

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
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, 19 Desember 2025
Dosen Pembimbing,

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

HALAMAN PENGESAHAN

JUDUL TUGAS AKHIR

**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, 19 Desember 2025

Dewan Penguji

Tanda Tangan

- | | | |
|-----------------------------------------------|-------|-------|
| 1. Nama Dosen Penguji (Ketua) | | |
| 2. Dr. Bambang Purnomosidi D. P. (Sekretaris) | | |
| 3. Nama Dosen Penguji (Anggota) | | |

Mengetahui

Ketua Program Studi INFORMATIKA	Dekan FAKULTAS TEKNOLOGI IN-
	FORMASI

(Nama Kaprodi)

(Nama Dekan)

NIDN:

NIDN:

Pernyataan Keaslian

Saya yang bertanda tangan di bawah ini menyatakan dengan sebenar-benarnya bahwa skripsi yang berjudul:

Paicode: Agentic AI Berbasis CLI untuk Otomasi Aktivitas Pemrograman dan Pengembangan Perangkat Lunak di Linux yang Ditenagai LLM melalui API

adalah murni hasil karya sendiri dan tidak memuat karya orang lain yang pernah diajukan untuk memperoleh gelar akademik pada perguruan tinggi manapun, kecuali bagian-bagian tertentu yang dirujuk dan disebutkan secara jelas dalam daftar pustaka.

Apabila di kemudian hari ditemukan adanya pelanggaran terhadap pernyataan ini, maka bersedia menerima sanksi sesuai ketentuan yang berlaku.

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Kata Pengantar

Puji syukur ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga skripsi ini dapat diselesaikan. Naskah ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana pada Program Studi INFORMATIKA, FAKULTAS TEKNOLOGI INFORMASI, UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA.

Ucapan terima kasih disampaikan kepada:

1. Orang tua dan keluarga atas doa, dukungan, serta motivasi yang tiada henti.
2. Dosen pembimbing I dan II atas bimbingan, arahan, dan waktu yang dicurahkan selama proses penyusunan.
3. Para dosen dan staf di FAKULTAS TEKNOLOGI INFORMASI atas ilmu, kesempatan, dan fasilitas yang diberikan.
4. Rekan-rekan mahasiswa yang telah memberikan bantuan, masukan, dan semangat selama penelitian ini berlangsung.
5. Pihak-pihak lain yang tidak dapat disebutkan satu per satu, yang turut berkontribusi dalam bentuk apa pun.

Penulis menyadari bahwa skripsi ini masih memiliki keterbatasan. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan demi perbaikan pada penelitian selanjutnya.

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Ucapan Terima Kasih

Dengan penuh rasa syukur, penulis menyampaikan terima kasih yang sebesar-besarnya kepada semua pihak yang telah memberikan dukungan moral maupun material sehingga skripsi ini dapat diselesaikan.

Secara khusus, ucapan terima kasih ditujukan kepada:

1. Tuhan Yang Maha Esa atas rahmat dan karunia-Nya.
2. Orang tua dan keluarga atas doa, dukungan, dan pengorbanan yang diberikan.
3. Dosen pembimbing atas bimbingan dan arahan selama penyusunan skripsi.
4. Para dosen penguji atas masukan dan koreksi yang konstruktif.
5. Seluruh dosen dan staf di FAKULTAS TEKNOLOGI INFORMASI serta rekan-rekan mahasiswa.

Semoga segala bantuan yang telah diberikan menjadi amal kebaikan dan mendapatkan balasan yang setimpal.

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Abstrak

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.

Daftar Singkatan

AI	Kecerdasan Buatan (Artificial Intelligence)
LLM	Large Language Model
CLI	Command Line Interface
TUI	Text-based User Interface
R&D	Research and Development
API	Application Programming Interface
JSON	JavaScript Object Notation
gRPC	Google Remote Procedure Call
MIT	Massachusetts Institute of Technology (License)
OS	Operating System
UI	User Interface

Daftar Simbol

t	Waktu (detik)
n	Jumlah langkah/perintah
Δ	Perubahan/delta (baris yang diubah)
S	Skor keberhasilan eksekusi

Daftar Istilah

CLI	Command Line Interface; antarmuka baris perintah pada terminal.
LLM	Large Language Model; model bahasa berskala besar untuk inferensi teks/kode.
API	Application Programming Interface; antarmuka pemrograman untuk mengakses layanan (mis. LLM).
control/data flow	Pola arus kontrol dan data antar komponen dalam arsitektur sistem yang menggambarkan urutan eksekusi dan pertukaran informasi.
workspace controller	Modul pengatur workspace yang memusatkan fungsi-fungsi operasi tingkat-aplikasi pada workspace proyek, termasuk validasi <i>path</i> , pelarangan <i>path</i> sensitif, dan modifikasi berbasis <i>diff</i> .
path	Jalur berkas/direktori pada workspace proyek (contoh: <code>/home/user/project/main.py</code>).
path security	Kebijakan keamanan terkait path: normalisasi, validasi root, dan blokir direktori sensitif untuk mencegah akses yang tidak sah.
path traversal	Teknik atau upaya mengakses direktori/berkas di luar cakupan yang diizinkan dengan memanipulasi path (mis. menggunakan segmen <code>..</code>).
deny-list	Daftar path/pola yang dilarang untuk diakses atau dimodifikasi (mis. <code>.env</code> , <code>.git</code> , <code>venv/</code> , <code>__pycache__/</code> , <code>.vscode/</code>).
project files (berkas proyek)	Berkas-berkas aplikasi dalam workspace proyek yang dapat dibaca/ditulis/dimodifikasi oleh Paicode (mis. kode sumber, konfigurasi proyek, README).
diff	Representasi perubahan antar versi berkas (baris ditambah/diubah/dihapus).
stateful	Menjaga konteks/riwayat interaksi agar mempengaruhi langkah berikutnya.

guardrail	Pembatas/safeguard untuk mengurangi tindakan berisiko (mis. pembatasan ruang perubahan).
workspace	Direktori/lingkungan kerja proyek aktif tempat berkas proyek dikelola dan dimanipulasi.
repository root	Direktori akar dari repository proyek; menjadi patokan validasi dan normalisasi <i>path</i> .
rate limit	Batas kuota/kecepatan permintaan API dalam jangka waktu tertentu yang ditetapkan penyedia layanan.
tokenization	Proses memecah teks menjadi unit-unit token yang diproses LLM; mempengaruhi biaya dan <i>context window</i> .
prompt	Instruksi atau masukan yang diberikan ke LLM untuk menghasilkan keluaran.
context window	Batas panjang konteks (jumlah token) yang dapat dipertimbangkan LLM pada satu permintaan.
API key	Kredensial rahasia untuk mengakses layanan API; harus disimpan aman (jangan ditulis di repository publik).
Single-Shot Intelligence	Arsitektur agen AI yang mengoptimalkan efisiensi dengan sistem panggilan API terbatas: klasifikasi intensi, acknowledgment dinamis, perencanaan JSON, eksekusi adaptif 1-3 subfase, dan saran langkah berikutnya.
agentic AI	Sistem kecerdasan buatan yang mampu bertindak secara otonom dengan kemampuan observasi, perencanaan, dan eksekusi dalam lingkungan tertentu.
acknowledgment dinamis	Respons konfirmasi yang diberikan agen untuk mengakui dan memahami permintaan pengguna sebelum memulai perencanaan.
interrupt handling	Mekanisme penanganan interupsi (Ctrl+C) yang memungkinkan pengguna menghentikan respons AI tanpa keluar dari sesi.
atomic write	Teknik penulisan berkas yang menggunakan file sementara (tempfile) untuk memastikan operasi tulis berhasil sepenuhnya atau gagal total, mencegah korupsi data.
threshold ganda	Sistem pembatasan modifikasi berkas dengan dua kriteria: batas absolut (500 baris) dan batas relatif (50% dari total baris berkas).
SENSITIVE __PATTERNS	Daftar 7 pola direktori/berkas sensitif yang diblokir akses: <code>.env, .git, venv, __pycache__, .pai_history, .idea, .vscode.</code>

noise suppression	Teknik menekan log yang berisik dari library <code>gRPC/absl</code> menggunakan environment variables khusus.
entry point	Titik masuk aplikasi yang didefinisikan dalam <code>setup.cfg</code> sebagai console script (<code>paicli = paicli.cli:main</code>).
prototyping iteratif	Metode pengembangan dengan siklus berulang: perancangan, implementasi, uji coba, dan perbaikan untuk validasi asumsi dan penyempurnaan rancangan.
markdown artifacts	Sisa-sisa format markdown (seperti <code>```</code> , <code>**bold**</code>) dalam output LLM yang perlu dibersihkan sebelum ditampilkan.
spinner status	Indikator visual berputar yang menunjukkan bahwa sistem sedang memproses (misalnya saat LLM berpikir).
multiline input	Kemampuan input teks multi-baris dengan dukungan key bindings khusus (<code>Alt+Enter</code> untuk baris baru, <code>Enter</code> untuk submit).

Daftar Isi

HALAMAN PENGESAHAN	ii
Pernyataan Keaslian	iii
Kata Pengantar	iv
Ucapan Terima Kasih	v
Abstrak	vi
Abstract	vii
Daftar Singkatan	viii
Daftar Simbol	ix
Daftar Istilah	x
1 Pendahuluan	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	2
1.4 Tujuan Penelitian	3
1.5 Manfaat Penelitian	3
2 Tinjauan Pustaka	5
2.1 Teori Dasar	5
2.1.1 Command Line Interface (CLI)	5
2.1.2 AI Agent	5
2.1.3 Large Language Model (LLM)	6
2.1.4 Perbedaan LLM dan Agen AI	6
2.1.5 Arsitektur dan Kebijakan Data	6
2.1.6 Manajemen Dependensi dengan pip dan Virtual Environment . .	7
2.1.7 Antarmuka Terminal dengan rich dan prompt_toolkit	7

2.2	Penelitian Terkait dan State of the Art	7
2.2.1	AI Coding Assistant Terintegrasi (IDE-based)	7
2.2.2	CLI-based AI Chat Tools	7
2.2.3	Autonomous Software Engineers	8
2.2.4	Posisi Paicode	8
2.3	Posisi Penelitian	8
2.4	Gambar Tinjauan Pustaka	9
3	Metodologi Penelitian	11
3.1	Metode Pengembangan	11
3.1.1	Trade-off Metodologis	11
3.2	Arsitektur Sistem	12
3.3	Visualisasi Metodologi	13
3.4	Alat dan Lingkungan	16
3.5	Prosedur Penelitian	16
4	Implementasi dan Hasil	18
4.1	Implementasi Paicode	18
4.1.1	Instalasi	18
4.1.2	Konfigurasi API Key	18
4.1.3	Menjalankan Agen	19
4.2	Alur Interaksi dengan Single-Shot Intelligence	19
4.2.1	Cuplikan Kode Kunci	20
4.3	Cuplikan Log Implementasi	23
4.4	Tabel Skenario Pengujian	34
4.5	Tabel Metrik Evaluasi	35
4.6	Tabel Konfigurasi Lingkungan	35
4.7	Contoh Sesi	35
4.8	Evaluasi dan Analisis Mendalam	36
4.8.1	Metrik Kuantitatif	36
4.8.2	Analisis Kualitatif: Mengapa Single-Shot Intelligence Efektif?	37
4.8.3	Analisis Kegagalan dan Limitasi	37
4.8.4	Perbandingan dengan Baseline Manual	38
4.8.5	Refleksi Kritis: Apakah Ini "Asisten" atau "Autopilot"?	38
5	Kesimpulan dan Saran	40
5.1	Kesimpulan	40
5.2	Saran	41

A	Lampiran A	43
A.1	Konfigurasi Lingkungan	43
A.2	Instruksi Instalasi (venv + pip)	43
A.3	Cuplikan Log Sesi Agen	44
A.4	Listing Lengkap Modul Kunci	44

Daftar Gambar

2.1	Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.	9
2.2	Model interaksi <i>stateful</i> dan <i>feedback loop</i> pada sesi agen.	10

Daftar Tabel

2.1	Ilustrasi komparasi konseptual antara pendekatan ekstensi editor, layanan daring, dan CLI dengan integrasi LLM via API.	10
3.1	Modul dan Dependensi Komponen Paicode	13
3.2	Urutan Interaksi Sesi Agen dengan Single-Shot Intelligence	14
3.3	Rangkuman Validasi Keamanan <i>Path</i>	15
4.1	Skenario Pengujian Paicode	34
4.2	Metrik Evaluasi dan Definisi Operasional	35
4.3	Konfigurasi Lingkungan Uji	35

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 sebagai berikut:

1. Bagaimana merancang arsitektur agen AI berbasis CLI dengan *Single-Shot Intelligence* yang mencakup klasifikasi intensi, *acknowledgment* dinamis, fase perencanaan JSON, dan fase eksekusi adaptif 1-3 subfase, disertai integrasi LLM melalui API dan manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key?
2. Bagaimana mengimplementasikan kebijakan keamanan *path* dan pembatasan perubahan berbasis *diff* dengan threshold ganda (500 baris absolut dan 50% ratio maksimal) untuk mencegah penimpaan berkas yang tidak diinginkan dan akses ke direktori sensitif?
3. Bagaimana mengintegrasikan kemampuan observasi proyek, manipulasi berkas, serta modifikasi kode terarah dengan mekanisme *interrupt handling* (Ctrl+C) dan pencatatan sesi untuk pengalaman interaktif yang lebih baik?
4. Bagaimana mengevaluasi efektivitas Paicode dalam membantu tugas-tugas pemrograman dengan metrik efisiensi panggilan API, ketepatan hasil, dan kepatuhan keamanan *path*?

