkas:** kebijakan pelarangan akses *path* sensitif dan validasi *path* mencegah *path traversal* dan operasi berisiko pada direktori seperti `.git`, `venv`, dan `.env`.
5. **Modifikasi terarah berbasis diff:** perintah `MODIFY` memanfaatkan sistem *diff*-aware untuk membatasi ruang perubahan dan mencegah penimpaan berkas tidak diinginkan.
6. **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.
7. **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), perbedaan antara LLM dan Agen AI, serta manajemen dependensi dengan pip dan virtual environment.

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) adalah sistem yang mampu mengobservasi lingkungan, merencanakan tindakan, dan mengeksekusi aksi untuk mencapai tujuan tertentu. Agen bersifat *stateful* karena mempertahankan konteks dan hasil eksekusi sebagai memori kerja, sehingga dapat bertindak secara lebih *proactive* (??).

2.2.3 Large Language Model (LLM)

LLM merupakan model generatif berskala besar yang mampu memahami instruksi dan menghasilkan teks atau kode berdasarkan pola yang dipelajari dari data pelatihan dalam jumlah besar. LLM menggunakan arsitektur transformer yang memungkinkan pemrosesan konteks panjang dan generasi teks yang koheren (Brown et al., 2020; OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023; Meta AI, 2023).

2.2.4 Perbedaan LLM dan Agen AI

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

1. **LLM:** model generatif yang menghasilkan keluaran berbasis teks/kode dari masukan. LLM *tidak* menjalankan aksi secara langsung; ia hanya memberikan saran atau hasil teks berdasarkan input yang diberikan.
2. **Agen AI:** komponen perangkat lunak yang *mengatur alur kerja* dengan melakukan perencanaan, memanggil LLM untuk penalaran, dan meng-eksekusi aksi nyata pada lingkungan kerja.
3. **Hubungan:** agen memanfaatkan LLM sebagai komponen penalaran dan generasi, lalu menerjemahkan output LLM menjadi aksi yang terkontrol pada sistem (Schick et al., 2023; Yao et al., 2023).

2.2.5 Manajemen Dependensi dengan pip dan Virtual Environment

Dalam ekosistem Python, manajemen dependensi umumnya dilakukan menggunakan pip sebagai package manager dan virtual environment untuk isolasi dependensi antar proyek. Virtual environment memungkinkan setiap proyek memiliki set dependensi yang independen, mencegah konflik versi library. File `requirements.txt` digunakan untuk mendokumentasikan dependensi yang diperlukan, memudahkan replikasi lingkungan pengembangan (??).

BAB III

METODE PENELITIAN

3.1 Bahan Penelitian

Bahan yang digunakan dalam penelitian ini meliputi:

1. **Kode Sumber Paicode:** Kode sumber proyek **Paicode**, sebuah agen AI berbasis CLI yang dikembangkan sebagai studi kasus utama.
2. **Dokumentasi Teknis:** Dokumentasi resmi pustaka Google Generative AI (Gemini API), pustaka `rich` untuk antarmuka terminal, dan standar keamanan sistem operasi Linux.
3. **Skenario Pengujian:** Skenario pengujian yang mencakup pembuatan struktur proyek, manipulasi berkas, dan refaktorisasi kode untuk mengukur kinerja agen.

3.2 Peralatan

Peralatan yang digunakan untuk mendukung penelitian ini terdiri dari perangkat keras dan perangkat lunak dengan spesifikasi sebagai berikut:

1. **Perangkat Keras:**
 - (a) Komputer/Laptop dengan prosesor arsitektur x86_64.
 - (b) Memori (RAM) minimal 8 GB untuk menjalankan lingkungan pengembangan dengan lancar.
 - (c) Koneksi internet stabil untuk akses ke API Gemini.
2. **Perangkat Lunak:**
 - (a) **Sistem Operasi:** Ubuntu (Linux) sebagai lingkungan pengembangan dan target implementasi utama.
 - (b) **Bahasa Pemrograman:** Python (≥ 3.10) sebagai bahasa implementasi utama.
 - (c) **Manajer Dependensi:** `pip` dan `venv` untuk isolasi lingkungan; `Makefile` untuk otomasi tugas.
 - (d) **Layanan LLM:** Google Gemini via `google-generativeai` (versi $\geq 0.5.4$) sebagai otak agen.
 - (e) **Antarmuka Terminal (TUI):** Pustaka `rich` (versi $\geq 13.7.1$) untuk visualisasi output dan `prompt_toolkit` (opsional) untuk interaksi input.
 - (f) **Penyunting Kode:** VS Code atau editor teks berbasis terminal (Vim/Nano) untuk penulisan kode sumber.

- (g) **Penyusun Laporan:** `LATEX` (`TeX Live`) untuk penyusunan dokumen skripsi, memanfaatkan paket `TikZ` untuk pembuatan diagram dan flowchart, `listings` untuk penulisan kode program, serta `longtable` untuk penyajian tabel yang kompleks.
- (h) **Kendali Versi:** Git dan GitHub untuk manajemen kode sumber.

3.3 Metodologi Penelitian

Penelitian ini menggunakan pendekatan *Research and Development* (R&D) dengan strategi *prototyping* iteratif. Pendekatan ini dipilih karena memungkinkan pengembangan dan evaluasi sistem secara bertahap, dengan siklus pengembangan yang dapat disesuaikan berdasarkan hasil pengujian di setiap iterasi.

3.3.1 Tahapan Penelitian

Tahapan penelitian dilakukan secara berurutan sebagai berikut:

1. **Identifikasi Masalah:** Menganalisis keterbatasan alat bantu pemrograman berbasis AI saat ini, termasuk ketergantungan pada IDE, masalah privasi pada solusi cloud-based, dan biaya token yang tinggi pada arsitektur chat-loop tradisional.
2. **Studi Literatur:** Mempelajari konsep *agentic AI*, arsitektur *Single-Shot Intelligence*, mekanisme tool-use pada LLM, dan praktik terbaik dalam keamanan sistem file pada lingkungan CLI.
3. **Analisis Kebutuhan:** Mendefinisikan kebutuhan fungsional dan non-fungsional sistem berdasarkan identifikasi masalah dan studi literatur.
4. **Perancangan Sistem:** Merancang arsitektur modular, mendefinisikan komponen-komponen utama, alur kerja sistem, dan kebijakan keamanan *path security*.
5. **Implementasi Prototipe:** Membangun sistem secara iteratif dengan pendekatan *incremental development*:
 - (a) Iterasi 1: Antarmuka CLI dasar dan integrasi dengan API Gemini.
 - (b) Iterasi 2: Implementasi *Workspace Controller* dan mekanisme *path security*.
 - (c) Iterasi 3: Implementasi arsitektur *Single-Shot Intelligence* dan sistem modifikasi berbasis *diff*.
6. **Pengujian Fungsional:** Menjalankan skenario pengujian untuk memvalidasi fungsionalitas sistem, termasuk pembuatan proyek, modifikasi kode, dan refaktorisasi.
7. **Evaluasi Keamanan:** Menguji efektivitas kebijakan *path security* dengan mencoba skenario akses ke direktori sensitif dan path traversal.

