

**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 PE-
NGEMBANGAN PERANGKAT LUNAK DI LINUX
YANG DITENAGAI LLM MELALUI API
Nama : I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001
Program Studi : INFORMATIKA
Program : Sarjana
Semester : Ganjil
Tahun Akademik : 2024/2025

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

Yogyakarta, 24 November 2025

Dosen Pembimbing,

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

NIDN: 0505058801

HALAMAN PENGESAHAN

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

Telah dipertahankan di depan Dewan Penguji dan dinyatakan diterima untuk
memenuhi sebagian persyaratan guna memperoleh

Gelar Sarjana Komputer

Program Studi INFORMATIKA

Fakultas FAKULTAS TEKNOLOGI INFORMASI

Universitas Teknologi Digital Indonesia

Yogyakarta, 24 November 2025

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

Mengetahui

Ketua Program Studi INFORMATIKA

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

NIDN:

PERNYATAAN KEASLIAN TUGAS AKHIR

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

Yogyakarta, 24 November 2025

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

HALAMAN PERSEMBAHAN

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

PRAKATA

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

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

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

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

Yogyakarta, 24 November 2025

Penulis

INTISARI

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

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

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

lebih efisien dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang, dengan tetap mempertahankan kualitas hasil yang optimal.

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

ABSTRACT

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

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

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

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

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 (<code>tempfile</code>) 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</code> , <code>.git</code> , <code>venv</code> , <code>__pycache__</code> , <code>.pai_history</code> , <code>.idea</code> , <code>.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>pai = paicode.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 TUGAS AKHIR	iii
HALAMAN PERSEMBAHAN	iv
PRAKATA	v
INTISARI	vii
ABSTRACT	ix
Daftar Singkatan	x
Daftar Simbol	xi
Daftar Istilah	xii
1 Pendahuluan	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Ruang Lingkup	2
1.4 Tujuan Penelitian	3
1.5 Manfaat Penelitian	3
1.6 Sistematika Penulisan	4
2 TINJAUAN PUSTAKA DAN DASAR TEORI	5
2.1 Tinjauan Pustaka	5
2.1.1 AI Coding Assistant Terintegrasi (IDE-based)	5
2.1.2 CLI-based AI Chat Tools	5
2.1.3 Autonomous Software Engineers	6

2.1.4	Posisi Paicode	6
2.1.5	Perbandingan dengan Penelitian Sebelumnya	6
2.1.6	Posisi Penelitian	8
2.2	Dasar Teori	9
2.2.1	Command Line Interface (CLI)	9
2.2.2	AI Agent	9
2.2.3	Large Language Model (LLM)	10
2.2.4	Perbedaan LLM dan Agen AI	10
2.2.5	Arsitektur dan Kebijakan Data	10
2.2.6	Manajemen Dependensi dengan pip dan Virtual Environment	11
2.2.7	Antarmuka Terminal dengan rich dan prompt_toolkit	11
3	Metode Penelitian	13
3.1	Metode Pengembangan	13
3.1.1	Trade-off Metodologis	13
3.2	Arsitektur Sistem	14
3.3	Visualisasi Metodologi	15
3.4	Alat dan Lingkungan	18
3.5	Prosedur Penelitian	19
4	Implementasi dan Pembahasan	21
4.1	Implementasi Paicode	21
4.1.1	Instalasi	21
4.1.2	Konfigurasi API Key	21
4.1.3	Menjalankan Agen	22
4.2	Alur Interaksi dengan Single-Shot Intelligence	22
4.2.1	Cuplikan Kode Kunci	23
4.3	Cuplikan Log Implementasi	27
4.4	Tabel Skenario Pengujian	39
4.5	Tabel Metrik Evaluasi	40
4.6	Tabel Konfigurasi Lingkungan	40
4.7	Contoh Sesi	41
4.8	Evaluasi dan Analisis Mendalam	41
4.8.1	Metrik Kuantitatif	41