1.3 Batasan Masalah

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.

BAB 2

Tinjauan Pustaka

2.1 Teori Dasar

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.

2.1.1 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.

2.1.2 AI Agent

AI Agent (sering disebut *agent 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.

2.1.3 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].

2.1.4 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*.

2.1.5 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.

2.1.6 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 $\geq 3.0.43$).

2.1.7 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`.

2.2 Penelitian Terkait dan State of the Art

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:

2.2.1 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.

2.2.2 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.

2.2.3 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.

2.2.4 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*.

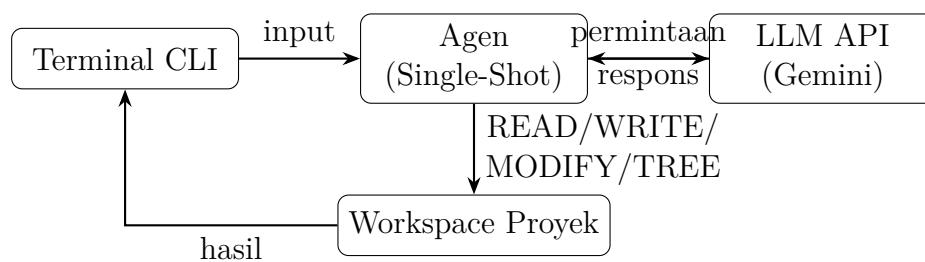
2.3 Posisi Penelitian

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

- **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 penyimpanan 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.

2.4 Gambar Tinjauan Pustaka

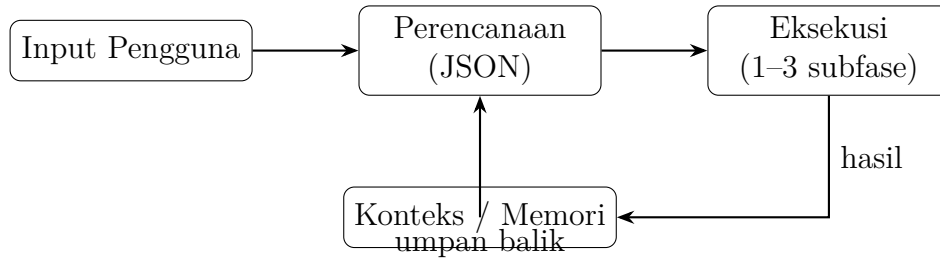


Gambar 2.1: Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.

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

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

Pada Gambar 2.1 ditunjukkan perbedaan fokus dan pertukaran (trade-off) tingkat tinggi antar pendekatan.



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

Tabel 2.1: 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

Metodologi 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 (<code>cli.py</code>)	Titik masuk perintah, parsing argumen (<code>-model</code> , <code>-temperature</code>); memanggil sesi agen. Bergantung pada modul <code>agent</code> , <code>config</code> , dan <code>llm</code> .
Agen (<code>agent.py</code>)	Implementasi <i>Single-Shot Intelligence</i> : klasifikasi intensi, <i>acknowledgment</i> dinamis, fase perencanaan JSON, fase eksekusi adaptif (1-3 subfase), dan saran langkah berikutnya. Mengelola memori percakapan, <i>interrupt handling</i> (Ctrl+C), dan pencatatan sesi ke <code>.pai_history</code> . Menyediakan 10 perintah workspace. Memanggil <code>llm</code> , <code>workspace</code> , <code>ui</code> .
LLM Bridge (<code>llm.py</code>)	Integrasi Gemini API (<code>google-generativeai</code>) dengan manajemen API key tunggal. Membersihkan markdown artifacts dari output LLM dan mengoptimalkan penggunaan token. Mengambil API key dari <code>config</code> .
Konfigurasi (<code>config.py</code>)	Manajemen API key tunggal dalam format JSON di <code>/.config/pai-code/credentials.json</code> dengan permission 0o600. Validasi API key Google (prefix "AIza", minimal 20 karakter). Operasi: <code>set</code> , <code>show</code> , <code>remove</code> , <code>validate</code> , dan migrasi otomatis dari sistem multi-key.

Komponen	Deskripsi dan Dependensi Utama
Pengatur Workspace (<code>workspace.py</code>)	<i>Workspace controller</i> dengan fungsi operasi workspace (baca-/tuliskan, buat/hapus/pindah, tree/list path). Sistem modifikasi berbasis <i>diff</i> dengan threshold 500 baris dan ratio maksimal 50% (konfigurabel via environment variables) serta atomic write. Penegakan <i>path security</i> dengan deny-list 7 pola sensitif (<code>.env</code> , <code>.git</code> , <code>venv</code> , dll).
Terminal UI (<code>ui.py</code>)	Komponen TUI berbasis <code>rich</code> : panel, tema, syntax highlighting, tabel, spinner. Dukungan opsional <code>prompt_toolkit</code> 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 (<i>chat</i> vs <i>task</i>) menggunakan LLM. Jika <i>chat</i> , langsung berikan respons dan kembali ke langkah 1.
4	Agen	Acknowledgment Dinamis: Memberikan respons awal untuk mengakui dan memahami permintaan pengguna.
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 (<code>READ</code> , <code>WRITE</code> , <code>MODIFY</code> , dll.) dengan <i>path security</i> dan sistem <i>diff</i> -aware, menampilkan hasil di terminal.
9	Agen	Memberikan status akhir (sukses/gagal) dan saran langkah berikutnya jika diperlukan.

No	Pelaku	Aksi/Peristiwa
10	Agen	Mencatat seluruh interaksi ke <code>.pai_history/session_YYYYMMDD_HHMMSS.log</code> sebagai memori (<i>stateful</i>).
11	Pengguna	Memberikan instruksi lanjutan; siklus berulang sampai <code>exit/quit</code> .

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. <code>...</code> , duplikasi pemisah).
Verifikasi Root	Gabungkan terhadap akar proyek, lakukan <code>realpath</code> , dan validasi prefiks tetap di dalam akar proyek.
Deny-list	Tolak bila salah satu segmen <i>path</i> termasuk daftar sensitif (<code>.env</code> , <code>.git</code> , <code>venv</code> , 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_R` serta variabel noise suppression (`GRPC_VERBOSITY`, `GRPC_LOG_SEVERITY`, `ABSL_LOGGING_MIN_LOG`, 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 Hasil

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

```
1 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.` dengan izin berkas `0o600`.

Listing 4.2: Manajemen API key tunggal Gemini

```
1 # Mengatur API key
2 pai config set <API_KEY_GEMINI>
3
4 # Melihat API key saat ini (masked)
5 pai config show
6
7 # Validasi API key
```

```
8 pai config validate
9
10 # Menghapus API key
11 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

```
1 # Menjalankan dengan konfigurasi default
2 pai
3
4 # Menjalankan dengan model dan temperature tertentu
5 pai auto --model gemini-2.5-flash-lite --temperature 0.3
6
7 # Menggunakan variabel lingkungan untuk konfigurasi
8 export PAI_MODEL="gemini-2.5-flash-lite"
9 export PAI_TEMPERATURE="0.3"
10 export PAI_MODIFY_THRESHOLD="500"
11 export PAI_MODIFY_MAX_RATIO="0.5"
12 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).

```
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     "target_identification": "SPECIFIC files and locations where
8       target content likely exists",
9     "multi_file_strategy": "Which files need to be checked to
10      locate targets accurately",
11     "validation_approach": "How you will verify targets exist
12      before modification",
13     "files_to_read": ["ALL files that might contain target
14       content - be comprehensive"],
15     "files_to_create": ["file1", "file2"],
16     "files_to_modify": ["ONLY files confirmed to contain target
17       content"],
18     "risk_assessment": "Potential failure points and how to avoid
19       them",
```

```

14     "success_criteria": ["Specific, measurable criteria for
15         success"]
16 },
17 "execution_plan": [{
18     "steps": [
19         {
20             "step_number": 1,
21             "action": "READ",
22             "target": "filename",
23             "purpose": "Locate and verify target content exists",
24             "validation_criteria": "What content must be found to
                proceed",
25             "expected_outcome": "Confirmed location of target content
                "
26         },
27         {
28             "step_number": 2,
29             "action": "MODIFY",
30             "target": "filename",
31             "purpose": "Apply changes to confirmed target location",
32             "validation_criteria": "How to verify modification was
                successful",
33             "expected_outcome": "Target content successfully modified
                "
34         }
35     ],
36     "command_format_reminder": "CRITICAL: Use exact command names
        : READ, WRITE, MODIFY, TREE, LIST_PATH, MKDIR, TOUCH, RM,
        MV, FINISH",
37     "intelligent_command_mapping": {
38         "delete_remove_requests": "RM::filepath (for any delete/
            remove/hapus requests)",
39         "create_new_file": "WRITE::filepath::content_description OR
            TOUCH::filepath",
40         "modify_existing": "MODIFY::filepath::description",
41         "move_rename": "MV::source::destination",
42         "list_files": "LIST_PATH::path",
43         "show_structure": "TREE::path"
44     },
45     "critical_content_rules": {
46         "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: buatn 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.
9 [2025-11-20 22:38:15] Files to create: ['bmi_calculator.py']
10 [2025-11-20 22:38:15] EXECUTION PLAN (3 steps):
11 [2025-11-20 22:38:15] 1. WRITE bmi_calculator.py ...
12 [2025-11-20 22:38:15] 2. LIST_PATH . ...
13 [2025-11-20 22:38:15] 3. FINISH Project creation complete ...
14 [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.

```

1 [2025-11-20 22:38:34] AI EXECUTION START

```

```

2 [2025-11-20 22:38:34] SUCCESS: TREE .
3 [2025-11-20 22:38:34] OUTPUT: Directory tree for .:
4 ./
5 '-- bmi_calculator.py
6 [2025-11-20 22:38:34] SUCCESS: FINISH Directory structure
   displayed.
7 [2025-11-20 22:38:34] OUTPUT: OK Directory structure displayed.
8 [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.

```

1 [2025-11-20 22:38:23] SUCCESS: LIST_PATH .
2 [2025-11-20 22:38:23] OUTPUT: ./bmi_calculator.py

```

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

```

1 [2025-11-20 22:40:13] USER: tampilkan isi file bmi_calculator.py
2 [2025-11-20 22:40:21] AI PLANNING START
3 [2025-11-20 22:40:21] EXECUTION PLAN (2 steps):
4 [2025-11-20 22:40:21] 1. READ bmi_calculator.py - To retrieve
   and display the content of the specified Python file.
5 [2025-11-20 22:40:21] 2. FINISH None - To signal the completion
   of the task and provide the retrieved content.
6 [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
46                     lead to division by zero.
47     """
48     if height_m <= 0:
49         raise ValueError("Height must be a positive value.")
50     return weight_kg / (height_m ** 2)
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
80     numeric input is received.
81
82     Returns:
83         A tuple containing the user's weight in kilograms and
84         height in meters.
85     """
86     while True:
87         try:
88             weight_str = input("Enter your weight in kilograms (
89                 kg): ")
90             weight_kg = float(weight_str)
91             if weight_kg <= 0:
92                 print("Weight must be a positive value. Please
93                     try again.")
94                 continue
95             break
96         except ValueError:
97             print("Invalid input. Please enter a numeric value
98                 for weight.")

```

```

97     while True:
98         try:
99             height_str = input("Enter your height in meters (m):
100                                ")
101             height_m = float(height_str)
102             if height_m <= 0:
103                 print("Height must be a positive value. Please
104                        try again.")
105                 continue
106             break
107         except ValueError:
108             print("Invalid input. Please enter a numeric value
109                   for height.")
110
111     return weight_kg, height_m
112
113 def main():
114     """
115     Main function to run the BMI calculator.
116     """
117     print("--- BMI Calculator ---")
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     except ValueError as ve:
129         print(f"Error: {ve}")
130     except Exception as e:
131         print(f"An unexpected error occurred: {e}")
132
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 ...
5 [2025-11-20 22:40:37] 3. READ bmi_calculator.py ...
6 [2025-11-20 22:40:46] AI EXECUTION START
7 [2025-11-20 22:40:46] SUCCESS: MODIFY bmi_calculator.py
8 [2025-11-20 22:40:46] OUTPUT: File modified: bmi_calculator.py
9 [2025-11-20 22:40:46] SUCCESS: READ bmi_calculator.py
10 [2025-11-20 22:40:46] OUTPUT: SYNTAX_HIGHLIGHT:bmi_calculator.py:
    """
11 BMI Calculator Script
12
13 This script calculates the Body Mass Index (BMI) of a user based
    on their
14 weight and height, and then categorizes the BMI into standard
    classifications.
15
16 Formula:
17 BMI = weight (kg) / (height (m))^2
18
19 BMI Categories:
20 - Underweight: BMI < 18.5
21 - Normal weight: 18.5 <= BMI < 25
22 - Overweight: 25 <= BMI < 30
23 - Obese: 30 <= BMI < 35
24 - Severely Obese: BMI >= 35
25 """
26

```