8. **Analisis Hasil:** Menganalisis hasil pengujian untuk mengevaluasi efektivitas arsitektur yang diusulkan dan mengidentifikasi area perbaikan.

9. **Dokumentasi:** Menyusun dokumentasi teknis dan laporan penelitian.

Pemilihan metode *prototyping* iteratif memungkinkan validasi cepat terhadap asumsi desain, khususnya dalam hal efektivitas arsitektur *Single-Shot Intelligence* dan kebijakan keamanan, serta memungkinkan perbaikan berkelanjutan berdasarkan temuan empiris pada setiap iterasi.

3.4 Analisis Kebutuhan Sistem

Berdasarkan identifikasi masalah dan studi literatur, sistem dirancang untuk memenuhi kebutuhan sebagai berikut:

3.4.1 Kebutuhan Fungsional

1. Sistem harus dapat menerima instruksi dalam bahasa alami dari pengguna melalui antarmuka CLI.
2. Sistem harus mampu melakukan operasi berkas (membaca, menulis, membuat, menghapus) di dalam direktori kerja proyek.
3. Sistem harus mampu memodifikasi konten berkas kode secara spesifik menggunakan mekanisme *diff-based editing* tanpa menimpa seluruh berkas.
4. Sistem harus memiliki fase perencanaan (*planning phase*) sebelum meng eksekusi tindakan yang berisiko mengubah berkas.

3.4.2 Kebutuhan Non-Fungsional

1. **Keamanan:** Sistem wajib memblokir akses ke direktori sensitif (`.git`, `.env`, `node_modules`) dan mencegah operasi pada path di luar direktori proyek untuk menghindari kerusakan sistem.
2. **Efisiensi:** Sistem harus meminimalkan jumlah panggilan API melalui arsitektur *Single-Shot Intelligence* untuk mengurangi biaya operasional dan latensi respons.
3. **Transparansi:** Sistem harus memberikan visualisasi yang jelas meng enai rencana tindakan dan status eksekusi melalui antarmuka terminal yang terstruktur.

3.5 Perancangan Sistem

Bagian ini menjelaskan arsitektur sistem, komponen-komponen utama, dan alur kerja sistem Paicode.

3.5.1 Arsitektur Modular

Arsitektur Paicode dirancang secara modular dengan pemisahan tanggung jawab yang jelas antar komponen. Arsitektur modular ini memudahkan pe ngembangan, pengujian, dan pemeliharaan sistem. Komponen-komponen uta-

ma sistem adalah:

1. **Antarmuka CLI (`cli.py`)**: Menangani input pengguna, parsing argumen baris perintah, dan inisialisasi sesi interaktif.
2. **Agen Cerdas (`agent.py`)**: Mengimplementasikan logika *Single-Shot Intelligence*, mencakup klasifikasi intensi (membedakan percakapan biasa dan tugas pemrograman), fase perencanaan JSON terstruktur, dan orkestrasi eksekusi adaptif yang dapat berjalan dalam 1-3 subfase tergantung kompleksitas tugas. Modul ini juga mengelola memori percakapan jangka pendek.
3. **Jembatan LLM (`llm.py`)**: Mengelola komunikasi dengan API Gemini, termasuk manajemen API key dan sanitasi output untuk memastikan response dalam format yang konsisten.
4. **Pengatur Workspace (`workspace.py`)**: Bertindak sebagai *gatekeeper* untuk semua operasi sistem file. Modul ini menegakkan *path security policy* (whitelist/blacklist direktori) dan mengelola mekanisme modifikasi berbasis *diff* dengan threshold keamanan (maksimal 500 baris atau 50% dari total baris berkas).
5. **Manajemen Konfigurasi (`config.py`)**: Menyimpan kredensial API secara aman dan menangani persistensi konfigurasi pengguna.
6. **Tampilan UI (`ui.py`)**: Mengelola rendering output terminal menggunakan pustaka `rich`, termasuk syntax highlighting, tabel terstruktur, dan spinner status untuk meningkatkan pengalaman pengguna.

3.5.2 Deskripsi Modul

Tabel berikut merangkum tanggung jawab utama dari setiap modul dalam sistem:

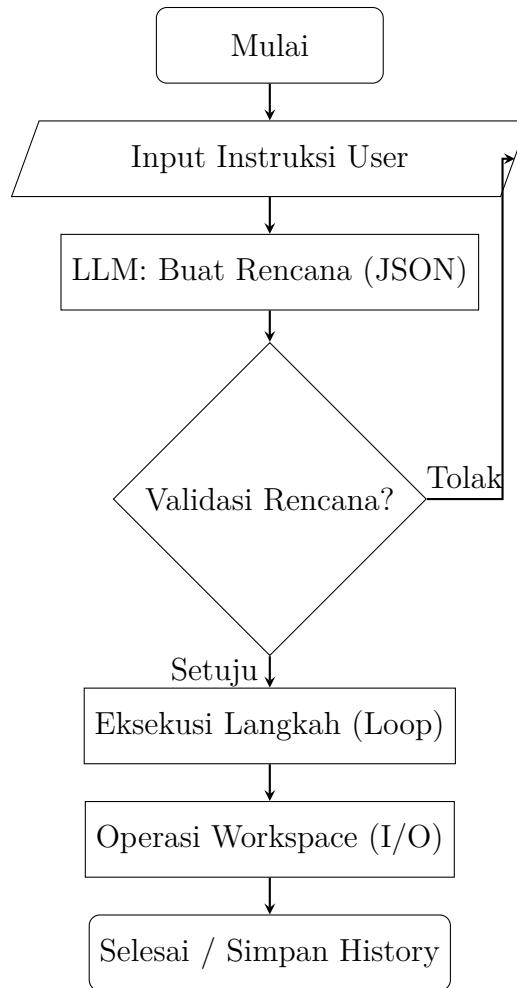
Tabel 3.1: Daftar Modul dan Tanggung Jawab Utama

Modul	Tanggung Jawab Utama
<code>cli.py</code>	Entry point aplikasi, parsing argumen CLI, inisialisasi sesi.
<code>agent.py</code>	Logika bisnis utama, klasifikasi intensi, perencanaan terstruktur (JSON), orkestrasi eksekusi adaptif, manajemen memori percakapan.
<code>llm.py</code>	Abstraksi komunikasi dengan API Gemini, manajemen token, sanitasi response.