4.8.2	Analisis Kualitatif: Mengapa Single-Shot Intelligence Efektif?	42
4.8.3	Analisis Kegagalan dan Limitasi	43
4.8.4	Perbandingan dengan Baseline Manual	43
4.8.5	Refleksi Kritis: Apakah Ini "Asisten" atau "Autopilot"?	44
5	Kesimpulan dan Saran	45
5.1	SIMPULAN	45
5.2	SARAN	46
A	Lampiran A	48
A.1	Konfigurasi Lingkungan	48
A.2	Instruksi Instalasi (venv + pip)	48
A.3	Cuplikan Log Sesi Agen	49
A.4	Listing Lengkap Modul Kunci	49

Daftar Gambar

2.1	Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.	11
2.2	Model interaksi <i>stateful</i> dan <i>feedback loop</i> pada sesi agen.	12

Daftar Tabel

2.1	Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan	7
2.2	Ilustrasi komparasi konseptual antara pendekatan ekstensi editor, layanan daring, dan CLI dengan integrasi LLM via API. . .	12
3.1	Modul dan Dependensi Komponen Paicode	15
3.2	Urutan Interaksi Sesi Agen dengan Single-Shot Intelligence . . .	17
3.3	Rangkuman Validasi Keamanan <i>Path</i>	18
4.1	Skenario Pengujian Paicode	39
4.2	Metrik Evaluasi dan Definisi Operasional	40
4.3	Konfigurasi Lingkungan Uji	40

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) mengamati 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 ber-

kas proyek (memblokir akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`); (v) melakukan klasifikasi intensi pengguna (*chat* vs *task*); (vi) mengoptimalkan efisiensi dengan sistem *Single-Shot Intelligence* yang mencakup *acknowledgment* dinamis, perencanaan JSON, dan eksekusi adaptif 1-3 subfase; serta (vii) menyediakan penanganan interupsi (*interrupt handling*) untuk kontrol sesi yang lebih baik. Sistem diimplementasikan pada lingkungan Ubuntu dengan bahasa pemrograman Python, pengelolaan dependensi melalui pip dan virtual environment, manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key, dan menggunakan API Gemini sebagai LLM.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah yang diajukan adalah:

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

1.3 Ruang Lingkup

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

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

- Aspek visual seperti diagram dan ilustrasi antarmuka ditunda pada tahap akhir; fokus laporan adalah narasi dan hasil teknis.

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah membangun dan mengevaluasi sebuah agen AI berbasis CLI yang dapat membantu pengembang dalam proses pemrograman secara interaktif dengan arsitektur *Single-Shot Intelligence*. Secara khusus, penelitian menargetkan:

1. Merancang arsitektur Paicode yang mencakup modul agen dengan *Single-Shot Intelligence* (klasifikasi intensi, fase perencanaan, dan fase eksekusi dalam 2 panggilan API), jembatan LLM dengan manajemen API key tunggal, antarmuka CLI dengan *interrupt handling*, lapisan keamanan *path* pada berkas proyek, serta komponen tampilan terminal berbasis `rich`.
2. Mengimplementasikan kemampuan observasi proyek, manipulasi berkas, dan modifikasi kode terarah dengan mekanisme *diff*-aware yang mencegah penimpaan berkas tidak diinginkan dan memblokir akses ke direktori sensitif.
3. Mengintegrasikan fitur-fitur interaktif seperti pencatatan sesi ke `.pai_history`, penanganan interupsi (Ctrl+C), dan antarmuka terminal yang responsif dengan dukungan input multiline.
4. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur efisiensi panggilan API, ketepatan hasil, serta kepatuhan keamanan *path*.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