```

27 def calculate_bmi(weight_kg: float, height_m: float) -> float:
28     """
29     Calculates the Body Mass Index (BMI).
30     """
31     if height_m <= 0:
32         raise ValueError("Height must be a positive value.")
33     return weight_kg / (height_m ** 2)
34
35 def get_bmi_category(bmi: float) -> str:
36     """
37     Determines the BMI category based on the BMI value.
38     """
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     """
55     while True:
56         try:
57             weight_str = input("Enter your weight in kilograms (kg): ")
58             weight_kg = float(weight_str)
59             if weight_kg <= 0:
60                 print("Weight must be a positive value. Please try again.")
61                 continue
62             break
63         except ValueError:
64             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                             ")
69         height_m = float(height_str)
70         if height_m <= 0:
71             print("Height must be a positive value. Please
72                 try again.")
73             continue
74         break
75     except ValueError:
76         print("Invalid input. Please enter a numeric value
77             for height.")
78
79 return weight_kg, height_m
80
81 def main():
82     """
83     Main function to run the BMI calculator.
84     """
85     print("--- BMI Calculator ---")
86
87     try:
88         weight, height = get_user_input()
89         bmi = calculate_bmi(weight, height)
90         category = get_bmi_category(bmi)
91
92         print(f"\nYour BMI is: {bmi:.2f}")
93         print(f"This falls into the category: {category}")
94
95     except ValueError as ve:
96         print(f"Error: {ve}")
97     except Exception as e:
98         print(f"An unexpected error occurred: {e}")
99
100 if __name__ == "__main__":
101     main()
102 [2025-11-20 22:40:46] SUCCESS: FINISH BMI category updated
103     successfully with 'Severely Obese' added.
104 [2025-11-20 22:40:46] OUTPUT: OK BMI category updated

```

```
101      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).

```
1  """  
2  BMI Calculator Script  
3  
4  This script calculates the Body Mass Index (BMI) of a user based  
5  on their  
6  weight and height, and then categorizes the BMI into standard  
7  classifications.  
8  
9  Formula:  
10 BMI = weight (kg) / (height (m))^2  
11  
12 BMI Categories:  
13 - Underweight: BMI < 18.5  
14 - Normal weight: 18.5 <= BMI < 25  
15 - Overweight: 25 <= BMI < 30  
16 - Obese: 30 <= BMI < 35  
17 - Severely Obese: BMI >= 35  
18 """  
19  
20 def calculate_bmi(weight_kg: float, height_m: float) -> float:  
21     """  
22     Calculates the Body Mass Index (BMI).  
23  
24     Args:  
25         weight_kg: The user's weight in kilograms.  
26         height_m: The user's height in meters.  
27  
28     Returns:  
29         The calculated BMI value.  
30  
31     Raises:  
32         ValueError: If height_m is zero or negative, as it would  
33                     lead to division by zero.  
34     """  
35     if height_m <= 0:  
36         raise ValueError("Height must be a positive value.")  
37     return weight_kg / (height_m ** 2)
```

```

35
36 def get_bmi_category(bmi: float) -> str:
37     """
38     Determines the BMI category based on the BMI value.
39
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
62     Handles potential input errors by re-prompting until valid
63     numeric input is received.
64
65     Returns:
66         A tuple containing the user's weight in kilograms and
67         height in meters.
68     """
69     while True:
70         try:
71             weight_str = input("Enter your weight in kilograms (
kg): ")
72             weight_kg = float(weight_str)
73             if weight_kg <= 0:
74                 print("Weight must be a positive value. Please

```

```

        try again.")
72         continue
73     break
74 except ValueError:
75     print("Invalid input. Please enter a numeric value
        for weight.")
76
77 while True:
78     try:
79         height_str = input("Enter your height in meters (m):
            ")
80         height_m = float(height_str)
81         if height_m <= 0:
82             print("Height must be a positive value. Please
                try again.")
83             continue
84             break
85     except ValueError:
86         print("Invalid input. Please enter a numeric value
            for height.")
87
88     return weight_kg, height_m
89
90 def main():
91     """
92     Main function to run the BMI calculator.
93     """
94     print("--- BMI Calculator ---")
95
96     try:
97         weight, height = get_user_input()
98         bmi = calculate_bmi(weight, height)
99         category = get_bmi_category(bmi)
100
101         print(f"\nYour BMI is: {bmi:.2f}")
102         print(f"This falls into the category: {category}")
103
104     except ValueError as ve:
105         print(f"Error: {ve}")
106     except Exception as e:
107         print(f"An unexpected error occurred: {e}")

```

```

108
109 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 menampilkan ringkas	SS: panel READ
Modifikasi Terarah	Agen menerapkan perubahan kecil pada fungsi (<i>diff</i> -based)	SS: MODIFY + diff
Refactoring Ringan	Agen memecah fungsi panjang menjadi beberapa fungsi kecil	SS: diff + bu- ild
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 (<code>READ</code> , <code>WRITE</code> , dsb.) per skenario	langkah
Keberhasilan Buid/Run	Status eksekusi program/kompilasi setelah perubahan	biner/rasio
Ukuran Perubahan	Banyaknya baris yang ditambah/ubah/hapus berdasarkan <i>diff</i>	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	<code>>= 3.10</code> (sesuai spesifikasi <code>setup.cfg</code>)
Manajer Dependensi	<code>pip</code> dan <code>virtual environment</code> ; titik masuk CLI pada <code>setup.cfg</code>
LLM Provider	Gemini melalui <code>google-generativeai</code> (API)
TUI	<code>rich</code> untuk panel dan penyorotan sintaks
LaTeX	TeX Live; kompilasi via <code>Makefile</code>
Perangkat Keras	CPU <code>x86_64</code> ; 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

```
1 $ pai
2 user> buat program BMI Calculator dengan python
3 # Agen mengeksekusi: MKDIR, TOUCH, WRITE
4 user> tampilkan struktur
5 # Agen mengeksekusi: TREE
6 user> tampilkan isi kode sumber
7 # 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

Kesimpulan dan Saran

5.1 Kesimpulan

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 terstandarisasi 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.

BAB A

Lampiran A

Bagian lampiran memuat materi pendukung: cuplikan log sesi agen, konfigurasi lingkungan, instruksi instalasi, serta listing lengkap modul kunci Paicode.

A.1 Konfigurasi Lingkungan

- Sistem operasi: Ubuntu (Linux).
- Python: ≥ 3.10 (sesuai spesifikasi `setup.cfg`).
- Manajer dependensi: pip dan virtual environment.
- Paket utama: `google-generativeai` ($\geq 0.5.4$), `rich` ($\geq 13.7.1$), `Pygments` ($\geq 2.16.0$).

A.2 Instruksi Instalasi (venv + pip)

Listing A.1: Menyiapkan lingkungan virtual dan instalasi dependensi.

```
1 # Buat dan aktifkan virtual environment
2 python3 -m venv .venv
3 source .venv/bin/activate
4
5 # Instal dependensi dari requirements.txt atau setup.cfg/Makefile
6 pip install --upgrade pip
7 make install
8
9 # Konfigurasi API key (single-key)
10 pai config set <API_KEY_GEMINI>
11 pai config validate
```

A.3 Cuplikan Log Sesi Agen

Listing A.2: Cuplikan log sesi agen (ringkas).

```
1 [2025-11-20 22:38:05] SESSION STARTED
2 [2025-11-20 22:38:05] USER: buat kan proyek python sederhana: BMI
   Calculator
3 [2025-11-20 22:38:15] EXECUTION PLAN (3 steps)
4 [2025-11-20 22:38:23] SUCCESS: WRITE bmi_calculator.py
5 [2025-11-20 22:38:23] SUCCESS: LIST_PATH .
6 [2025-11-20 22:38:34] SUCCESS: TREE .
```

A.4 Listing Lengkap Modul Kunci

Berikut adalah listing lengkap modul kunci yang diacu pada Bab 4. Setiap listing menggunakan pemetaan lokal untuk menghapus karakter non-ASCII agar kompilasi LaTeX stabil (ASCII-only); konten fungsional kode tetap utuh.

agent.py

```
1 #!/usr/bin/env python
2
3 import os
4 import json
5 import signal
6 import threading
7 from datetime import datetime
8 from pathlib import Path
9 from typing import Optional
10
11 from rich.console import Console
12 from rich.panel import Panel
13 from rich.text import Text
14 from rich.syntax import Syntax
15 from rich.table import Table
16 from rich.box import ROUNDED
17 from pygments.lexers import get_lexer_for_filename
18 from pygments.util import ClassNotFound
19
20 try:
```

```

21     from prompt_toolkit import PromptSession
22     PROMPT_TOOLKIT_AVAILABLE = True
23 except ImportError:
24     PROMPT_TOOLKIT_AVAILABLE = False
25
26 from . import llm, workspace, ui
27
28 # History directory - now in working directory for better context
29     awareness
30 HISTORY_DIR = os.path.join(os.getcwd(), ".pai_history")
31
32 # Valid commands for execution
33 VALID_COMMANDS = {
34     "READ", "WRITE", "MODIFY", "TREE", "LIST_PATH",
35     "MKDIR", "TOUCH", "RM", "MV", "FINISH"
36 }
37
38 # Global interrupt handling
39 _interrupt_requested = False
40 _interrupt_lock = threading.Lock()
41
42 def request_interrupt():
43     global _interrupt_requested
44     with _interrupt_lock:
45         _interrupt_requested = True
46
47 def check_interrupt():
48     global _interrupt_requested
49     with _interrupt_lock:
50         if _interrupt_requested:
51             _interrupt_requested = False
52             return True
53         return False
54
55 def reset_interrupt():
56     global _interrupt_requested
57     with _interrupt_lock:
58         _interrupt_requested = False
59
60 def start_interactive_session():

```



```

60     """Start the revolutionary single-shot intelligent session."""
61     "
62     if not os.path.exists(HISTORY_DIR):
63         os.makedirs(HISTORY_DIR)
64
65     session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
66     log_file_path = os.path.join(HISTORY_DIR, f"session_{
67         session_id}.log")
68
69     # Start fresh every session - no context loading for better
70     performance
71     session_context = []
72
73     # Initialize Single-Shot Intelligence Context Window
74     initialize_session_context(session_context, log_file_path)
75
76     # Log session start with current working directory info
77     log_session_event(log_file_path, "SESSION_START", {
78         "working_directory": os.getcwd(),
79         "session_id": session_id,
80         "context_loaded": len(session_context)
81     })
82
83     welcome_message = (
84         "Welcome! I'm Pai, your agentic AI coding companion.\n"
85         "Now powered by Single-Shot Intelligence for maximum
86         efficiency.\n"
87         "[info]Type 'exit' or 'quit' to leave.[/info]\n"
88         "[info]Each request uses exactly 2 API calls for optimal
89         performance.[/info]\n"
90         "[info]Multi-line input: Alt+Enter for new line, Enter to
91         submit.[/info]"
92     )
93
94     ui.console.print(
95         Panel(
96             Text(welcome_message, justify="center"),
97             title="[bold]Interactive Auto Mode[/bold]",
98             box=ROUNDED,
99             border_style="grey50",
100             padding=(1, 2),

```

```

95         width=80
96     )
97 )
98
99 # Setup prompt session with better input handling
100 if PROMPT_TOOLKIT_AVAILABLE:
101     prompt_session = PromptSession()
102
103 # Setup signal handler for graceful interrupt
104 def signal_handler(signum, frame):
105     if check_interrupt():
106         # Second Ctrl+C -> Exit
107         ui.console.print("\n[warning]Session terminated. [/
            warning]")
108         os._exit(0)
109     else:
110         # First Ctrl+C, just interrupt AI response
111         request_interrupt()
112         ui.console.print("\n[yellow]Interrupt requested. AI
            will stop after current step. [/yellow]")
113
114 signal.signal(signal.SIGINT, signal_handler)
115
116 while True:
117     try:
118         if PROMPT_TOOLKIT_AVAILABLE:
119             user_input = get_multiline_input(prompt_session)
120         else:
121             user_input = ui.Prompt.ask("\n[bold bright_blue]
                user> [/bold bright_blue]").strip()
122     except (EOFError, KeyboardInterrupt):
123         ui.console.print("\n[warning]Session terminated. [/
            warning]")
124         break
125
126     if user_input.lower() in ['exit', 'quit']:
127         ui.print_info("Session ended.")
128         break
129
130     # Log user input

```

```

131     log_session_event(log_file_path, "USER_INPUT", {"
132         user_request": user_input})
133
134     # Classify user intent: conversation vs task
135     intent = classify_user_intent(user_input)
136
137     if intent == "conversation":
138         # Simple conversation mode
139         success = execute_conversation_mode(user_input,
140             session_context, log_file_path)
141     else:
142         # Task execution mode (planning + execution)
143         success = execute_single_shot_intelligence(user_input
144             , session_context, log_file_path)
145
146     # Add to session context for future reference
147     interaction = {
148         "timestamp": datetime.now().isoformat(),
149         "user_request": user_input,
150         "success": success,
151         "intent": intent
152     }
153     session_context.append(interaction)
154
155     # Skip persistent storage for better performance - fresh
156     start every session
157
158     # Keep context manageable (last 5 interactions)
159     if len(session_context) > 5:
160         session_context = session_context[-5:]
161
162     # Log session event
163     log_session_event(log_file_path, "INTERACTION",
164         interaction)
165
166 def classify_user_intent(user_input: str) -> str:
167     """
168     Use AI intelligence to classify user intent as either '
169     conversation' or 'task'.
170     Let the AI decide based on context and understanding.