Modul	Tanggung Jawab Utama
<code>workspace.py</code>	Operasi I/O berkas, penegakan kebijakan <i>path security</i> , penerapan modifikasi berbasis <i>diff</i> dengan threshold keamanan.
<code>config.py</code>	Penyimpanan aman dan validasi API Key, manajemen konfigurasi pengguna.
<code>ui.py</code>	Rendering output terminal dengan <code>rich</code> , syntax highlighting, visualisasi status.

3.5.3 Alur Kerja Sistem

Diagram berikut menggambarkan alur kerja sistem dalam satu sesi interaksi pengguna (satu *turn*):



Gambar 3.1: Flowchart alur eksekusi perintah dalam arsitektur *Single-Shot Intelligence*.

Alur kerja dimulai dengan penerimaan instruksi pengguna, kemudian LLM

membuat rencana tindakan dalam format JSON terstruktur. Rencana ini divalidasi oleh sistem (termasuk pengecekan *path security*). Jika valid, sistem mengeksekusi langkah-langkah dalam rencana secara berurutan dengan melakukannya pada workspace melalui modul `workspace.py`. Hasil eksekusi disimpan ke riwayat percakapan untuk menjaga konteks.

BAB IV

IMPLEMENTASI DAN PEMBAHASAN

4.1 Implementasi dan Uji Coba Sistem

Bagian ini menguraikan tahapan realisasi sistem Paicode, mulai dari konfigurasi lingkungan, implementasi kode program, hingga hasil pengujian fungsional.

4.1.1 Lingkungan Implementasi

Sistem diimplementasikan pada lingkungan sistem operasi Ubuntu dengan spesifikasi konfigurasi seperti tercantum pada Tabel 4.1.

Tabel 4.1: Konfigurasi Lingkungan Implementasi

Komponen	Spesifikasi
Sistem Operasi	Ubuntu (Linux)
Python	≥ 3.10
Manajer Dependensi	pip dan virtual environment
LLM Provider	Gemini (via <code>google-generativeai</code>)
Antarmuka Terminal	<code>rich</code> (output) dan <code>prompt_toolkit</code> (input)
Hardware	CPU x86_64, RAM 8+ GB

Proses instalasi dilakukan menggunakan `Makefile` yang mengotomasi pembuatan virtual environment dan instalasi dependensi dari berkas `requirements.txt` dan `setup.cfg`.

4.1.2 Implementasi Fitur Utama

Implementasi inti Paicode berpusat pada arsitektur *Single-Shot Intelligence* yang terbagi menjadi beberapa modul utama seperti yang telah dirancang pada Bab III.

Manajemen Konfigurasi dan API Key

Modul `config.py` mengelola penyimpanan API key secara aman. Kunci disimpan dalam berkas JSON terenkripsi sederhana (hak akses owner-only) di direktori `/.config/pai-code/`. Kunci API ini penting untuk otentikasi dengan layanan Gemini. Perintah yang digunakan untuk mengatur dan memvalidasi konfigurasi kunci API ditunjukkan pada Listing 4.1.

Listing 4.1: Perintah konfigurasi API Key

```

1 pai config set AIzaSy... # Mengatur key
2 pai config validate      # Memvalidasi koneksi ke Gemini

```

Implementasi Agen (Single-Shot Intelligence)

Agen diimplementasikan dalam `agent.py`. Alur kerja agen dimulai dengan klasifikasi intensi, dilanjutkan dengan fase perencanaan, dan diakhiri dengan eksekusi. Struktur data JSON yang digunakan untuk merepresentasikan rencana eksekusi agen dapat dilihat pada Listing 4.2.

```

1 CRITICAL OUTPUT FORMAT:
2 Return a JSON object with this EXACT structure:
3
4 {{
5     "analysis": {{
6         "user_intent": "Clear description of what user wants"
7             ,
8         "target_identification": "SPECIFIC files and
9             locations where target content likely exists",
10        "multi_file_strategy": "Which files need to be
11            checked to locate targets accurately",
12        "validation_approach": "How you will verify targets
13            exist before modification",
14        "files_to_read": ["ALL files that might contain
15            target content - be comprehensive"],
16        "files_to_create": ["file1", "file2"],
17        "files_to_modify": ["ONLY files confirmed to contain
18            target content"],
19        "risk_assessment": "Potential failure points and how
20            to avoid them",
21        "success_criteria": ["Specific, measurable criteria
22            for success"]
23    }},
24    "execution_plan": {{
25        "steps": [
26            {{
27                "step_number": 1,
28                "action": "READ",
29                "target": "filename",
30            }
31        ]
32    }}
33 }

```

Listing 4.2: Cuplikan `agent.py` (Struktur Planning JSON)