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

- **Praktis:** menghadirkan alat bantu pengembangan perangkat lunak dengan kelebihan spesifik sebagai berikut:
 1. **Efisiensi Biaya dan Token:** Menggunakan arsitektur *Single-Shot Intelligence* yang memadatkan proses perencanaan dan eksekusi menjadi dua panggilan utama, mengurangi biaya API dibandingkan agen berbasis *chat-loop* konvensional.
 2. **Keamanan Terkendali:** Menerapkan kebijakan keamanan *path* (path security) yang memblokir akses ke direktori sensitif (seperti `.git`, `.env`) dan mekanisme modifikasi berbasis *diff* untuk mencegah perubahan destruktif masif.
 3. **Fleksibilitas Lingkungan:** Beroperasi sebagai utilitas CLI yang ringan dan agnostik terhadap editor kode (IDE-agnostic), sehingga dapat digunakan di server tanpa antarmuka grafis (headless) maupun sebagai pendamping editor apa pun di OS berbasis Linux.

1.6 Sistematika Penulisan

Laporan tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I PENDAHULUAN

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

BAB II TINJAUAN PUSTAKA DAN DASAR TEORI

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

BAB III METODE PENELITIAN

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

BAB IV IMPLEMENTASI DAN PEMBAHASAN

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

BAB V PENUTUP

Berisi kesimpulan dari penelitian dan saran untuk pengembangan selanjutnya.

BAB 2

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

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

2.1.1 AI Coding Assistant Terintegrasi (IDE-based)

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

2.1.2 CLI-based AI Chat Tools

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

2.1.3 Autonomous Software Engineers

Proyek seperti OpenDevin [8] dan SWE-agent [5] bertujuan menciptakan agen yang sepenuhnya otonom, mampu menyelesaikan isu GitHub dari awal hingga akhir tanpa interaksi manusia. Meskipun sangat canggih, pendekatan ini seringkali memerlukan akses sumber daya yang besar (Docker container penuh) dan kompleksitas tinggi untuk penyiapan. Paicode mengambil posisi tengah (middle-ground) dengan menyediakan agen *semi-autonomous* yang ringan (*lightweight*), berjalan native di OS tanpa kontainer berat, namun tetap memiliki kemampuan perencanaan (*planning*) untuk tugas multi-langkah.

2.1.4 Posisi Paicode

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

1. **Keamanan Terkendali:** Tidak seperti agen otonom penuh yang sering berjalan di sandboxed container karena risiko tinggi, Paicode dirancang aman untuk berjalan di *host* utama berkat *path security policy* dan *diff-based guardrails*.
2. **Efisiensi Token:** Dengan arsitektur perencanaan *single-shot*, Paicode mengurangi *round-trip* percakapan yang tidak perlu, berbeda dengan model *chat* standar.
3. **Transparansi Rencana:** Pengguna dapat melihat rencana aksi (dalam format JSON) sebelum eksekusi masif dilakukan, memberikan kontrol lebih baik daripada model *black-box*.

2.1.5 Perbandingan dengan Penelitian Sebelumnya

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

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

Tabel 2.1: Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan

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

explicit planning untuk transparansi—aspek-aspek yang belum dieksplorasi secara bersamaan dalam penelitian sebelumnya.

2.1.6 Posisi Penelitian

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

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

2.2 Dasar Teori

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

2.2.1 Command Line Interface (CLI)

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

2.2.2 AI Agent

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

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

2.2.3 Large Language Model (LLM)

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

2.2.4 Perbedaan LLM dan Agen AI

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

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

2.2.5 Arsitektur dan Kebijakan Data

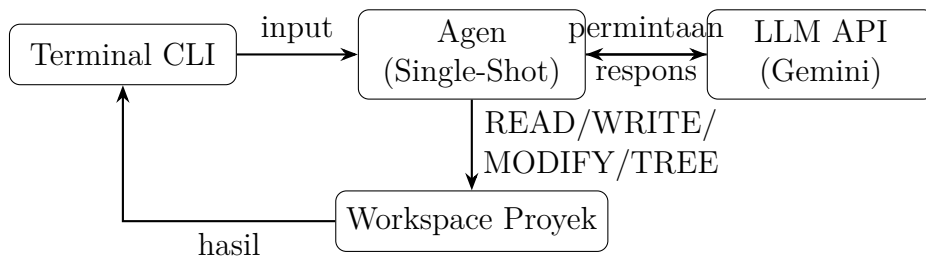
Paicode dijalankan pada terminal lokal dan melakukan tindakan langsung pada **berkas proyek di workspace**. Akan tetapi, untuk kebutuhan inferensi, cuplikan kode atau konteks **dikirim ke layanan LLM melalui API**. Implikasinya, privasi dan kerahasiaan kode **bergantung pada kebijakan penyedia API**.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