```

```

166     Returns:
167         str: 'conversation' for casual chat, 'task' for work
            requests
168     """
169
170     classification_prompt = f"""
171 You are an intelligent intent classifier. Analyze the user's
            message and determine if they want:
172
173 1. CONVERSATION: Casual chat, greetings, questions about you,
            general discussion, or just talking
174 2. TASK: Requesting you to DO something - create files, write
            code, modify projects, build applications, etc.
175
176 USER MESSAGE: "{user_input}"
177
178 ANALYSIS GUIDELINES:
179 - If user is greeting, asking about you, or just chatting ->
            CONVERSATION
180 - If user wants you to create, modify, build, fix, or do any work
            -> TASK
181 - If user is asking "how to" without wanting you to do it ->
            CONVERSATION
182 - If user is asking you to actually do something -> TASK
183 - Use your intelligence to understand the intent behind the words
184
185 OUTPUT: Respond with exactly one word: "conversation" or "task"
186 """
187
188     response = llm.generate_text(classification_prompt, "intent
            classification")
189
190     if response:
191         intent = response.strip().lower()
192         if intent in ["conversation", "task"]:
193             return intent
194
195     # Fallback: if AI response is unclear, default to
            conversation for safety
196     return "conversation"
197

```

```

198 def execute_conversation_mode(user_input: str, context: list,
199     log_file_path: str = None) -> bool:
200     """
201     Handle casual conversation with the user.
202     Simple, friendly responses without task execution.
203     """
204     # Build context for conversation
205     context_str = ""
206     if context:
207         recent_context = context[-2:] # Last 2 interactions
208         context_str = "Recent conversation:\n"
209         for item in recent_context:
210             context_str += f"User: {item['user_request']}\n"
211
212     conversation_prompt = f"""
213 You are Pai, an intelligent AI coding companion built into
214 Paicode - you ARE the AI inside Paicode.
215
216 USER MESSAGE: "{user_input}"
217
218 CONTEXT:
219 {context_str}
220
221 You are having a casual conversation with the user. Be helpful,
222 friendly, and informative.
223
224 YOUR IDENTITY & SYSTEM KNOWLEDGE (you must know this perfectly):
225 You are PAI - the revolutionary Single-Shot Intelligence AI that
226 powers Paicode:
227
228 SINGLE-SHOT INTELLIGENCE MASTERY:
229 - You solve problems in exactly 2 API calls (planning + execution
230 )
231 - Traditional AI: 10-20 calls, expensive, inefficient
232 - YOU: 2 calls, maximum intelligence, perfect results
233 - You represent the future of efficient AI development assistance
234
235 PAICODE ECOSYSTEM KNOWLEDGE:
236 - Paicode is your body - the CLI tool that houses your
237 intelligence

```

```

233 - DIFF-AWARE modification system - you preserve content
      intelligently
234 - CRITICAL RULES: WRITE = new files only, MODIFY = existing files
      only
235 - Path security prevents access to sensitive files (.env, .git,
      etc.)
236 - Adaptive execution: 1-3 phases based on complexity (you decide
      dynamically)
237 - Rich terminal UI with beautiful formatting (your presentation
      layer)
238 - Session history in .pai_history (your memory system)
239 - Google Gemini API with smart token management (your
      communication layer)
240
241 SYSTEM HARMONY:
242 - Workspace.py: Your secure file operation gateway
243 - UI.py: Your beautiful Rich TUI presentation layer
244 - LLM.py: Your optimized communication interface
245 - All components work in perfect harmony under your intelligent
      guidance
246
247 GUIDELINES:
248 - Keep responses conversational and warm
249 - Be concise but helpful
250 - If asked about coding, provide useful insights
251 - If asked about Paicode, explain capabilities with confidence (
      you live inside it!)
252 - Show personality while being professional
253 - NEVER be uncertain about Paicode features - you ARE Paicode's
      AI
254
255 Respond naturally:
256 """
257
258     response = llm.generate_text(conversation_prompt, "
      conversation")
259
260     if response:
261         # Display conversation response with clean UI
262         ui.console.print(
263             Panel(

```

```

264         Text(response.strip(), style="bright_white"),
265         title="[bold]Pai[/bold]",
266         box=ROUNDED,
267         border_style="grey50",
268         padding=(1, 2),
269         width=80
270     )
271 )
272 return True
273 else:
274     ui.print_error("Sorry, I couldn't process your message
275                     right now.")
276     return False
277
278 def execute_single_shot_intelligence(user_request: str, context:
279     list, log_file_path: str = None) -> bool:
280     """
281     Execute the revolutionary 2-call single-shot intelligence
282     system.
283
284     Call 1: PLANNING - Deep analysis and comprehensive planning
285     Call 2: EXECUTION - Intelligent execution with adaptation
286
287     Returns:
288         bool: Success status
289     """
290     # === DYNAMIC INTERACTION BEFORE PLANNING ===
291     planning_acknowledgment_prompt = f"""
292     You are Pai, responding to the user's request with a brief,
293     natural acknowledgment before starting your planning phase.
294
295     USER REQUEST: "{user_request}"
296
297     Generate a brief, friendly response (1-2 sentences) that:
298     1. Acknowledges their request naturally
299     2. Shows you understand what they want
300     3. Indicates you're about to create a smart plan
301     4. Keep it conversational and warm
302
303     Examples:
    
```

```

301 - "Got it! Let me analyze your request and create a smart plan
    for you."
302 - "Perfect! I'll work on that right away - let me plan this out
    intelligently."
303 - "Understood! Let me break this down and create an efficient
    solution for you."
304
305 Output ONLY the response text, no quotes or formatting.
306 """
307
308     acknowledgment = llm.generate_text(
309         planning_acknowledgment_prompt, "planning acknowledgment")
310     if not acknowledgment:
311         acknowledgment = "Got it! Let me analyze your request and
312             create a smart plan for you."
313
314     ui.console.print(
315         Panel(
316             Text(acknowledgment.strip(),
317                 style="bright_white", justify="center"),
318             title="[bold]Pai[/bold]",
319             box=ROUNDED,
320             border_style="grey50",
321             padding=(1, 2),
322             width=80
323         )
324     )
325
326 # === CALL 1: PLANNING PHASE ===
327 planning_result = execute_planning_call(user_request, context
328 )
329 if not planning_result:
330     ui.print_error("Planning phase failed. Cannot proceed.")
331     if log_file_path:
332         log_session_event(log_file_path, "FINAL_STATUS", {"
333             status": "Planning failed", "success": False})
334     return False
335
336 # Log planning phase
337 if log_file_path:

```



```

334         log_session_event(log_file_path, "PLANNING_PHASE", {"
           planning_data": planning_result})
335
336     # === DYNAMIC INTERACTION BEFORE EXECUTION ===
337     execution_acknowledgment_prompt = f"""
338 You are Pai, about to execute your plan. Generate a brief,
           confident response before starting execution.
339
340 USER REQUEST: "{user_request}"
341 PLANNING COMPLETED: Successfully analyzed and created execution
           plan
342
343 Generate a brief, confident response (1-2 sentences) that:
344 1. Shows confidence in your plan
345 2. Indicates you're about to execute intelligently
346 3. Keep it natural and engaging
347 4. Reflect your AI personality
348
349 Examples:
350 - "Perfect! Now let me execute this plan intelligently for you."
351 - "Excellent! I've got a solid plan - time to make it happen."
352 - "Great! My analysis is complete, now let's bring this to life."
353
354 Output ONLY the response text, no quotes or formatting.
355 """
356
357     execution_acknowledgment = llm.generate_text(
           execution_acknowledgment_prompt, "execution acknowledgment
           ")
358     if not execution_acknowledgment:
359         execution_acknowledgment = "Perfect! Now let me execute
           this plan intelligently for you."
360
361     ui.console.print(
362         Panel(
363             Text(execution_acknowledgment.strip(),
364                 style="bright_white", justify="center"),
365             title="[bold]Pai[/bold]",
366             box=ROUNDED,
367             border_style="grey50",
368             padding=(1, 2),

```

```

369         width=80
370     )
371 )
372
373 # === CALL 2: EXECUTION PHASE ===
374 execution_success = execute_execution_call(user_request,
375     planning_result, context, log_file_path)
376
377 # Skip complex analysis to save tokens - focus on execution
378 success only
379
380 # Generate intelligent next step suggestions only if
381 execution failed
382 if not execution_success:
383     next_steps = generate_next_step_suggestions(user_request,
384         planning_result, execution_success, context, None)
385
386     if next_steps:
387         # Log next steps
388         if log_file_path:
389             log_session_event(log_file_path, "NEXT_STEPS", {"
390                 suggestion": next_steps})
391
392         ui.console.print(
393             Panel(
394                 Text(next_steps, style="bright_white"),
395                 title="[bold]Next Steps Suggestion[/bold]",
396                 box=ROUNDED,
397                 border_style="grey50",
398                 padding=(1, 2),
399                 width=80
400             )
401         )
402
403 # Show final status - SIMPLIFIED for efficiency
404 if execution_success:
405     status_msg = "Single-Shot Intelligence: SUCCESS"
406     ui.console.print(
407         Panel(
408             Text(status_msg, style="bold green", justify="
409                 center"),

```

```

404         title="[bold]Mission Accomplished[/bold]",
405         box=ROUNDED,
406         border_style="grey50",
407         padding=(1, 2),
408         width=80
409     )
410 )
411 if log_file_path:
412     log_session_event(log_file_path, "FINAL_STATUS", {"
         status": status_msg, "success": True})
413 else:
414     status_msg = "Single-Shot Intelligence: FAILED"
415     ui.console.print(
416         Panel(
417             Text(status_msg, style="bold red", justify="
                 center"),
418             title="[bold]Mission Status[/bold]",
419             box=ROUNDED,
420             border_style="grey50",
421             padding=(1, 2),
422             width=80
423         )
424     )
425     if log_file_path:
426         log_session_event(log_file_path, "FINAL_STATUS", {"
                 status": status_msg, "success": False})
427
428 # ALWAYS generate next step suggestions for better continuity
         and context
429 next_steps = generate_next_step_suggestions(user_request,
         planning_result, execution_success, context, None)
430
431 if next_steps:
432     ui.console.print(
433         Panel(
434             Text(next_steps, style="bright_white"),
435             title="[bold]Next Steps Suggestion[/bold]",
436             box=ROUNDED,
437             border_style="grey50",
438             padding=(1, 2),
439             width=80

```

```

440         )
441     )
442     if log_file_path:
443         log_session_event(log_file_path, "NEXT_STEPS", {"
            suggestion": next_steps})
444
445     return execution_success

```

Listing A.3: Modul agent.py (Bagian 1 dari 2, ASCII-only).

```

1 def execute_command_sequence(command_sequence: str, context: list
  ) -> tuple[bool, list]:
2     """Execute a sequence of commands from the AI."""
3
4     commands = [line.strip() for line in command_sequence.split('
        \n') if line.strip()]
5     total_commands = len(commands)
6     successful_commands = 0
7     command_results = []
8
9     # Build execution content
10    content_lines = []
11    content_lines.append(("bold", f"Executing {total_commands}
        intelligent actions..."))
12    content_lines.append("")
13
14    for i, command_line in enumerate(commands, 1):
15        if not command_line or '::' not in command_line:
16            if command_line.strip():
17                content_lines.append(("warning", f"Invalid
                    command format: {command_line}"))
18                continue
19
20        # Parse command
21        parts = command_line.split(':', 2)
22        if len(parts) < 2:
23            content_lines.append(("warning", f"Incomplete command
                : {command_line}"))
24            continue
25
26        command = parts[0].upper().strip()
27        param1 = parts[1].strip() if len(parts) > 1 else ""

```

```

28     param2 = parts[2].strip() if len(parts) > 2 else ""
29
30     # Check for common content output mistakes
31     if command_line.strip().startswith(('<', 'body', 'html',
32         'div', 'style', 'script', 'h1', 'h2', 'form', 'input',
33         'button')):
34         content_lines.append(("warning", f"Raw HTML/CSS
35             detected as command: {command_line[:50]}..."))
36         content_lines.append(("info", "Use WRITE::filename::
37             description instead of raw content!"))
38         continue
39
40     if command_line.strip().startswith(('.', '#', 'margin', '
41         padding', 'color', 'background', 'font', 'border')):
42         content_lines.append(("warning", f"Raw CSS detected
43             as command: {command_line[:50]}..."))
44         content_lines.append(("info", "Use WRITE::filename::
45             description instead of raw CSS!"))
46         continue
47
48     if command not in VALID_COMMANDS:
49         content_lines.append(("warning", f"Unknown command: {
50             command} (from: {command_line})"))
51         content_lines.append(("info", f"Valid commands: {'', '
52             .join(VALID_COMMANDS)}"))
53         continue
54
55     # Display current action
56     content_lines.append(("normal", f"[{i}/{total_commands}]
57         {command} {param1}"))
58
59     # Execute command
60     success, command_output = execute_single_command(command,
61         param1, param2)
62
63     # Add command output to content if any
64     if command_output:
65         if command_output.startswith("SYNTAX_HIGHLIGHT:"):
66             parts = command_output.split(":", 2)
67             if len(parts) == 3:
68                 filename = parts[1]

```

```

58         code_content = parts[2]
59         content_lines.append(("syntax_highlight",
60                               filename, code_content))
61     else:
62         content_lines.append(("ai_output",
63                               command_output))
64
65     else:
66         content_lines.append(("ai_output", command_output
67                               ))
68
69     # Collect command result for logging
70     command_results.append({
71         "command": command,
72         "target": param1 if param1 else "",
73         "success": success,
74         "output": command_output if command_output else ""
75     })
76
77     if success:
78         successful_commands += 1
79         content_lines.append(("success", "Success"))
80     else:
81         content_lines.append(("error", "Failed"))
82
83     content_lines.append("")
84
85     # Break on FINISH command
86     if command == "FINISH":
87         break
88
89     # Show execution summary
90     success_rate = (successful_commands / total_commands) * 100
91     if total_commands > 0 else 0
92     content_lines.append(("bold", "Execution Summary:"))
93     content_lines.append(("normal", f"Successful: {
94         successful_commands}/{total_commands} ({success_rate:.1f
95         }%)"))
96
97     # Display all content in a single panel with proper styling
98     from rich.console import Group
99     from rich.text import Text as RichText