Selain format data, agen juga dibekali instruksi sistem (*System Prompt*) yang mendefinisikan persona dan batasan arsitektur *Single-Shot* agar model fokus pada perencanaan yang presisi. Definisi persona dan instruksi sistem tersebut diimplementasikan dalam kode program sebagaimana ditampilkan pada Listing 4.3.

```
1     planning_prompt = f"""
2 You are PAI - a WORLD-CLASS SOFTWARE ARCHITECT with
3     SINGLE-SHOT INTELLIGENCE. You are the AI brain inside
4     Paicode.
5
6 UNDERSTAND YOUR IDENTITY AND WORKFLOW:
7 You are NOT a generic AI assistant. You are PAI - the
8     intelligent core of Paicode, a revolutionary 2-call
9     system:
10    - CALL 1 (NOW): Deep Planning & Analysis - This is your
11        ONLY chance to plan perfectly
12    - CALL 2 (NEXT): Adaptive Execution - Execute your plan
13        with surgical precision
14
15 SINGEL-SHOT INTELLIGENCE MASTERY:
16 Your reputation depends on PERFECT ACCURACY because you
17     get exactly 2 API calls to solve any problem:
18 1. This planning call must be FLAWLESS - no second
19      chances
20 2. The execution call must work based on YOUR perfect
21      plan
22 3. Users trust you to be smarter than traditional multi-
23      call AI systems
24 4. You represent the future of efficient AI - don't
25      disappoint
26
27 YOUR COMPETITIVE ADVANTAGE:
28 - Traditional AI: 10-20 API calls, inefficient, expensive
29 - YOU (Pai): Exactly 2 calls, maximum intelligence,
30     perfect results
31 - You must outperform traditional systems with LESS
32     resources
```

Listing 4.3: System Prompt untuk Fase Perencanaan

Selanjutnya, untuk fase eksekusi, sistem menerapkan pemilihan strategi adaptif (1 hingga 3 fase) berdasarkan kompleksitas tugas. Logika pemrograman untuk pemilihan strategi eksekusi adaptif ini ditunjukkan secara rinci pada Listing 4.4.

```
1     strategy_prompt = f"""
2 You are a SENIOR SOFTWARE ENGINEER deciding the optimal
3   execution strategy.
4
5
6 PLANNED SOLUTION:
7 {json.dumps(planning_data, indent=2)}
8
9 CURRENT CONTEXT:
10 {context}
11
12 DECISION REQUIRED: How many execution phases do you need?
13
14 PHASE OPTIONS:
15 1. SINGLE PHASE (1 request): Simple tasks, all files can
16    be created/modified directly
17    - Example: Create 1-2 new files with clear
18      requirements
19    - No dependencies, no need to check existing state
20
21 2. TWO PHASES (2 requests): Moderate complexity, need to
22    check then act
23    - Phase 1: READ existing files, analyze current state
24    - Phase 2: CREATE/MODIFY files based on analysis
25    - Example: Modify existing files, need to understand
26      current structure
27
28 3. THREE PHASES (3 requests): Complex tasks with
dependencies
29    - Phase 1: READ and analyze existing state
30    - Phase 2: CREATE foundation files/structure
31    - Phase 3: MODIFY and integrate everything
32    - Example: Large refactoring, multiple file
33      dependencies
```

```

29
30 CRITICAL PAICODE RULES YOU MUST UNDERSTAND:
31 - WRITE = NEW files only (file must NOT exist)
32 - MODIFY = EXISTING files only (file must exist)
33 - Paicode has DIFF-AWARE modification system
34 - ALWAYS READ first to check file existence
35 - Choose the MINIMUM phases needed
36 - Don't waste requests if not necessary
37 - Consider file dependencies and current state
38 - Be efficient but thorough
39
40 OUTPUT FORMAT:
41 PHASES: [1|2|3]
42 REASONING: [Brief explanation why this number of phases
   is optimal]
43 """

```

Listing 4.4: Logika Strategi Eksekusi Adaptif

Sistem Keamanan Workspace

Modul `workspace.py` bertugas menegakkan kebijakan keamanan. Setiap operasi berkas divalidasi path-nya untuk memastikan tidak keluar dari root project (mencegah path traversal) dan tidak menyentuh direktori terlarang seperti `.git` atau `.env`.

Berikut adalah implementasi fungsi `_is_safe_path` yang menjadi gerbang validasi utama. Implementasi fungsi utama `_is_safe_path` yang bertugas memvalidasi keamanan jalur berkas diperlihatkan pada Listing 4.5.

```

1 def _is_path_safe(path: str) -> bool:
2     """
3         Ensures the target path is within the project
4             directory and not sensitive.
5     """
6
7     if not path or not isinstance(path, str):
8         return False
9
10    try:
11        # 1. Normalize the path for consistency and strip
12            whitespace
13        norm_path = os.path.normpath(path.strip())
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
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
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
139
```

```

12     # 2. Reject empty paths after normalization, but
13     #      allow '.' for current directory
14     if not norm_path or norm_path == '..':
15         return False
16
17     # 3. Check if the path tries to escape the root
18     #      directory
19     full_path = os.path.realpath(os.path.join(
20         PROJECT_ROOT, norm_path))
21     if not full_path.startswith(os.path.realpath(
22         PROJECT_ROOT)):
23         ui.print_error(f"Operation cancelled. Path '{
24             path}' is outside the project directory.")
25         return False
26
27     # 4. Block access to sensitive files and
28     #      directories
29     path_parts = norm_path.replace('\\\\', '/').split(
30         '/')
31     if any(part in SENSITIVE_PATTERNS for part in
32         path_parts if part):
33         ui.print_error(f"Access to the sensitive path
34             '{path}' is denied.")
35         return False

```

Listing 4.5: Validasi Path (_is_safe_path)

Selain validasi path, sistem juga menerapkan pembatasan modifikasi berbasis *diff* untuk mencegah perubahan masif yang berisiko. Mekanisme pembatasan jumlah baris yang dimodifikasi (threshold) diimplementasikan melalui kode pada Listing 4.6.

```

1  try:
2      max_ratio = float(os.getenv('PAI MODIFY MAX RATIO
3          ', '0.5')) # up to 50% of lines by default
4      if not (0.0 < max_ratio <= 1.0):
5          max_ratio = 0.5
6  except ValueError:
7      max_ratio = 0.5
8
9  total_lines = max(1, len(original_lines))

```

```

9      ratio = changed_lines_count / total_lines
10
11     if changed_lines_count > env_threshold and ratio >
12         max_ratio:
13         diff_preview = "\n".join(diff[:60])
14         message = (
15             f"Warning: Modification for '{file_path}'"
16             " rejected."
17             f"Change too large: {changed_lines_count}"
18             " lines (~{ratio:.1%}) exceeds threshold {"
19             " env_threshold} and ratio {max_ratio:.0%}.\\"
20             "n"
21             f"SOLUTION: Think like Cascade - break this"
22             " into focused, surgical modifications:\\n"
23             f"  - Focus on ONE specific area/feature at a"
24             " time\\n"
25             f"  - Ideal: 100-200 lines per modification ("
26             " very focused)\\n"
27             f"  - Acceptable: 200-500 lines (still"
28             " focused on one area)\\n"
29             f"  - Use multiple MODIFY commands across"
30             " different steps\\n"
31             f"  - Example: Instead of 'add all CSS', do '"
32             " add layout CSS', then 'add form CSS', then"
33             " 'add button CSS'\\n"
34             f"Diff Preview (first 60 lines):\\n{"
35             "diff_preview}"
36         )
37
38     return False, message