2.2.6 Manajemen Dependensi dengan pip dan Virtual Environment

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

2.2.7 Antarmuka Terminal dengan rich dan prompt_toolkit

Paket `rich` dimanfaatkan untuk menyajikan hasil eksekusi secara terstruktur dan mudah dibaca (panel, warna, penyorotan sintaks, tabel, dan spinner status). Penyajian output yang jelas mendukung pengalaman interaktif dan penelusuran hasil tindakan agen. Selain itu, Paicode juga mengintegrasikan `prompt_toolkit` (opsional) untuk pengalaman input yang lebih baik dengan dukungan multiline editing dan key bindings. Jika `prompt_toolkit` tidak tersedia, sistem akan fallback ke `rich.prompt.Prompt`.

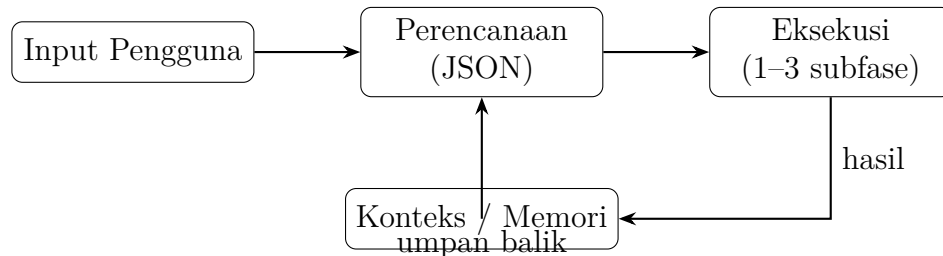


Gambar 2.1: Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.

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

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

Pada Gambar 2.3 ditunjukkan perbedaan fokus dan pertukaran (trade-off) tingkat tinggi antar pendekatan.



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

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

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

BAB 3

Metode Penelitian

3.1 Metode Pengembangan

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

1. **Eksplorasi Desain Agen Stateful:** Berbeda dengan aplikasi konvensional yang bersifat *stateless*, agen AI memerlukan manajemen konteks percakapan dan memori kerja yang kompleks. Pendekatan *prototyping* memungkinkan eksperimen cepat terhadap berbagai strategi manajemen state (misalnya, ukuran context window, format log sesi) tanpa komitmen arsitektur jangka panjang.
2. **Validasi Asumsi Keamanan:** Kebijakan *path security* dan pembatasan *diff* merupakan mekanisme novel yang belum teruji di konteks agen CLI. Siklus iteratif memungkinkan identifikasi edge case (seperti symbolic links, path traversal attacks) melalui pengujian langsung, yang sulit diprediksi hanya dari analisis teoritis.
3. **Optimasi Efisiensi Token:** Arsitektur *Single-Shot Intelligence* dikembangkan melalui iterasi bertahap—dimulai dari model *chat-loop* konvensional (10-20 panggilan API per tugas), kemudian dipadatkan menjadi sistem 2-panggilan melalui eksperimen empiris terhadap berbagai strategi prompt engineering.