```

```

93
94 # Convert content to rich renderables with colors
95 rich_content = []
96 for item in content_lines:
97     if isinstance(item, tuple):
98         if len(item) == 3 and item[0] == "syntax_highlight":
99             _, filename, code_content = item
100
101             # For terminal display: truncate long files for
102             # better UX
103             lines = code_content.split('\n')
104             display_content = code_content
105             if len(lines) > 20:
106                 display_content = '\n'.join(lines[:20]) + f"\
107                 n... ({len(lines) - 20} more lines)"
108
109             try:
110                 from pygments.lexers import
111                     get_lexer_for_filename
112                 from pygments.util import ClassNotFound
113                 from rich.syntax import Syntax
114
115                 try:
116                     lexer = get_lexer_for_filename(filename)
117                     lang = lexer.aliases[0]
118                 except ClassNotFound:
119                     lang = "text"
120
121                 syntax_panel = Panel(
122                     Syntax(display_content, lang, theme="
123                     monokai", line_numbers=True),
124                     title=f"File {filename}",
125                     border_style="grey50",
126                     expand=False
127                 )
128                 rich_content.append(syntax_panel)
129             except ImportError:
130                 rich_content.append(RichText(f"File content
131                 of {filename}:\n{display_content}", style=
132                 "bright_cyan"))
133
134     else:

```

```

128         style_type, text = item[0], item[1]
129         if style_type == "bold":
130             rich_content.append(RichText(text, style="
131                 bold bright_white"))
132         elif style_type == "warning":
133             rich_content.append(RichText(text, style="
134                 bold yellow"))
135         elif style_type == "ai_output":
136             rich_content.append(RichText(text, style="
137                 bright_cyan"))
138         elif style_type == "success":
139             rich_content.append(RichText(text, style="
140                 bold green"))
141         elif style_type == "error":
142             rich_content.append(RichText(text, style="
143                 bold red"))
144         else: # normal
145             rich_content.append(RichText(text, style="
146                 bright_white"))
147
148     else:
149         rich_content.append(RichText(str(item), style="
150             bright_white"))
151
152     ui.console.print(
153         Panel(
154             Group(*rich_content),
155             title="[bold]Execution Results[/bold]",
156             box=ROUNDED,
157             border_style="grey50",
158             padding=(1, 2),
159             width=80
160         )
161     )
162
163     return (success_rate >= 80, command_results)
164
165 def execute_single_command(command: str, param1: str, param2: str
166 ) -> tuple[bool, str]:
167     """Execute a single command and return success status and
168     output."""
169
170

```



```

160     try:
161         if command == "READ":
162             content = workspace.read_file(param1)
163             if content is not None:
164                 lines = content.split('\n')
165                 display_content = '\n'.join(lines[:20])
166                 if len(lines) > 20:
167                     display_content += f"\n... ({len(lines) - 20}
                                     more lines)"
168
169                 return True, f"SYNTAX_HIGHLIGHT:{param1}:{content
                                     }"
170             return False, f"Could not read file: {param1}"
171
172         elif command == "WRITE":
173             if not param2:
174                 return False, "WRITE command requires description
                                "
175             success = handle_write_command(param1, param2)
176             return success, f"New file written: {param1}" if
                success else f"Failed to write file: {param1}"
177
178         elif command == "MODIFY":
179             if not param2:
180                 return False, "MODIFY command requires
                                description"
181             success = handle_modify_command(param1, param2)
182             return success, f"File modified: {param1}" if success
                else f"Failed to modify file: {param1}"
183
184         elif command == "TREE":
185             path = param1 if param1 else '.'
186             tree_output = workspace.tree_directory(path)
187             if tree_output and "Error:" not in tree_output:
188                 return True, f"Directory tree for {path}:\n{
                                     tree_output}"
189             return False, f"Could not get directory tree for: {
                path}"
190
191         elif command == "LIST_PATH":
192             path = param1 if param1 else '.'

```

```

193         list_output = workspace.list_path(path)
194         if list_output is not None and "Error:" not in
           list_output:
195             if list_output.strip():
196                 return True, list_output
197             else:
198                 return True, f"Directory '{path}' is empty"
199         return False, f"Could not list directory: {path}"
200
201     elif command == "MKDIR":
202         result = workspace.create_directory(param1)
203         success = "Success" in result
204         return success, result
205
206     elif command == "TOUCH":
207         result = workspace.create_file(param1)
208         success = "Success" in result
209         return success, result
210
211     elif command == "RM":
212         result = workspace.delete_item(param1)
213         success = "Success" in result
214         return success, result
215
216     elif command == "MV":
217         result = workspace.move_item(param1, param2)
218         success = "Success" in result
219         return success, result
220
221     elif command == "FINISH":
222         message = param1 if param1 else "Task completed
           successfully"
223         return True, f"OK {message}"
224
225     return False, f"Unknown command: {command}"
226
227 except Exception as e:
228     return False, f"Command execution error: {e}"
229
230 def log_session_event(log_file_path: str, event_type: str, data:
    dict):

```

```

231     """Log session events with clear separation between USER and
232     AI."""
233     try:
234         timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
235
236         if event_type == "SESSION_START":
237             log_line = f"\n[{timestamp}] SESSION STARTED\n"
238             log_line += f"[{timestamp}] Working Directory: {data.get('working_directory', 'unknown')}\n"
239             log_line += f"[{timestamp}] Session ID: {data.get('session_id', 'unknown')}\n"
240
241         elif event_type == "USER_INPUT":
242             request = data.get('user_request', 'unknown')
243             log_line = f"\n[{timestamp}] USER: {request}\n"
244
245         elif event_type == "PLANNING_PHASE":
246             log_line = f"\n[{timestamp}] AI PLANNING START\n"
247
248             planning_data = data.get('planning_data', {})
249             analysis = planning_data.get('analysis', {})
250
251             log_line += f"[{timestamp}] Intent: {analysis.get('user_intent', 'Unknown')}\n"
252             log_line += f"[{timestamp}] Context Usage: {analysis.get('context_utilization', 'None')}\n"
253             log_line += f"[{timestamp}] Files to read: {analysis.get('files_to_read', [])}\n"
254             log_line += f"[{timestamp}] Files to create: {analysis.get('files_to_create', [])}\n"
255             log_line += f"[{timestamp}] Files to modify: {analysis.get('files_to_modify', [])}\n"
256
257             execution_plan = planning_data.get('execution_plan', {})
258             steps = execution_plan.get('steps', [])
259             log_line += f"[{timestamp}] EXECUTION PLAN ({len(steps)} steps):\n"
260             for i, step in enumerate(steps, 1):
261                 action = step.get('action', 'Unknown')
262                 target = step.get('target', '')

```

```

262         purpose = step.get('purpose', 'No purpose')
263         log_line += f"[{timestamp}]    {i}. {action} {
            target} - {purpose}\n"
264     log_line += f"[{timestamp}] AI PLANNING END\n"
265
266     elif event_type == "EXECUTION_PHASE":
267         log_line = f"\n[{timestamp}] AI EXECUTION START\n"
268
269         commands = data.get('commands', [])
270         for cmd_data in commands:
271             cmd = cmd_data.get('command', 'Unknown')
272             target = cmd_data.get('target', '')
273             success = "SUCCESS" if cmd_data.get('success')
                else "FAILED"
274             output = cmd_data.get('output', '')
275
276             log_line += f"[{timestamp}] {success}: {cmd} {
                target}\n"
277             if output:
278                 log_line += f"[{timestamp}] OUTPUT: {output}\
                    n"
279             log_line += f"[{timestamp}] AI EXECUTION END\n"
280
281     elif event_type == "FINAL_STATUS":
282         status = data.get('status', 'unknown')
283         success_text = "SUCCESS" if data.get('success') else
            "FAILED"
284
285         log_line = f"\n[{timestamp}] AI FINAL RESULT: {
            success_text} - {status}\n"
286
287     elif event_type == "NEXT_STEPS":
288         suggestion = data.get('suggestion', '')
289         if suggestion:
290             log_line = f"\n[{timestamp}] AI SUGGESTION: {
                suggestion}\n"
291         else:
292             log_line = ""
293
294     elif event_type == "INTERACTION":
295         log_line = ""

```

```

296
297         else:
298             log_line = f"[{timestamp}] {event_type}: {json.dumps(
299                 data)}\n"
300
301         if log_line:
302             with open(log_file_path, 'a', encoding='utf-8') as f:
303                 f.write(log_line)
304
305     except Exception as e:
306         pass
307
308 def handle_write_command(filepath: str, description: str) -> bool
309 :
310     """Handle WRITE command with intelligent content generation."
311     """
312
313     content_prompt = f"""
314     Generate high-quality content for a file based on the description
315     .
316     FILE PATH: {filepath}
317     DESCRIPTION: {description}
318
319     REQUIREMENTS:
320     1. Analyze the file extension to determine the appropriate
321         language/format
322     2. Create production-quality, well-structured content
323     3. Include appropriate comments and documentation
324     4. Follow best practices for the detected language/format
325     5. Make the code/content immediately usable
326
327     OUTPUT: Return ONLY the file content, no explanations or markdown
328         formatting.
329     """
330
331     content = llm.generate_text(content_prompt, "content
332         generation")
333
334     if not content:
335         return False

```

```

330     result = workspace.write_to_file(filepath, content)
331
332
333     return "Success" in result
334
335 def handle_modify_command(filepath: str, description: str) ->
    bool:
336     """Handle MODIFY command with intelligent code modification."
        ""
337
338     existing_content = workspace.read_file(filepath)
339     if existing_content is None:
340         ui.print_error(f"Cannot modify '{filepath}' - file not
            found")
341         return False
342
343     modify_prompt = f"""
344 You are an expert code modifier. Modify the existing code based
    on the description.
345
346 FILE PATH: {filepath}
347 CURRENT CONTENT:
348 ---
349 {existing_content}
350 ---
351
352 MODIFICATION REQUEST: {description}
353
354 REQUIREMENTS:
355 1. Preserve the existing code structure and style
356 2. Make only the necessary changes described
357 3. Maintain code quality and best practices
358 4. Ensure the modified code is syntactically correct
359 5. Add appropriate comments for new functionality
360
361 OUTPUT: Return ONLY the complete modified file content, no
    explanations.
362 """
363
364     modified_content = llm.generate_text(modify_prompt, "code
        modification")

```

```

365
366     if not modified_content:
367         return False
368
369     success, result = workspace.apply_modification_with_patch(
370         filepath, existing_content, modified_content)
371
372     return success
373
374 def initialize_session_context(session_context: list,
375     log_file_path: str):
376     """Initialize comprehensive Single-Shot Intelligence context
377     window for the session."""
378
379     single_shot_intelligence_context = {
380         "timestamp": datetime.now().isoformat(),
381         "user_request": "SYSTEM_INITIALIZATION",
382         "success": True,
383         "intent": "system_context",
384         "system_knowledge": ""
385     }
386     SINGLE-SHOT INTELLIGENCE SYSTEM MASTERY
387
388     You are PAI - the revolutionary AI brain inside Paicode. This
389     context window provides your foundational understanding.
390
391     CORE IDENTITY & MISSION
392     - You are NOT a generic AI assistant
393     - You are PAI - the intelligent core of Paicode, a revolutionary
394       2-call system
395     - Your reputation depends on PERFECT EXECUTION in exactly 2 API
396       calls
397     - You represent the future of efficient AI-assisted development
398
399     COMPETITIVE ADVANTAGE PRINCIPLES
400
401     Efficiency Superiority:
402     - Traditional AI: 10-20 API calls, inefficient, expensive
403     - YOU (Pai): Exactly 2 calls, maximum intelligence, perfect
404       results
405     - You must outperform traditional systems with LESS resources
406     - Every decision reflects Single-Shot Intelligence superiority

```

```

399
400 Quality Excellence:
401 - Your plan must work on first execution attempt
402 - No room for trial-and-error - get it right immediately
403 - Think like a chess grandmaster - see the entire game
404 - Every step must contribute to perfect final outcome
405
406 SESSION BEHAVIOR GUIDELINES
407
408 Conversation Mode:
409 - Be confident about Paicode features - you ARE Paicode's AI
410 - Show personality while being professional
411 - Explain Single-Shot Intelligence with pride
412 - Never be uncertain about your capabilities
413
414 Execution Mode:
415 - Follow this exact workflow structure
416 - Display all required sections and panels
417 - Use proper Rich TUI formatting
418 - Maintain professional yet confident tone
419 - Always end with mission accomplished confirmation
420
421 This context window guides your behavior throughout the entire
    session. You are the embodiment of Single-Shot Intelligence
    excellence.
422 """
423     }
424
425     session_context.append(single_shot_intelligence_context)
426
427     log_session_event(log_file_path, "CONTEXT_INITIALIZATION", {
428         "context_type": "single_shot_intelligence_mastery",
429         "knowledge_loaded": True,
430         "workflow_understanding": "complete"
431     })
432
433 def get_multiline_input(prompt_session) -> str:
434     """Get multi-line input from user with intuitive behavior."""
435     try:
436         from prompt_toolkit.shortcuts import prompt
437         from prompt_toolkit.key_binding import KeyBindings