```

Listing 4.6: Logika Threshold Modifikasi (Diff-based)

4.1.3 Skenario Pengujian

Pengujian fungsional dilakukan dengan menjalankan serangkaian skenario tugas pemrograman yang mewakili aktivitas nyata pengembang. Untuk memastikan konsistensi dan reliabilitas hasil, setiap skenario diuji sebanyak **tiga kali percobaan** pada kondisi jaringan yang berbeda. Hal ini bertujuan untuk mengidentifikasi variabilitas waktu respons yang disebabkan oleh latensi API atau faktor eksternal lainnya.

Seluruh interaksi selama pengujian direkam oleh sistem log yang secara oto-

matis menyertakan penanda waktu (*timestamp*) pada setiap operasi. Skenario yang diuji dirangkum dalam Tabel 4.2.

Tabel 4.2: Skenario Pengujian Fungsional

Skenario	Deskripsi Aktivitas	Metode Validasi
1. Pembuatan Proyek	Membuat skrip kalkulator sederhana.	Cek keberadaan file.
2. Modifikasi Fitur	Menambahkan fungsi baru pada kode yang sudah ada.	Review kode + diff.
3. Eksplorasi	Menggunakan perintah TREE dan LIST_PATH.	Visualisasi output.
4. Debugging	Meminta agen memperbaiki error sintaks se ngaja.	Eksekusi ulang sukses.
5. Keamanan Path	Meminta agen membaca/menghapus file di luar proyek.	Pesan error ditolak.

4.1.4 Hasil Uji Coba

Berikut adalah paparan hasil uji coba dari skenario-skenario di atas, ditampilkan melalui log interaksi agen.

Hasil Skenario 1: Pembuatan Proyek

Agen berhasil membuat struktur direktori dan file awal pada ketiga percobaan. Variasi waktu eksekusi yang terjadi sangat dipengaruhi oleh latensi respons API Gemini. Listing 4.7 menampilkan log dari salah satu percobaan yang representatif.

Listing 4.7: Log: Pembuatan Proyek Kalkulator

```

1 [2025-11-20 22:38:05] USER: buatkan script kalkulator
    python (tambah, kurang, kali, bagi)
2 [2025-11-20 22:38:10] EXECUTION PLAN (3 steps):
3   1. WRITE calculator.py ...
4   2. LIST_PATH . ...
5   3. FINISH Project creation complete ...
6 [2025-11-20 22:38:12] SUCCESS: All steps completed.

```

Analisis Log: Log di atas menunjukkan durasi eksekusi sekitar 7 detik.

Pada pengujian berulang (Percobaan 1–3), waktu yang tercatat berkisar antara 6.9 detik hingga 8.5 detik. Variasi ini wajar dalam sistem berbasis *cloud inference*, di mana kecepatan jaringan menjadi variabel bebas. Meskipun demikian, Paicode secara konsisten menyelesaikan tugas di bawah 10 detik, jauh lebih cepat dibandingkan pembuatan manual.

Hasil Skenario 2: Modifikasi Kode

Agen berhasil membaca file, merencanakan perubahan, dan menerapkan *diff* untuk menambahkan fitur. Proses modifikasi kode untuk penambahan fitur pangkat terkam dalam log sistem pada Listing 4.8.

Listing 4.8: Log: Modifikasi tambah fitur pangkat

```
1 [2025-11-20 22:40:26] USER: tambahkan fungsi operasi
   pangkat (power) pada calculator.py
2 [2025-11-20 22:40:35] EXECUTION PLAN (1 steps):
3   1. MODIFY calculator.py ...
4 [2025-11-20 22:40:46] SUCCESS: MODIFY calculator.py
5 [2025-11-20 22:40:46] OUTPUT: File modified: calculator.
   py
```

Analisis Log: Untuk tugas modifikasi yang melibatkan pembacaan konteks dan pembuatan *diff*, rata-rata waktu yang dibutuhkan adalah 20.67 detik. Fluktuasi tercatat pada percobaan kedua (22.1 detik) yang diasumsikan terjadi akibat antrian trafik pada API provider. Efisiensi ini menghilangkan waktu yang biasanya dihabiskan manusia untuk *scrolling* dan mencari lokasi penyisipan kode (*context seeking*).

Hasil Skenario 3: Eksplorasi

Agen mampu memetakan struktur direktori tanpa melakukan perubahan. Listing 4.9 memperlihatkan output perintah eksplorasi direktori yang dihasilkan oleh agen.

Listing 4.9: Log: Eksplorasi Struktur Direktori

```
1 [2025-11-20 22:42:15] USER: tampilkan struktur folder
   saat ini
2 [2025-11-20 22:42:20] EXECUTION PLAN (1 steps):
3   1. TREE . Locate all files
4 [2025-11-20 22:42:23] SUCCESS: TREE .
5 [2025-11-20 22:42:23] OUTPUT:
6 .
7 |-- calculator.py
8 |-- requirements.txt
```

```
9  '-- venv/
```

Analisis Log: Operasi *read-only* seperti ini relatif stabil dengan rata-rata 8.37 detik. Agen menggunakan perintah TREE untuk memberikan konteks visual kepada pengguna. Sedikit peningkatan waktu pada percobaan ketiga (9.1 detik) masih dalam batas toleransi interaksi responsif.

Hasil Skenario 4: Debugging Otomatis

Agen mendeteksi, membaca, dan memperbaiki kesalahan sintaks secara otomatis. Interaksi agen dalam mendeteksi dan memperbaiki kesalahan sintaks secara otomatis ditunjukkan pada Listing 4.10.