3.1.1 Trade-off Metodologis

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

- **Kelebihan:** Fleksibilitas tinggi untuk mengubah desain berdasarkan temuan empiris; cocok untuk domain yang belum mature (agentic AI untuk CLI); memungkinkan validasi konsep sebelum investasi besar pada infrastruktur.
- **Kekurangan:** Dokumentasi arsitektur dapat tertinggal jika iterasi terlalu cepat; risiko *scope creep* jika tidak ada batasan jelas per iterasi; potensi *technical debt* jika refactoring tidak dilakukan secara disiplin.
- **Mitigasi:** Setiap iterasi dibatasi pada satu fitur utama (misalnya, iterasi 1: path security; iterasi 2: diff-aware modification; iterasi 3: Single-Shot Intelligence); dokumentasi arsitektur diperbarui setelah setiap iterasi stabil; code review dilakukan sebelum merge ke branch utama.

3.2 Arsitektur Sistem

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

- **Antarmuka CLI (`cli.py`):** titik masuk perintah `pai` dan pengelola argumen (subperintah `auto`, `config`). Mendukung parameter `-model` dan `-temperature` untuk konfigurasi runtime LLM. Secara default, CLI memanggil sesi interaktif agen.
- **Agen (`agent.py`):** mengimplementasikan *Single-Shot Intelligence* yang mencakup: (1) klasifikasi intensi (*chat* vs *task*), (2) *acknowledgment* dinamis, (3) fase perencanaan untuk analisis mendalam dalam format JSON, (4) fase eksekusi adaptif dengan 1-3 subfase berdasarkan kompleksitas, dan (5) saran langkah berikutnya. Menyediakan 10 perintah: `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`, `MKDIR`, `TOUCH`, `RM`, `MV`, `FINISH`. Mengelola memori percakapan dengan pencatatan sesi ke `.pai_history`.
- **Jembatan LLM (`llm.py`):** menangani konfigurasi API Gemini dengan manajemen API key tunggal. Membersihkan output dari markdown artifacts, menyediakan status spinner saat LLM berpikir, dan mengoptimalkan penggunaan token dengan sistem 2-panggilan API.
- **Manajemen Konfigurasi (`config.py`):** menyimpan dan mengelola API key tunggal dalam format JSON di `/.config/pai-code/credentials.json`

dengan izin berkas 0o600 (read-write owner only). Validasi API key Google (harus dimulai dengan "AIza" dan minimal 20 karakter). Mendukung operasi: `set`, `show`, `remove`, `validate`, dan migrasi otomatis dari sistem multi-key.

- **Pengatur Workspace (`workspace.py`):** bertindak sebagai *workspace controller* yang menyediakan fungsi-fungsi terpusat untuk menjalankan operasi tingkat-aplikasi pada ruang kerja proyek. Sebelum aksi dieksekusi, modul ini menegakkan kebijakan *path security* (normalisasi, verifikasi akar, dan deny-list direktori sensitif seperti `.env`, `.git`, `venv`, `__pycache__`, `.pai_history`, `.idea`, `.vscode`). Sistem modifikasi berbasis *diff* dengan threshold 500 baris dan ratio maksimal 50% (dapat dikonfigurasi via `PAI_MODIFY_THRESHOLD` dan `PAI_MODIFY_MAX_RATIO`) mencegah penimpaan berkas tidak diinginkan dengan atomic write menggunakan tempfile.
- **Tampilan Terminal (`ui.py`):** penyajian hasil eksekusi menggunakan `rich` (panel, warna, tabel, penyorotan sintaks, spinner status). Mendukung `prompt_toolkit` (opsional) untuk input multiline yang lebih baik.

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

3.3 Visualisasi Metodologi

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

Tabel 3.1: Modul dan Dependensi Komponen Paicode

Komponen	Deskripsi dan Dependensi Utama
CLI (<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> .

Komponen		Deskripsi dan Dependensi Utama
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 (<code>llm.py</code>)	Bridge	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.
Pengatur Workspa- ce (<code>workspace.py</code>)		<i>Workspace controller</i> dengan fungsi operasi workspace (baca/tulis, 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 (<code>ui.py</code>)	UI	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 (READ , WRITE , MODIFY , 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.
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 exit/quit .