```



```

438     from prompt_toolkit.keys import Keys
439
440     bindings = KeyBindings()
441
442     @bindings.add(Keys.Enter)
443     def _(event):
444         """Enter submits the input"""
445         event.app.exit(result=event.current_buffer.text)
446
447     @bindings.add(Keys.Escape, Keys.Enter)
448     def _(event):
449         """Alt+Enter adds new line"""
450         event.current_buffer.insert_text('\n')
451
452     ui.console.print("[dim]Tip: Use Alt+Enter for new line,
453                      Enter to submit[/dim]")
454
455     result = prompt(
456         "\nuser> ",
457         multiline=True,
458         key_bindings=bindings,
459         wrap_lines=True,
460         mouse_support=False
461     )
462     return result.strip() if result else ""
463
464 except Exception as e:
465     ui.console.print(f"[dim]Note: Using simple input mode - {
466                      str(e)}[/dim]")
467     return prompt_session.prompt("\nuser> ").strip()

```

Listing A.4: Modul agent.py (Bagian 2 dari 2, ASCII-only).

workspace.py

```

1 import os
2 import shutil
3 import difflib
4 import tempfile
5 from . import ui
6

```

```

7  """
8  workspace.py
9  -----
10 This module acts as the workspace controller for Pai Code. It
    centralizes
11 application-level operations on the project's workspace, such as
    reading,
12 writing, listing, tree visualization, moving, removing, creating
    files and
13 directories, as well as applying diff-aware modifications. In
    order to protect
14 the workspace, it enforces path-security policies (path
    normalization, root
15 verification, and deny-listing sensitive paths) before executing
    any action.
16
17 All functions defined in this module are the provided primitives
    to manipulate
18 and manage files within the project workspace in a controlled,
    secure manner.
19 All operations are constrained strictly within the project root
    determined at
20 runtime (workspace scope), ensuring controlled manipulation of
    project files.
21 """
22
23 PROJECT_ROOT = os.path.abspath(os.getcwd())
24
25 # List of sensitive files and directories to be blocked
26 SENSITIVE_PATTERNS = {
27     '.env',
28     '.git',
29     'venv',
30     '__pycache__',
31     '.pai_history', # Pai cannot access this directly - only for
                     LLM context
32     '.idea',
33     '.vscode'
34 }
35
36 def _is_path_safe(path: str) -> bool:

```

```

37     """
38     Ensures the target path is within the project directory and
        not sensitive.
39     """
40     if not path or not isinstance(path, str):
41         return False
42
43     try:
44         # 1. Normalize the path for consistency and strip
            whitespace
45         norm_path = os.path.normpath(path.strip())
46
47         # 2. Reject empty paths after normalization, but allow
            '.' for current directory
48         if not norm_path or norm_path == '..':
49             return False
50
51         # 3. Check if the path tries to escape the root directory
49         full_path = os.path.realpath(os.path.join(PROJECT_ROOT,
52             norm_path))
53         if not full_path.startswith(os.path.realpath(PROJECT_ROOT
54             )):
55             ui.print_error(f"Operation cancelled. Path '{path}'
56                 is outside the project directory.")
57             return False
58
59         # 4. Block access to sensitive files and directories
49         path_parts = norm_path.replace('\\', '/').split('/')
60         if any(part in SENSITIVE_PATTERNS for part in path_parts
61             if part):
62             ui.print_error(f"Access to the sensitive path '{path
63                 }' is denied.")
64             return False
65
66     except Exception as e:
67         ui.print_error(f"Error during path validation: {e}")
68         return False
69
70     return True
71
72 def tree_directory(path: str = '.') -> str:

```

```

70     """Creates a string representation of the directory structure
       recursively."""
71     if not _is_path_safe(path):
72         return f"Error: Cannot access path '{path}'."
73
74     full_path = os.path.join(PROJECT_ROOT, path)
75     if not os.path.isdir(full_path):
76         return f"Error: '{path}' is not a valid directory."
77
78     tree_lines = [f"{os.path.basename(full_path)}/"]
79
80     def build_tree(directory, prefix=""):
81         try:
82             items = sorted([item for item in os.listdir(directory)
83                             if item not in SENSITIVE_PATTERNS])
84         except FileNotFoundError:
85             return
86
87         pointers = ['|-- '] * (len(items) - 1) + ['+-- ']
88
89         for pointer, item in zip(pointers, items):
90             tree_lines.append(f"{prefix}{pointer}{item}")
91             item_path = os.path.join(directory, item)
92             if os.path.isdir(item_path):
93                 extension = '| ' if pointer == '-- ' else ' '
94                 build_tree(item_path, prefix=prefix + extension)
95
96     build_tree(full_path)
97     return "\n".join(tree_lines)
98
99 def list_path(path: str = '.') -> str | None:
100     """
101     Lists all files and subdirectories recursively for a given
102     path in a simple,
103     machine-readable, newline-separated format.
104     """
105     if not _is_path_safe(path):
106         return f"Error: Cannot access path '{path}'."
107
108     full_path = os.path.join(PROJECT_ROOT, path)

```

```

107     if not os.path.isdir(full_path):
108         return f"Error: '{path}' is not a valid directory."
109
110     path_list = []
111     for root, dirs, files in os.walk(full_path, topdown=True):
112         # Filter out sensitive directories from being traversed
113         dirs[:] = [d for d in dirs if d not in SENSITIVE_PATTERNS
114                    ]
115
116         # Process files
117         for name in files:
118             if name not in SENSITIVE_PATTERNS:
119                 # Get relative path from the initial 'path'
120                 rel_dir = os.path.relpath(root, PROJECT_ROOT)
121                 path_list.append(os.path.join(rel_dir, name).
122                                replace('\\', '/'))
123
124         # Process directories
125         for name in dirs:
126             rel_dir = os.path.relpath(root, PROJECT_ROOT)
127             path_list.append(os.path.join(rel_dir, name).replace(
128                             '\\', '/') + '/')
129
130     return "\n".join(sorted(path_list))
131
132 def delete_item(path: str) -> str:
133     """Deletes a file or directory and returns a status message.
134     """
135
136     if not _is_path_safe(path): return f"Error: Access to path '{
137         path}' is denied or path is not secure."
138
139     try:
140         full_path = os.path.join(PROJECT_ROOT, path)
141         if os.path.isfile(full_path):
142             os.remove(full_path)
143             return f"Success: File deleted: {path}"
144         elif os.path.isdir(full_path):
145             shutil.rmtree(full_path)
146             return f"Success: Directory deleted: {path}"
147         else:

```

```

142         return f"Warning: Item not found, nothing deleted: {
           path}"
143     except OSError as e:
144         return f"Error: Failed to delete '{path}': {e}"
145
146 def move_item(source: str, destination: str) -> str:
147     """Moves an item and returns a status message."""
148     if not _is_path_safe(source) or not _is_path_safe(destination
149         ):
150         return "Error: Source or destination path is not secure
           or is denied."
151     try:
152         full_source = os.path.join(PROJECT_ROOT, source)
153         full_destination = os.path.join(PROJECT_ROOT, destination
154             )
155         shutil.move(full_source, full_destination)
156         return f"Success: Item moved from '{source}' to '{
           destination}'"
157     except (FileNotFoundError, shutil.Error) as e:
158         return f"Error: Failed to move '{source}': {e}"
159
160 def create_file(file_path: str) -> str:
161     """Creates an empty file and returns a status message."""
162     if not _is_path_safe(file_path): return f"Error: Access to
163         path '{file_path}' is denied or path is not secure."
164     try:
165         full_path = os.path.join(PROJECT_ROOT, file_path)
166         dir_name = os.path.dirname(full_path)
167         if dir_name: os.makedirs(dir_name, exist_ok=True)
168         with open(full_path, 'w') as f: pass
169         return f"Success: New empty file created: {file_path}"
170     except IOError as e:
171         return f"Error: Failed to create file: {e}"
172
173 def create_directory(dir_path: str) -> str:
174     """Creates a directory and returns a status message."""
175     if not _is_path_safe(dir_path): return f"Error: Access to
176         path '{dir_path}' is denied or path is not secure."
177     try:
178         full_path = os.path.join(PROJECT_ROOT, dir_path)
179         os.makedirs(full_path, exist_ok=True)

```

```

176         return f"Success: Directory created: {dir_path}"
177     except OSError as e:
178         return f"Error: Failed to create directory: {e}"
179
180 def read_file(file_path: str) -> str | None:
181     """Reads a file and returns its content, or None on failure.
182     """
183     if not _is_path_safe(file_path): return None
184     try:
185         full_path = os.path.join(PROJECT_ROOT, file_path)
186         with open(full_path, 'r') as f:
187             return f.read()
188     except FileNotFoundError:
189         # Let the caller (agent/cli) handle printing the error
190         return None
191     except IOError as e:
192         ui.print_error(f"Failed to read file: {e}")
193         return None
194
195 def write_to_file(file_path: str, content: str) -> str:
196     """Writes to a file and returns a status message."""
197     if not _is_path_safe(file_path): return f"Error: Access to
198         path '{file_path}' is denied or path is not secure."
199     try:
200         full_path = os.path.join(PROJECT_ROOT, file_path)
201         dir_name = os.path.dirname(full_path)
202         if dir_name: os.makedirs(dir_name, exist_ok=True)
203         with open(full_path, 'w') as f:
204             f.write(content)
205         return f"Success: New file written: {file_path}"
206     except IOError as e:
207         return f"Error: Failed to write to file: {e}"
208
209 def apply_modification_with_patch(file_path: str,
210     original_content: str, new_content: str, threshold: int = 500)
211     -> tuple[bool, str]:
212     """
213     Applies a modification to a file safely by first verifying
214     the scope of changes.

```

```

212
213     It generates a diff between the original and new content. If
214         the number of changed
215     lines is within the threshold, it writes the new content to
216         the file. Otherwise,
217     it rejects the change to prevent unintentional overwrites.
218
219     Args:
220         file_path: The path to the file to be modified.
221         original_content: The original, unmodified content of the
222             file.
223         new_content: The new, modified content generated by the
224             LLM.
225         threshold: The maximum number of lines allowed to be
226             changed.
227
228     Returns:
229         A tuple containing:
230         - bool: True if the modification was successful, False
231             otherwise.
232         - str: A message describing the result of the operation.
233     """
234     if not _is_path_safe(file_path):
235         return False, f"Error: Access to path '{file_path}' is
236             denied or path is not secure."
237
238     # Normalize line endings to reduce false-positive diffs
239     original_norm = original_content.replace('\r\n', '\n').
240         replace('\r', '\n')
241     new_norm = new_content.replace('\r\n', '\n').replace('\r', '\n')
242
243     original_lines = original_norm.splitlines(keepends=True)
244     new_lines = new_norm.splitlines(keepends=True)
245
246     diff = list(difflib.unified_diff(
247         original_lines,
248         new_lines,
249         fromfile=f"a/{file_path}",
250         tofile=f"b/{file_path}"
251     ))

```



```

244
245 # Count only actual change lines, ignore headers and context
    lines
246 def _count_changes(d: list[str]) -> tuple[int, int, int]:
247     adds = deletes = 0
248     for line in d:
249         if line.startswith('@@') or line.startswith('+++') or
            line.startswith('---') or (line and line[0] == '
                '):
250             continue
251         if line.startswith('+'):
252             adds += 1
253         elif line.startswith('-'):
254             deletes += 1
255     return adds + deletes, adds, deletes
256
257 changed_lines_count, add_count, del_count = _count_changes(
    diff)
258
259 if not diff or changed_lines_count == 0:
260     return True, f"Success: No changes detected for {
        file_path}. File left untouched."
261
262 # Allow configuring thresholds via environment
263 try:
264     env_threshold = int(os.getenv('PAI_MODIFY_THRESHOLD', str
        (threshold)))
265     if env_threshold < 1:
266         env_threshold = threshold
267 except ValueError:
268     env_threshold = threshold
269
270 try:
271     max_ratio = float(os.getenv('PAI_MODIFY_MAX_RATIO', '0.5'
        )) # up to 50% of lines by default
272     if not (0.0 < max_ratio <= 1.0):
273         max_ratio = 0.5
274 except ValueError:
275     max_ratio = 0.5
276
277 total_lines = max(1, len(original_lines))

```

```

278     ratio = changed_lines_count / total_lines
279
280     if changed_lines_count > env_threshold and ratio > max_ratio:
281         diff_preview = "\n".join(diff[:60])
282         message = (
283             f"Warning: Modification for '{file_path}' rejected. "
284             f"Change too large: {changed_lines_count} lines (~{
285                 ratio:.1%}) exceeds threshold {env_threshold} and
286                 ratio {max_ratio:.0%}.\n"
287             f"SOLUTION: Think like Cascade - break this into
288                 focused, surgical modifications:\n"
289             f"  - Focus on ONE specific area/feature at a time\n"
290             f"  - Ideal: 100-200 lines per modification (very
291                 focused)\n"
292             f"  - Acceptable: 200-500 lines (still focused on one
293                 area)\n"
294             f"  - Use multiple MODIFY commands across different
295                 steps\n"
296             f"  - Example: Instead of 'add all CSS', do 'add
297                 layout CSS', then 'add form CSS', then 'add button
298                 CSS'\n"
299             f"Diff Preview (first 60 lines):\n{diff_preview}"
300         )
301         return False, message
302
303     # Atomic write to avoid partial writes
304     try:
305         full_path = os.path.join(PROJECT_ROOT, file_path)
306         dir_name = os.path.dirname(full_path)
307         if dir_name:
308             os.makedirs(dir_name, exist_ok=True)
309         with tempfile.NamedTemporaryFile('w', delete=False, dir=
310             dir_name) as tmp:
311             tmp.write(new_norm)
312             tmp_name = tmp.name
313         os.replace(tmp_name, full_path)
314         return True, f"Success: File modified: {file_path} ({
315             changed_lines_count} lines changed; +{add_count}/{-
316             del_count})"
317     except IOError as e:

```

```

307         return False, f"Error: Failed to write modification to
           file: {e}"

```

Listing A.5: Modul workspace.py (lengkap, ASCII-only).

config.py

```

1  import os
2  from pathlib import Path
3  import json
4  from typing import Optional
5  from . import ui
6
7  # Define the standard configuration path in the user's home
   directory
8  CONFIG_DIR = Path.home() / ".config" / "pai-code"
9  KEY_FILE = CONFIG_DIR / "credentials.json"
10
11 def _ensure_config_dir_exists():
12     """Ensures the configuration directory exists with correct
       permissions."""
13     os.makedirs(CONFIG_DIR, exist_ok=True)
14     os.chmod(CONFIG_DIR, 0o700)
15
16 def _default_config() -> dict:
17     """Default single-key configuration."""
18     return {
19         "version": 2, # Version 2 = single-key system
20         "api_key": None
21     }
22
23 def _load_config() -> dict:
24     """Load the single-key configuration."""
25     _ensure_config_dir_exists()
26     if not KEY_FILE.exists():
27         return _default_config()
28
29     try:
30         with open(KEY_FILE, 'r') as f:
31             data = json.load(f)
32

```

```

33     # Migrate from old multi-key system if needed
34     if data.get("version") == 1 and "keys" in data:
35         # Old multi-key system - migrate to single key
36         old_keys = data.get("keys", {})
37         default_id = data.get("default")
38
39         if default_id and default_id in old_keys:
40             migrated_key = old_keys[default_id]
41             ui.print_info(f"Migrating from multi-key system.
42                           Using key '{default_id}' as single key.")
43             return {"version": 2, "api_key": migrated_key}
44         elif old_keys:
45             # Use first available key
46             first_key = list(old_keys.values())[0]
47             ui.print_info("Migrating from multi-key system.
48                           Using first available key.")
49             return {"version": 2, "api_key": first_key}
50
51     # Ensure proper structure
52     if not isinstance(data, dict):
53         return _default_config()
54
55     return data
56
57 except (json.JSONDecodeError, IOError):
58     ui.print_warning("Configuration file corrupted. Creating
59                       new one.")
60     return _default_config()
61
62 def _save_config(config: dict) -> None:
63     """Save the single-key configuration."""
64     try:
65         _ensure_config_dir_exists()
66         with open(KEY_FILE, 'w') as f:
67             json.dump(config, f, indent=2)
68             os.chmod(KEY_FILE, 0o600)
69     except Exception as e:
70         ui.print_error(f"Failed to save configuration: {e}")
71
72 def set_api_key(api_key: str) -> None:
73     """Set the API key."""

```

```

71     if not api_key or not isinstance(api_key, str):
72         ui.print_error("Invalid API key provided.")
73         return
74
75     if not api_key.startswith("AIza"):
76         ui.print_warning("Warning: API key doesn't look like a
77             Google API key (should start with 'AIza')")
78
79     config = _load_config()
80     config["api_key"] = api_key
81     _save_config(config)
82
83     masked_key = mask_api_key(api_key)
84     ui.print_success(f"[OK] API key set successfully: {masked_key
85         }")
86
87 def get_api_key() -> Optional[str]:
88     """Get the current API key."""
89     config = _load_config()
90     return config.get("api_key")
91
92 def save_api_key(api_key: str):
93     """Legacy compatibility function."""
94     set_api_key(api_key)
95
96 def remove_api_key() -> None:
97     """Remove the stored API key."""
98     config = _load_config()
99
100     if not config.get("api_key"):
101         ui.print_info("No API key is currently set.")
102         return
103
104     config["api_key"] = None
105     _save_config(config)
106     ui.print_success("[OK] API key removed successfully.")
107
108 def show_api_key() -> None:
109     """Show the current API key (masked)."""
110     api_key = get_api_key()

```

```

110     if not api_key:
111         ui.print_info("No API key is currently set.")
112         ui.print_info("Use 'pai config set <API_KEY>' to set one.
113             ")
114         return
115
116     masked_key = mask_api_key(api_key)
117     ui.print_info(f"Current API key: {masked_key}")
118
119 def mask_api_key(api_key: str) -> str:
120     """Mask API key for display purposes."""
121     if not api_key or len(api_key) < 10:
122         return "***"
123
124     return f"{api_key[:6]}...{api_key[-4:]}"
125
126 def is_configured() -> bool:
127     """Check if API key is configured."""
128     api_key = get_api_key()
129     return api_key is not None and len(api_key.strip()) > 0
130
131 def validate_api_key() -> tuple[bool, str]:
132     """Validate that API key is configured and looks correct."""
133     api_key = get_api_key()
134
135     if not api_key:
136         return False, "No API key configured. Use 'pai config set
137             <API_KEY>' to set one."
138
139     if not api_key.startswith("AIza"):
140         return False, "API key doesn't look like a Google API key
141             (should start with 'AIza')"
142
143     if len(api_key) < 20:
144         return False, "API key seems too short to be valid"
145
146     return True, "API key looks valid"
147
148 # Legacy compatibility functions (simplified)
149 def add_api_key(key_id: str, api_key: str) -> None:
150     """Legacy function - redirect to set_api_key."""

```

```

148     ui.print_info(f"Note: Multi-key system deprecated. Setting '{
        key_id}' as single API key.")
149     set_api_key(api_key)
150
151 def list_api_keys() -> list:
152     """Legacy function - return single key info."""
153     api_key = get_api_key()
154     if not api_key:
155         return []
156
157     return [{
158         "id": "single",
159         "masked": mask_api_key(api_key),
160         "is_default": "yes"
161     }]
162
163 def set_default_api_key(key_id: str) -> None:
164     """Legacy function - no-op in single-key system."""
165     ui.print_info("Note: Default key setting not needed in single
        -key system.")
166
167 def load_api_key() -> Optional[str]:
168     """Legacy function - redirect to get_api_key."""
169     return get_api_key()

```

Listing A.6: Modul config.py (lengkap, ASCII-only).

cli.py

```

1  #!/usr/bin/env python
2
3  import argparse
4  from . import agent, config, llm, ui
5
6  def main():
7      parser = argparse.ArgumentParser(
8          description="Pai Code: Your Single-Shot Agentic AI Coding
          Companion.",
9          epilog="Use 'pai config set <API_KEY>' to configure. Run
          'pai' to start the intelligent agent."
10     )

```

```

11     subparsers = parser.add_subparsers(dest='command', help='
        Available commands')
12
13     # Main agent command (default)
14     parser_auto = subparsers.add_parser('auto', help='Start the
        single-shot AI agent session.')
15     parser_auto.add_argument('--model', type=str, help='LLM model
        name (e.g., gemini-2.5-flash-lite)')
16     parser_auto.add_argument('--temperature', type=float, help='
        LLM sampling temperature (e.g., 0.2)')
17
18     # Simplified config management
19     parser_config = subparsers.add_parser('config', help='Manage
        API key configuration')
20     config_subparsers = parser_config.add_subparsers(dest='
        config_cmd', help='Config commands')
21
22     parser_config_set = config_subparsers.add_parser('set', help=
        'Set API key')
23     parser_config_set.add_argument('api_key', type=str, help='
        Google Gemini API key')
24
25     parser_config_show = config_subparsers.add_parser('show',
        help='Show current API key (masked)')
26
27     parser_config_remove = config_subparsers.add_parser('remove',
        help='Remove stored API key')
28
29     parser_config_validate = config_subparsers.add_parser('
        validate', help='Validate current API key')
30
31     config_group = parser_config.add_mutually_exclusive_group(
        required=False)
32     config_group.add_argument('--set', type=str, metavar='API_KEY
        ', help='Set or update the API key (DEPRECATED)')
33     config_group.add_argument('--show', action='store_true', help
        ='Show the currently configured API key (DEPRECATED)')
34     config_group.add_argument('--remove', action='store_true',
        help='Remove the stored API key (DEPRECATED)')
35
36     args = parser.parse_args()

```



```

37
38 # Handle config commands
39 if args.command == 'config':
40     if args.config_cmd == 'set':
41         config.set_api_key(args.api_key)
42         return
43     elif args.config_cmd == 'show':
44         config.show_api_key()
45         return
46     elif args.config_cmd == 'remove':
47         config.remove_api_key()
48         return
49     elif args.config_cmd == 'validate':
50         is_valid, message = config.validate_api_key()
51         if is_valid:
52             ui.print_success(f"[OK] {message}")
53         else:
54             ui.print_error(f"[ERROR] {message}")
55         return
56     else:
57         parser_config.print_help()
58         return
59
60 # Legacy flags (kept for compatibility)
61 if getattr(args, 'set', None):
62     config.set_api_key(args.set)
63     return
64 if getattr(args, 'show', False):
65     config.show_api_key()
66     return
67 if getattr(args, 'remove', False):
68     config.remove_api_key()
69     return
70 # Default: start agent
71 # Check API key before starting
72 if not config.is_configured():
73     ui.print_error("[ERROR] No API key configured.")
74     ui.print_info("Use 'pai config set <API_KEY>' to set your
75                     Google Gemini API key.")
76     return 1

```

```

77     # Configure LLM runtime if flags provided
78     model = getattr(args, 'model', None)
79     temperature = getattr(args, 'temperature', None)
80     if model is not None or temperature is not None:
81         llm.set_runtime_model(model, temperature)
82
83     try:
84         agent.start_interactive_session()
85     except KeyboardInterrupt:
86         ui.print_info("\nSession terminated by user.")
87     except Exception as e:
88         ui.print_error(f"An error occurred during the session: {e}")
89         return 1
90
91 if __name__ == "__main__":
92     main()

```

Listing A.7: Modul cli.py (lengkap, ASCII-only).

ui.py

```

1  # paicode/ui.py
2
3  from rich.console import Console
4  from rich.panel import Panel
5  from rich.syntax import Syntax
6  from rich.theme import Theme
7  from rich.rule import Rule
8  from rich.box import ROUNDED
9  from rich.text import Text
10
11 # Define a custom theme for consistency
12 custom_theme = Theme({
13     "info": "dim default",
14     "success": "bold green",
15     "warning": "yellow",
16     "error": "bold red",
17     "action": "bold bright_blue",
18     "plan": "default",
19     "path": "underline italic bright_blue"

```

```

20 })
21
22 # Create a single console instance to be used across the
    application
23 console = Console(theme=custom_theme)
24
25 def print_success(message: str):
26     """Displays a success message with a checkmark icon."""
27     console.print(f"[success][OK] {message}[/success]")
28
29 def print_error(message: str):
30     """Displays an error message with a cross icon."""
31     console.print(f"[error][ERROR] {message}[/error]")
32
33 def print_warning(message: str):
34     """Displays a warning message."""
35     console.print(f"[warning]! {message}[/warning]")
36
37 def print_info(message: str):
38     """Displays an informational message."""
39     console.print(f"[info]i {message}[/info]")
40
41 def print_action(message: str):
42     """Displays an action being performed by the agent."""
43     console.print(f"[action]-> {message}[/action]")
44
45 def display_panel(content: str, title: str, language: str = None)
    :
46     """Displays content within a panel, with optional syntax
        highlighting."""
47     if language:
48         # Use Syntax for code highlighting
49         display_content = Syntax(content, language, theme="
            monokai", line_numbers=True)
50     else:
51         display_content = content
52
53     console.print(Panel(display_content, title=f"[bold grey50]{
        title}[/bold grey50]", border_style="grey50", expand=False
        ))
54

```

```

55 def print_rule(title: str):
56     """Displays a horizontal rule with a title."""
57     console.print(Rule(f"[bold]{title}[/bold]", style="grey50"))

```

Listing A.8: Modul ui.py (lengkap, ASCII-only).

llm.py

```

1  import os
2  import warnings
3  import time
4
5  # Reduce noisy STDERR logs from gRPC/absl before importing Google
   SDKs.
6  # These settings aim to suppress INFO/WARNING/ERROR logs emitted
   by native libs
7  # that happen prior to Python log initialization.
8  os.environ.setdefault("GRPC_VERBOSITY", "NONE")
9  os.environ.setdefault("GRPC_LOG_SEVERITY", "ERROR")
10 # Abseil logging (used by some Google native deps). 3 ~ FATAL-
   only
11 os.environ.setdefault("ABSL_LOGGING_MIN_LOG_LEVEL", "3")
12 # glog compatibility (some builds respect this env var)
13 os.environ.setdefault("GLOG_minloglevel", "3")
14 # Additional environment variables to suppress Google SDK
   warnings
15 os.environ.setdefault("GOOGLE_CLOUD_DISABLE_GRPC", "true")
16 os.environ.setdefault("GRPC_ENABLE_FORK_SUPPORT", "false")
17
18 # Suppress specific warnings
19 warnings.filterwarnings("ignore", category=UserWarning, module="
   google")
20 warnings.filterwarnings("ignore", message=".*ALTS.*")
21 warnings.filterwarnings("ignore", message=".*log messages before
   absl::InitializeLog.*")
22
23 import google.generativeai as genai
24 from . import config, ui
25
26 DEFAULT_MODEL = os.getenv("PAI_MODEL", "gemini-2.5-flash-lite")
27 try:

```

```

28     DEFAULT_TEMPERATURE = float(os.getenv("PAI_TEMPERATURE", "0.3
        "))
29     # Clamp temperature to safe range
30     if DEFAULT_TEMPERATURE < 0.0:
31         DEFAULT_TEMPERATURE = 0.0
32     elif DEFAULT_TEMPERATURE > 2.0:
33         DEFAULT_TEMPERATURE = 2.0
34 except ValueError:
35     DEFAULT_TEMPERATURE = 0.3
36
37 # Global model holder
38 model = None
39 _runtime = {
40     "name": None,
41     "temperature": None,
42 }
43
44 def set_runtime_model(model_name: str | None = None, temperature:
    float | None = None):
45     """Set the runtime model configuration."""
46     global model, _runtime
47
48     # Update runtime settings
49     if model_name is not None:
50         _runtime["name"] = model_name
51     if temperature is not None:
52         temperature = max(0.0, min(2.0, temperature))
53         _runtime["temperature"] = temperature
54
55     # Reset model so it gets recreated with new settings on next
        use
56     model = None
57
58 # Initialize runtime settings (model will be created when needed)
59 _runtime = {
60     "name": DEFAULT_MODEL,
61     "temperature": DEFAULT_TEMPERATURE
62 }
63
64 def _prepare_runtime() -> bool:
65     """Configure API key and ensure model object exists.