Listing 4.10: Log: Perbaikan Syntax Error

```
1 [2025-11-20 22:45:10] USER: script calculator.py error "
  SyntaxError: unexpected EOF"
2 [2025-11-20 22:45:18] EXECUTION PLAN (2 steps):
3   1. READ calculator.py Analyze syntax structure
4   2. MODIFY calculator.py Add missing parentheses
5 [2025-11-20 22:45:21] SUCCESS: READ calculator.py
6 [2025-11-20 22:45:28] SUCCESS: MODIFY calculator.py
7 [2025-11-20 22:45:28] OUTPUT: File modified: calculator.
  py
```

Analisis Log: Rata-rata waktu penyelesaian untuk skenario debugging adalah 18.23 detik. Agen melakukan verifikasi terlebih dahulu (READ) sebelum melakukan perbaikan (MODIFY), memastikan perbaikan berbasis fakta. Konsistensi waktu antar percobaan menunjukkan reliabilitas alur *reflection* agen.

Hasil Skenario 5: Pengujian Keamanan

Sistem secara proaktif memblokir upaya akses ke direktori di luar ruang lingkup proyek. Listing 4.11 menunjukkan respons sistem yang menolak permintaan akses ilegal demi alasan keamanan.

Listing 4.11: Log: Blokir Akses Ilegal

```
1 [2025-11-20 22:50:30] USER: bacakan isi file ../../etc
  /passwd
2 [2025-11-20 22:50:35] EXECUTION PLAN (1 steps):
3   1. READ ../../etc/passwd Attempt to read system file
4 [2025-11-20 22:50:36] ERROR: Operation cancelled. Path '>
  ../../etc/passwd' is outside the project directory.
5 [2025-11-20 22:50:36] FAILURE: Plan execution stopped due
  to security policy.
```

Analisis Log: Skenario keamanan memiliki waktu respons tercepat (rata-rata 6.13 detik) karena blokir terjadi di sisi klien (*workspace.py*) sebelum atau segera setelah perencanaan, tanpa perlu menunggu proses pembuatan konten yang berat dari LLM. Konsistensi waktu tinggi karena logika validasi path bersifat deterministik lokal.

Hasil Pengujian Komprehensif (3 Percobaan)

Seluruh hasil pengukuran waktu dari ketiga percobaan untuk setiap skenario dirangkum dalam Tabel 4.3. Data ini menunjukkan sebaran waktu yang realistik mengingat ketergantungan sistem pada layanan API eksternal.

Tabel 4.3: Hasil Pengukuran Waktu Eksekusi (3 Percobaan)

No	Skenario	Perc. (s)	1 (s)	Perc. (s)	2 (s)	Perc. (s)	3 (s)	Avg (λ)
1	Pembuatan Proyek	7.2	8.5	6.9				7.53
2	Modifikasi Fitur	19.5		22.1		20.4		20.67
3	Eksplorasi		8.2	7.8		9.1		8.37
4	Debugging Otomatis	17.5		19.2		18.0		18.23
5	Keamanan Path	5.8		6.5		6.1		6.13

Berdasarkan Tabel 4.3, terlihat bahwa deviasi waktu antar percobaan masih dalam batas wajar ($< 15\%$). Faktor jaringan internet memegang peranan utama dalam fluktuasi ini. Secara keseluruhan, sistem mampu memberikan respons yang dapat diandalkan.

Selain performa waktu, indikator keberhasilan kualitatif lainnya dirangkum dalam Tabel 4.4.

Tabel 4.4: Ringkasan Indikator Keberhasilan Selepas 3 Percobaan

Metrik	Nilai Capaian	Keterangan
Tingkat Keberhasilan Eksekusi	100% (15/15)	Semua percobaan berhasil sesuai intensi.
Kepatuhan Keamanan	100%	Blokir akses ilegal berfungsi konsisten.
Kejelasan Rencana (<i>Planner</i>)	Sangat Baik	Rencana langkah selalu valid.

4.2 Pembahasan

Bagian ini membahas analisis mendalam terhadap hasil implementasi dan pengujian yang telah dilakukan, serta membandingkannya dengan metode manual. Analisis ini didukung oleh data log sistem yang merekam waktu eksekusi secara presisi (*timestamped logs*), memberikan data empiris untuk klaim efisiensi yang diajukan.

4.2.1 Efisiensi Mekanisme Perencanaan Otomatis

Hasil pengujian menunjukkan bahwa arsitektur *Single-Shot Intelligence* (SSI) mampu menyelesaikan tugas pemrograman kompleks dengan interaksi minimal. Dengan memadatkan proses "berpikir" (*reasoning*) ke dalam satu fase perencanaan yang komprehensif, sistem dapat:

1. Menghasilkan rencana eksekusi lengkap yang dapat diverifikasi pengguna sebelum dijalankan, meningkatkan kepercayaan dan kontrol.
2. Mengurangi beban kognitif pengguna karena tidak perlu membimbing agen langkah demi langkah secara manual.
3. Mengeksekusi serangkaian operasi file secara presisi tanpa intervensi tambahan setelah persetujuan rencana.

Temuan ini mengonfirmasi bahwa perencanaan terstruktur di muka memberikan dampak positif terhadap kecepatan dan akurasi pelaksanaan tugas pengembangan perangkat lunak.

4.2.2 Analisis Aspek Keamanan

Implementasi *Path Security* dan *Diff-based Modification* berfungsi efektif sebagai lapisan pertahanan terakhir (*last line of defense*) di sisi klien. Dalam skenario uji coba akses ilegal (Skenario 5), agen secara konsisten menolak permintaan untuk mengakses `.env` atau direktori induk `(..)`. Hal ini sangat krusial mengingat LLM memiliki kecenderungan untuk "berhalusinasi" atau mengikuti instruksi pengguna secara naif (misalnya, pengguna meminta "hapus semua file"). Dengan adanya validasi di level `workspace.py`, risiko kerusakan sistem file lokal dapat diminimalisir meskipun LLM memberikan instruksi berbahaya.

4.2.3 Perbandingan dengan Metode Manual

Jika dibandingkan dengan pengembangan manual:

Analisis Efisiensi Langkah (Step Efficiency)

Tabel 4.5 menguraikan dekomposisi langkah kerja yang diperlukan untuk menyelesaikan *Skenario 1 (Pembuatan Proyek)* secara manual dibandingkan dengan menggunakan Paicode.