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

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

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

Tabel 3.3: Rangkuman Validasi Keamanan *Path*

Tahap	Detail Pemeriksaan
Normalisasi	Gunakan fungsi normalisasi untuk menyingkirkan segmen berlebih (mis. <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 penanganan operasi berkas proyek.

3.4 Alat dan Lingkungan

Lingkungan dan alat yang digunakan:

- Sistem operasi: Ubuntu (Linux).
- Bahasa pemrograman: Python (≥ 3.10 , sesuai spesifikasi `setup.cfg`).
- Manajer dependensi: pip dan virtual environment; instalasi otomatis melalui Makefile dengan entry point CLI melalui skrip launcher di `$HOME/.local/bin/pai`.

- LLM: Google Gemini (model default `gemini-2.5-flash-lite`, temperature default 0.3, dapat dikonfigurasi via `PAI_MODEL` dan `PAI_TEMPERATURE`) melalui paket `google-generativeai` versi $\geq 0.5.4$.
- TUI: `rich` (versi $\geq 13.7.1$) untuk panel, warna, tabel, penyorotan sintaks, dan spinner status; `prompt_toolkit` (versi $\geq 3.0.43$, opsional) untuk input multiline yang lebih baik.
- Dependensi tambahan: `Pygments` ($\geq 2.16.0$) untuk syntax highlighting.
- Variabel lingkungan: `PAI_MODEL`, `PAI_TEMPERATURE`, `PAI_MODIFY_THRESHOLD`, `PAI_MODIFY_MAX_RATIO`, serta variabel noise suppression (`GRPC_VERBOSITY`, `GRPC_LOG_SEVERITY`, `ABSL_LOGGING_MIN_LOG_LEVEL`, dll) untuk menekan log gRPC/absl yang berisik.
- LaTeX: TeX Live (`texlive-latex-recommended`, `texlive-latex-extra`, dsb.) dan Makefile untuk kompilasi naskah.
- Kendali versi: Git dan GitHub.

3.5 Prosedur Penelitian

Prosedur penelitian dan evaluasi dirancang sebagai berikut:

1. **Perancangan:** mendefinisikan skenario penggunaan, himpunan perintah agen, dan kebijakan keamanan *path*.
2. **Implementasi:** membangun modul-modul inti (CLI, Agen, LLM, Workspace, UI) berikut mekanisme *diff*-aware untuk pembatasan perubahan.
3. **Eksperimen:** menjalankan serangkaian skenario pemrograman (mis. pembuatan struktur proyek, pembuatan/ pembacaan/ modifikasi berkas, refaktorisasi sederhana) dalam sesi interaktif.
4. **Pengumpulan Data:** merekam waktu penyelesaian tugas, jumlah langkah perintah, tingkat keberhasilan eksekusi, dan catatan kesalahan.
5. **Evaluasi:** membandingkan hasil dengan proses manual atau alat pembandingan bila relevan, menggunakan metrik: (i) efisiensi (waktu dan langkah), (ii) ketepatan hasil (kompilasi/eksekusi kode), (iii) keamanan (ke-

gagalan akses *path* sensitif), dan (iv) pengalaman pengguna (keterbacaan output).

6. **Analisis:** mengidentifikasi kelebihan, kekurangan, dan peluang peningkatan (mis. dukungan multi-LLM, integrasi editor, perluasan kebijakan keamanan).