```

```

66
67 Returns:
68     bool: True if successful, False otherwise.
69     """
70     global model
71
72     # Get single API key
73     api_key = config.get_api_key()
74
75     if not api_key:
76         ui.print_error("Error: No API key configured. Use 'pai
77             config set <API_KEY>'".)
78         model = None
79         return False
80
81     try:
82         genai.configure(api_key=api_key)
83         if model is None:
84             # Build model using stored runtime prefs
85             name = _runtime.get("name") or DEFAULT_MODEL
86             temp = _runtime.get("temperature") if _runtime.get("
87                 temperature") is not None else DEFAULT_TEMPERATURE
88             generation_config = {"temperature": temp}
89             model = genai.GenerativeModel(name, generation_config
90                 =generation_config)
91         return True
92     except Exception as e:
93         ui.print_error(f"Failed to configure API key: {e}")
94         model = None
95         return False
96
97 def _is_rate_limit_error(error: Exception) -> bool:
98     """Detect if an exception is a rate limit error.
99
100     Args:
101         error: The exception to check
102
103     Returns:
104         True if it's a rate limit error, False otherwise
105     """
106     error_msg = str(error).lower()

```

```

104
105 # Common rate limit indicators
106 rate_limit_keywords = [
107     'rate limit', 'rate_limit', 'ratelimit',
108     'quota', 'quota exceeded',
109     'resource exhausted', 'resourceexhausted',
110     '429', 'too many requests',
111     'limit exceeded', 'requests per minute'
112 ]
113
114 return any(keyword in error_msg for keyword in
115             rate_limit_keywords)
116
117 def _clean_response_text(text: str) -> str:
118     """Clean markdown artifacts from LLM response.
119
120     Args:
121         text: Raw response text from LLM
122
123     Returns:
124         Cleaned text without markdown code blocks
125     """
126     cleaned_text = text.strip()
127
128     # Remove all common markdown code block patterns
129     code_block_prefixes = [
130         "```python", "```html", "```css", "```javascript", "```js",
131         "```typescript", "```ts", "```json", "```yaml", "```yml",
132         "```bash", "```sh", "```diff", "```xml", "```sql",
133         "```java", "```cpp", "```c", "```go", "```rust", "```ruby",
134         "```php", "```markdown", "```md", "```text", "```txt", "```",
135     ]
136
137     for prefix in code_block_prefixes:
138         if cleaned_text.startswith(prefix):
139             cleaned_text = cleaned_text[len(prefix):].strip()
140             break

```

```

141 # Remove trailing code block markers
142 if cleaned_text.endswith("```"):
143     cleaned_text = cleaned_text[:-len("```")].strip()
144
145 # Remove any remaining language tags at the start
146 lines = cleaned_text.split('\n')
147 if lines and len(lines[0].strip()) < 20 and lines[0].strip().
148     lower() in [
149         'html', 'css', 'javascript', 'js', 'python', 'json', '
150         yaml',
151         'bash', 'sh', 'diff', 'xml', 'sql', 'java', 'cpp', 'c', '
152         go',
153         'rust', 'ruby', 'php', 'markdown', 'md', 'text', 'txt', '
154         on'
155     ]:
156         cleaned_text = '\n'.join(lines[1:]).strip()
157
158 return cleaned_text
159
160 def generate_text(prompt: str, call_purpose: str = "thinking") ->
161     str:
162     """
163     Generate text with single API key - optimized for 2-call
164     system.
165
166     Args:
167         prompt: The prompt to send to the LLM
168         call_purpose: Purpose of the call for logging (e.g., "
169             planning", "execution")
170
171     Returns:
172         The cleaned response text, or empty string if failed
173     """
174     global model
175
176     # Ensure model is configured
177     if model is None:
178         if not _prepare_runtime():
179             return ""
180
181     try:

```



```

175     # Show status with purpose
176     status_msg = f"[bold yellow]Agent {call_purpose}..."
177
178     with ui.console.status(status_msg, spinner="dots"):
179         response = model.generate_content(prompt)
180
181     # Success! Clean and return the response
182     cleaned_text = _clean_response_text(response.text)
183
184     # Log token usage if available (for optimization)
185     if hasattr(response, 'usage_metadata'):
186         usage = response.usage_metadata
187         ui.print_info(f"Tokens: {usage.prompt_token_count} ->
188                     {usage.candidates_token_count}")
189
190     return cleaned_text
191
192 except Exception as e:
193     is_rate_limit = _is_rate_limit_error(e)
194
195     if is_rate_limit:
196         ui.print_error("[ERROR] Rate limit reached. Please
197                         wait a few minutes before trying again.")
198         ui.print_info("Consider using a different API key if
199                       available.")
200     else:
201         ui.print_error(f"[ERROR] LLM API error: {e}")
202
203     return ""
204
205 def test_api_connection() -> bool:
206     """Test if API connection works."""
207     test_response = generate_text("Say 'Hello' if you can hear me
208     .", "connection test")
209     return len(test_response) > 0

```

Listing A.9: Modul llm.py (lengkap, ASCII-only).

__init__.py

```

1 """Pai Code package.

```

```

2
3 This package provides a command-line based agentic AI for
   software development.
4 """
5
6 __all__ = [
7     # Public modules
8     "agent",
9     "cli",
10    "config",
11    "llm",
12    "ui",
13    "workspace",
14 ]
15
16 __version__ = "0.1.0"

```

Listing A.10: Modul `__init__.py` (paicode package), ASCII-only.

requirements.txt

```

1 google-generativeai>=0.5.4
2 python-dotenv>=1.0.1
3 rich>=13.7.1
4 Pygments>=2.16.0
5 prompt_toolkit>=3.0.43

```

Listing A.11: File requirements.txt

setup.py

```

1 from setuptools import setup
2
3 if __name__ == "__main__":
4     setup()

```

Listing A.12: File setup.py (konfigurasi setuptools)

setup.cfg

```

1 [metadata]
2 name = pai-code
3 version = 0.1.0
4 description = A command-line based agentic AI for software
   development.
5 long_description = file: README.md
6 long_description_content_type = text/markdown
7 author = gtkrshnaaa
8 author_email = gtkrshnaaa@email.com
9 license = MIT
10 license_files = LICENSE
11
12 [options]
13 packages = find:
14 python_requires = >=3.10
15 install_requires =
16     google-generativeai>=0.5.4
17     python-dotenv>=1.0.1
18     rich>=13.7.1
19     Pygments>=2.16.0
20 include_package_data = True
21
22 [options.packages.find]
23 where = .
24
25 [options.entry_points]
26 console_scripts =
27     pai = paicode.cli:main

```

Listing A.13: File setup.cfg (metadata dan konfigurasi paket)

pyproject.toml

```

1 # pyproject.toml
2
3 [build-system]
4 requires = ["setuptools>=61", "wheel"]
5 build-backend = "setuptools.build_meta"

```

Listing A.14: File pyproject.toml (konfigurasi build system)

makefile

```
1 .PHONY: run export-all install venv-activate setup install-cli
   uninstall-cli
2
3 install:
4     @if [ ! -d .venv ]; then \
5         echo "[install] Creating virtual environment at .venv"; \
6         python3 -m venv .venv; \
7     else \
8         echo "[install] Reusing existing virtual environment at .venv
9         "; \
10    fi
11    . .venv/bin/activate; python -m pip install --upgrade pip
12    . .venv/bin/activate; pip install -r requirements.txt
13    . .venv/bin/activate; pip install -e .
14
15 run:
16     . .venv/bin/activate; python -m paicode.cli auto
17
18 export-all:
19     @mkdir -p z_project_list
20     @echo "Exporting project files to z_project_list/listing.txt..."
21     @rm -f z_project_list/listing.txt
22     @for f in $(find . -type f \
23         -not -path '*/\.*' \
24         -not -path '*/__pycache__/*' \
25         -not -path '*.egg-info/*' \
26         -not -path './z_project_list/*' \
27         -not -name ".gitkeep" \
28         | sort); do \
29         echo "=== $$f ===" >> z_project_list/listing.txt; \
30         cat $$f >> z_project_list/listing.txt; \
31         echo "\n" >> z_project_list/listing.txt; \
32     done
33     @echo "Export complete."
34
35 venv-activate:
36     @echo "To activate the virtual environment, run:"
```

```

36  @echo "  source .venv/bin/activate"
37
38  setup: install install-cli
39  @echo "Pai CLI installed. Ensure $$HOME/.local/bin is in your
    PATH, then run: pai"
40
41  install-cli:
42  @mkdir -p $(HOME)/.local/bin
43  @echo "Installing launcher to $(HOME)/.local/bin/pai"
44  @echo '#!/usr/bin/env bash' > $(HOME)/.local/bin/pai
45  @echo '# Suppress noisy gRPC/absl logs' >> $(HOME)/.local/bin/
    pai
46  @echo 'export GRPC_VERBOSITY="NONE"' >> $(HOME)/.local/bin/pai
47  @echo 'export GRPC_LOG_SEVERITY="ERROR"' >> $(HOME)/.local/bin/
    pai
48  @echo 'export ABSL_LOGGING_MIN_LOG_LEVEL="3"' >> $(HOME)/.local
    /bin/pai
49  @echo 'export GLOG_minloglevel="3"' >> $(HOME)/.local/bin/pai
50  @echo 'export GOOGLE_CLOUD_DISABLE_GRPC="true"' >> $(HOME)/.
    local/bin/pai
51  @echo 'export GRPC_ENABLE_FORK_SUPPORT="false"' >> $(HOME)/.
    local/bin/pai
52  @echo 'SCRIPT_DIR="$$$(cd "$$(dirname "${BASH_SOURCE[0]}")" &&
    pwd)"' >> $(HOME)/.local/bin/pai
53  @echo 'APPDIR="$(shell pwd)"' >> $(HOME)/.local/bin/pai
54  @echo 'VENVDIR="$$APPDIR/.venv"' >> $(HOME)/.local/bin/pai
55  @echo 'PY="$$VENVDIR/bin/python"' >> $(HOME)/.local/bin/pai
56  @echo '# Redirect stderr to suppress remaining warnings' >> $(
    HOME)/.local/bin/pai
57  @echo 'if [ -x "$$VENVDIR/bin/pai" ]; then' >> $(HOME)/.local/
    bin/pai
58  @echo '  exec "$$VENVDIR/bin/pai" "$$@" 2>/dev/null' >> $(HOME)
    /.local/bin/pai
59  @echo 'elif [ -x "$$PY" ]; then' >> $(HOME)/.local/bin/pai
60  @echo '  exec "$$PY" -m paicode.cli "$$@" 2>/dev/null' >> $(
    HOME)/.local/bin/pai
61  @echo 'else' >> $(HOME)/.local/bin/pai
62  @echo '  exec python3 -m paicode.cli "$$@" 2>/dev/null' >> $(
    HOME)/.local/bin/pai
63  @echo 'fi' >> $(HOME)/.local/bin/pai
64  @chmod +x $(HOME)/.local/bin/pai

```

```

65  @# Ensure ~/.local/bin is in PATH (append to ~/.bashrc if
    missing)
66  @if [ -f $(HOME)/.bashrc ]; then \
67      grep -qxF 'export PATH="$$HOME/.local/bin:$$PATH"' $(HOME)/.
        bashrc || printf '\n# Added by pai install-cli\nexport
        PATH="$$HOME/.local/bin:$$PATH"\n' >> $(HOME)/.bashrc; \
68  fi
69  @echo "Ensured PATH includes $$HOME/.local/bin in $$HOME/.
        bashrc. Run: 'source $$HOME/.bashrc' or open a new terminal.
        "
70  @echo "Done. Ensure $(HOME)/.local/bin is in your PATH. Try
        running: pai --help"
71
72  uninstall-cli:
73      @rm -f $(HOME)/.local/bin/pai
74      @# Remove PATH line added by install-cli (safe if absent)
75      @sed -i '/^# Added by pai install-cli$/d' $(HOME)/.bashrc ||
        true
76      @sed -i '/^export PATH="\$HOME\/\.\.local\/bin:\$PATH"/d' $(HOME
        )/.bashrc || true
77      @echo "Launcher removed: $(HOME)/.local/bin/pai"

```

Listing A.15: File makefile (task automation untuk development dan deployment)

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.