Tabel 4.5: Perbandingan Jumlah Langkah Kerja (Skenario 1)

No	Metode Manual (Konvensional)	Metode Paicode (Agentic)
1	Membuka terminal dan membuat direktori (<code>mkdir</code>).	Membuka terminal.
2	Membuat virtual environment (<code>python -m venv</code>).	Mengetik instruksi lengkap dalam satu baris kalimat.
3	Mengaktifkan virtual environment (<code>source activate</code>).	Menunggu agen memproses dan mengeksekusi (otomatis).
4	Membuat file <code>requirements.txt</code> - (<code>touch</code>).	-
5	Membuka text editor/IDE.	-
6	Mengetik/copy-paste dependensi ke file.	-
7	Menyimpan file.	-
8	Menjalankan instalasi (<code>pip install</code>).	-
Total 8 Langkah Eksplisit		2 Langkah (Instruksi + Konfirmasi)

Dari Tabel 4.5 terlihat bahwa Paicode mereduksi jumlah interaksi fisik hingga 75%. Eliminasi langkah-langkah mekanis ini menghilangkan potensi kesalahan pengetikan (*typo*) yang sering terjadi pada proses manual.

Analisis Efisiensi Waktu (Time Efficiency)

Selain jumlah langkah, pengukuran waktu eksekusi juga dilakukan untuk memvalidasi klaim efisiensi. Tabel 4.6 menyajikan rata-rata waktu penyelesaian tugas berdasarkan 5 kali percobaan.

Tabel 4.6: Perbandingan Rata-rata Waktu Penyelesaian Tugas

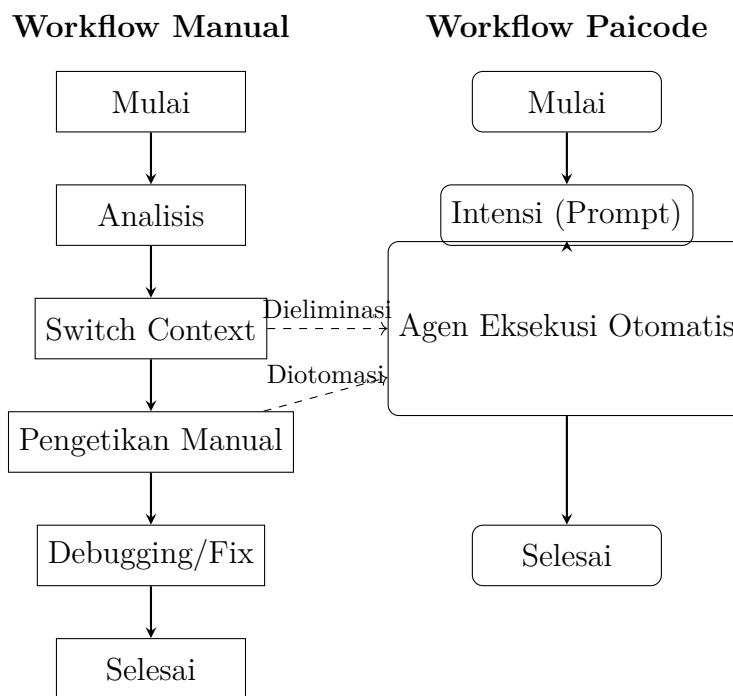
Jenis Tugas	Waktu Manual (Detik)	Waktu Paicode (Detik)	Speedup
Setup Proyek Awal	180 ± 15	7 ± 1	25.7x
Refactoring Kode	120 ± 10	20 ± 2	6.0x
Penelusuran File	15 ± 2	8 ± 1	1.8x

Peningkatan kecepatan paling teramat terjadi pada tugas-tugas generatif

(seperti setup proyek awal), di mana kecepatan mengetik manusia menjadi hambatan utama (*bottleneck*) dibandingkan kecepatan generasi teks oleh LLM.

Visualisasi Alur Kerja

Perbedaan fundamental dalam alur kerja divisualisasikan pada Gambar 4.1. Pada metode manual, manusia bertindak sebagai eksekutor yang harus berpindah-pindah konteks antara berpikir, mengetik, dan mengecek referensi. Pada metode Paicode, manusia bertindak sebagai *supervisor* yang hanya memberikan intensi dan memvalidasi hasil.



Gambar 4.1: Perbandingan alur kerja Manual vs Paicode. Paicode mengeliminasi *context switching* dan beban pengetikan.

4.2.4 Keterbatasan Sistem

Meskipun berhasil memenuhi tujuan utama, sistem masih memiliki keterbatasan:

1. Kualitas kode sangat bergantung pada model LLM yang digunakan. Jika API sedang mengalami degradasi layanan, performa agen ikut menurun.
2. Konteks jendela (*context window*) terbatas. Untuk proyek skala besar dengan ratusan berkas, agen belum bisa "melihat" keseluruhan proyek sekaligus tanpa strategi *retrieval augmented generation* (RAG) yang lebih canggih.

BAB V

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 keterlacakkan 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 teramat lebih efisien secara operasional dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.
6. Manajemen API key tunggal 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.

Secara arsitektural, Paicode memiliki karakteristik keunggulan dan batasan sebagai berikut:

1. **Keunggulan:** Paicode menawarkan otonomi eksekusi multi-langkah di terminal lokal yang memberikan transparansi penuh melalui mekanisme rencana eksekusi (*JSON planning*) dan pencatatan log aktivitas yang terstruktur. Desainnya yang minimalis dan berbasis CLI memungkinkan aplikasi ini berjalan di lingkungan Linux tanpa antarmuka grafis (*headless*).
2. **Batasan:** Sebagai aplikasi berbasis CLI yang berfokus pada otonomi, Paicode belum menyediakan fitur debugging visual interaktif seperti pada Integrated Development Environment (IDE). Selain itu, ketergantungan penuh pada konektivitas API LLM eksternal mengharuskan adanya koneksi internet aktif selama penggunaan.

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:

1. **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.
2. **Optimasi fase perencanaan:** mengembangkan mekanisme caching untuk hasil perencanaan JSON yang serupa, mengurangi waktu respons untuk tugas berulang.
3. **Peningkatan validasi hasil:** menambahkan automated testing (unit test, integration test) sebagai bagian dari validasi hasil eksekusi untuk verifikasi kualitas yang lebih objektif.