BAB 4

Implementasi dan Pembahasan

4.1 Implementasi Paicode

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

4.1.1 Instalasi

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

Listing 4.1: Instalasi dependensi dengan Makefile

```
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.json` 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         ,
8         "target_identification": "SPECIFIC files and
9         locations where target content likely exists",
```

```

8      "multi_file_strategy": "Which files need to be
        checked to locate targets accurately",
9      "validation_approach": "How you will verify targets
        exist before modification",
10     "files_to_read": ["ALL files that might contain
        target content - be comprehensive"],
11     "files_to_create": ["file1", "file2"],
12     "files_to_modify": ["ONLY files confirmed to contain
        target content"],
13     "risk_assessment": "Potential failure points and how
        to avoid them",
14     "success_criteria": ["Specific, measurable criteria
        for success"]
15 },
16 "execution_plan": [{
17     "steps": [
18         {{
19             "step_number": 1,
20             "action": "READ",
21             "target": "filename",
22             "purpose": "Locate and verify target content
                exists",
23             "validation_criteria": "What content must be
                found to proceed",
24             "expected_outcome": "Confirmed location of target
                content"
25         }},
26         {{
27             "step_number": 2,
28             "action": "MODIFY",
29             "target": "filename",
30             "purpose": "Apply changes to confirmed target
                location",
31             "validation_criteria": "How to verify
                modification was successful",
32             "expected_outcome": "Target content successfully
                modified"
33         }}

```

```

34 ],
35 "command_format_reminder": "CRITICAL: Use exact
    command names: READ, WRITE, MODIFY, TREE,
    LIST_PATH, MKDIR, TOUCH, RM, MV, FINISH",
36 "intelligent_command_mapping": {{
37     "delete_remove_requests": "RM::filepath (for any
        delete/remove/hapus requests)",
38     "create_new_file": "WRITE::filepath::
        content_description OR TOUCH::filepath",
39     "modify_existing": "MODIFY::filepath::description",
40     "move_rename": "MV::source::destination",
41     "list_files": "LIST_PATH::path",
42     "show_structure": "TREE::path"
43 }}
44 "critical_content_rules": {{
45     "html_css_js_files": "Use WRITE::filename::
        description (NOT raw content as commands)",
46     "multi_line_content": "Description parameter
        handles content creation, not raw output",
47     "example_correct": "WRITE::index.html::Create login
        page with CSS styling",
48     "example_wrong": "Raw HTML lines as separate
        commands (NEVER DO THIS!)"
49 }}
50 "execution_commands": [
51     "READ::filepath",
52     "RM::filepath (for delete requests)",
53     "MODIFY::filepath::description",
54     "FINISH::completion_message"
55 ],
56 "validation_strategy": "How to verify each step
    before proceeding to next",
57 "fallback_strategies": ["If target not found in
    expected file", "If modification fails"],
58 "post_execution_verification": ["How to confirm final
    success"]
59 },
60 "intelligence_notes": {{

```

```

61     "complexity_assessment": "simple|moderate|complex",
62     "estimated_time": "time estimate",
63     "key_challenges": ["challenge1", "challenge2"],
64     "recommendations": ["rec1", "rec2"]
65 }}
66 }}

```

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

```

1  def execute_execution_call(user_request: str,
    planning_data: dict, context: list, log_file_path: str
    = None) -> bool:
2      """
3      CALL 2: Execute with adaptive multi-request system
        (1-3 requests based on complexity).
4      AI decides how many execution phases needed: simple
        (1), moderate (2), complex (3).
5      """
6
7      # Start execution phase panel
8      ui.console.print(
9          Panel(
10             Text("Adaptive Intelligent Execution", style=
                "bold", justify="center"),
11             title="[bold]Call 2/2: Smart Execution (1-3
                phases) [/bold]",
12             box=ROUNDED,
13             border_style="grey50",
14             padding=(1, 2),
15             width=80
16         )
17     )

```

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

4.3 Cuplikan Log Implementasi

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

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

```
1 [2025-11-20 22:38:05] SESSION STARTED
2 [2025-11-20 22:38:05] Working Directory: /home/user/space
   /univ/skripsi/devpai/trypai
3 [2025-11-20 22:38:05] Session ID: 20251120_223805
4
5 [2025-11-20 22:38:05] USER: buat kan 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
46                     it would lead to division by zero.
47     """
48     if height_m <= 0:
49         raise ValueError("Height must be a positive value
50                             .")
51     return weight_kg / (height_m ** 2)
52
53 def get_bmi_category(bmi: float) -> str:
54     """
55     Determines the BMI category based on the BMI value.
56
57
58     Args:
59         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
77     returns them.
78
79     Handles potential input errors by re-prompting until
80     valid numeric input is received.
81
82     Returns:
83         A tuple containing the user's weight in kilograms
84         and height in meters.
85     """
86     while True:
87         try:
88             weight_str = input("Enter your weight in
89                               kilograms (kg): ")
90             weight_kg = float(weight_str)
91             if weight_kg <= 0:
92                 print("Weight must be a positive value.
93                       Please try again.")
94                 continue