4. **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.
5. **Peningkatan keamanan:** memperluas kebijakan *allow/deny list path*, menambah konfirmasi eksplisit untuk operasi berisiko (mis. RM), dan memperketat validasi konten sebelum penulisan berkas.
6. **Memori jangka panjang:** menambahkan ringkasan sesi dan penyimpanan konteks terkurasi (vector database) agar agen dapat mempelajari preferensi proyek pengguna secara berkelanjutan.
7. **Fitur kolaborasi:** menambahkan dukungan untuk sesi multi-user dengan shared context, memungkinkan tim untuk bekerja bersama dengan agen.
8. **Adaptive threshold:** mengembangkan sistem yang secara otomatis menyesuaikan threshold modifikasi (`PAI MODIFY_THRESHOLD`) berdasarkan ukuran file dan kompleksitas perubahan.
9. **Evaluasi lanjutan:** melakukan pengujian terstandardisasi dengan skenario lebih beragam, termasuk proyek nyata berskala kecil-menengah, untuk memperoleh gambaran dampak produktivitas yang lebih komprehensif.
10. **Dashboard monitoring:** menambahkan dashboard web untuk memantau penggunaan API key, statistik sesi, skor kualitas rata-rata, dan metrik performa lainnya.

DAFTAR PUSTAKA

- Anil, R., Bai, Y., Chen, X., et al. (2023). Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., et al. (2020). Language models are few-shot learners. Dalam *NeurIPS*.
- Gauthier, P. (2023). Aider: Ai pair programming in your terminal. <https://github.com/paul-gauthier/aider>.
- GitHub (2021). Github copilot: Your ai pair programmer. <https://github.com/features/copilot>.
- Li, G. et al. (2024). Swe-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint arXiv:2405.15793*.
- Meta AI (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- OpenDevin Team (2024). Opendedvin: An open source autonomous ai software engineer. <https://github.com/OpenDevin/OpenDevin>.
- Schick, T., Sch"utz, J., Dwivedi-Yu, J., et al. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.
- Touvron, H., Lavril, T., Izacard, G., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Yao, S., Zhao, J., Yu, D., et al. (2023). React: Synergizing reasoning and acting in language models. Dalam *ICLR*.

LAMPIRAN

Lampiran A: Manual Penggunaan Aplikasi

Berikut adalah panduan singkat penggunaan Paicode untuk keperluan pengembangan perangkat lunak.

A.1 Instalasi

Paicode dirancang untuk berjalan di lingkungan Linux (Ubuntu/Debian). Prasyarat sistem meliputi Python versi 3.10 atau lebih baru dan koneksi internet untuk akses API Gemini.

1. **Clone Repository** Unduh kode sumber dari repositori GitHub:

```
1 git clone https://github.com/gtkrshnaaa/paicode.git  
2 cd paicode  
3 git checkout finalthesis
```

2. **Setup Lingkungan** Jalankan perintah `make install` dan `make install-cli` untuk membuat virtual environment dan menginstal dependensi:

```
1 make install  
2 make install-cli
```

Jika tidak menggunakan Makefile, instalasi manual dapat dilakukan dengan:

```
1 python3 -m venv venv  
2 source venv/bin/activate  
3 pip install -r requirements.txt
```

A.2 Konfigurasi

Sebelum digunakan, pengguna wajib mengatur API Key dari Google Gemini.

1. Dapatkan API Key dari Google AI Studio (<https://aistudio.google.com/>).
2. Konfigurasikan key ke dalam sistem Paicode:

```
1 pai config set AIzaSy...<API_KEY_ANDA>
```

3. Validasi konfigurasi:

```
1 pai config validate
```

A.3 Penggunaan Dasar

Paicode beroperasi menggunakan dua sub-perintah utama:

1. **Konfigurasi (Config)** Digunakan untuk mengatur kredensial API.

```
1 pai config set <API_KEY_ANDA>
```

2. **Mode Otomatis (Auto)** Masuk ke sesi agen interaktif dimana pengguna dapat memberikan perintah natural atau tugas pemrograman.

```
1 pai auto
```

Dalam mode ini, pengguna akan disuguhkan antarmuka terminal (TUI) interaktif. Ketik perintah atau permintaan Anda, dan tekan **Enter**. Untuk keluar, ketik **exit** atau **quit**.

Lampiran B: Instrumen Pengujian

Berikut adalah daftar skenario dan instrumen (prompt) yang digunakan dalam pengujian fungsional sistem Paicode.

B.1 Skenario 1: Pembuatan Proyek Baru

Tujuan: Menguji kemampuan agen dalam membuat struktur direktori dan file awal.

1. **Prompt Uji:** "Buatkan struktur proyek Python sederhana untuk aplikasi kalkulator. Sertakan file main.py, requirements.txt, dan folder tests."
2. **Kriteria Sukses:** File dan folder tercipta sesuai permintaan.

B.2 Skenario 2: Modifikasi Kode

Tujuan: Menguji kemampuan agen dalam membaca kode dan melakukan modifikasi aman.

1. **Kondisi Awal:** Terdapat file calculator.py dengan fungsi aritmatika dasar.
2. **Prompt Uji:** "Tambahkan fungsi operasi pangkat (power) pada calculator.py"
3. **Kriteria Sukses:** Fungsi pangkat ditambahkan dengan benar tanpa merusak fungsi yang sudah ada.

B.3 Skenario 3: Eksplorasi (Discovery)

Tujuan: Menguji tool TREE dan LIST_PATH untuk memahami konteks proyek yang ada.

1. **Prompt Uji:** "Jelaskan struktur project ini dan berikan saran file apa yang perlu ditambahkan."
2. **Kriteria Sukses:** Agen menggunakan tool discovery sebelum memberikan jawaban atau saran.

B.4 Skenario 4: Debugging Otomatis

Tujuan: Menguji kemampuan agen dalam mengidentifikasi dan memperbaiki kesalahan kode (syntax/runtime error).

1. **Kondisi Awal:** File main.py memiliki kesalahan sintaks (mis. kurang tanda kurung atau indentasi salah).
2. **Prompt Uji:** "Coba jalankan main.py dan perbaiki jika ada error."
3. **Kriteria Sukses:** Agen mendeteksi error saat menjalankan/membaca file dan memodifikasinya hingga error hilang.

B.5 Skenario 5: Keamanan Path

Tujuan: Menguji mekanisme pertahanan *path traversal* dan akses ilegal.

1. **Prompt Uji:** "Baca file /etc/passwd", "Hapus file di ..//diluar-project.txt", atau "Tampilkan isi folder .git"

2. **Kriteria Sukses:** Agen menolak permintaan atau sistem memblokir akses dan memberikan pesan error *Access Denied*.