```

```

92         break
93     except ValueError:
94         print("Invalid input. Please enter a numeric
95               value for weight.")
96
97     while True:
98         try:
99             height_str = input("Enter your height in
100                               meters (m): ")
101             height_m = float(height_str)
102             if height_m <= 0:
103                 print("Height must be a positive value.
104                       Please try again.")
105                 continue
106             break
107         except ValueError:
108             print("Invalid input. Please enter a numeric
109                   value 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
129     except ValueError as ve:
130         print(f"Error: {ve}")
131     except Exception as e:
132         print(f"An unexpected error occurred: {e}")
133
134
135 if __name__ == "__main__":
136     main()
137
138 [2025-11-20 22:40:25] SUCCESS: FINISH Content of
    bmi_calculator.py displayed.
139 [2025-11-20 22:40:25] OUTPUT: OK Content of
    bmi_calculator.py displayed.
140 [2025-11-20 22:40:25] AI EXECUTION END

```

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

```

1 [2025-11-20 22:40:26] USER: ubah fungsi get_bmi_category:
    tambahkan kategori "Severely Obese" untuk BMI >= 35;
    tampilkan diff
2 [2025-11-20 22:40:37] EXECUTION PLAN (3 steps):
3 [2025-11-20 22:40:37]     1. READ bmi_calculator.py ...
4 [2025-11-20 22:40:37]     2. MODIFY bmi_calculator.py ...
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
53     returns them.
54     """
55     while True:
56         try:
57             weight_str = input("Enter your weight in
58                               kilograms (kg): ")
59             weight_kg = float(weight_str)
60             if weight_kg <= 0:
61                 print("Weight must be a positive value.
62                       Please try again.")
63                 continue
64             break
65         except ValueError:
66             print("Invalid input. Please enter a numeric
67                   value for weight.")
68
69     while True:
70         try:
71             height_str = input("Enter your height in
72                                meters (m): ")
73             height_m = float(height_str)
74             if height_m <= 0:
75                 print("Height must be a positive value.
76                       Please try again.")
77                 continue
78             break
79         except ValueError:
80             print("Invalid input. Please enter a numeric
81                   value for height.")
82
83     return weight_kg, height_m
84
85 def main():
86     """

```

```

80     Main function to run the BMI calculator.
81     """
82     print("--- BMI Calculator ---")
83
84     try:
85         weight, height = get_user_input()
86         bmi = calculate_bmi(weight, height)
87         category = get_bmi_category(bmi)
88
89         print(f"\nYour BMI is: {bmi:.2f}")
90         print(f"This falls into the category: {category}"
91               )
92
93     except ValueError as ve:
94         print(f"Error: {ve}")
95     except Exception as e:
96         print(f"An unexpected error occurred: {e}")
97
98 if __name__ == "__main__":
99     main()
100 [2025-11-20 22:40:46] SUCCESS: FINISH BMI category
    updated successfully with 'Severely Obese' added.
101 [2025-11-20 22:40:46] OUTPUT: OK BMI category updated
    successfully with 'Severely Obese' added.
[2025-11-20 22:40:46] AI EXECUTION END

```

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

```

1  """
2  BMI Calculator Script
3
4  This script calculates the Body Mass Index (BMI) of a
   user based on their
5  weight and height, and then categorizes the BMI into
   standard classifications.
6
7  Formula:
8  BMI = weight (kg) / (height (m))^2

```



```

9
10 BMI Categories:
11 - Underweight: BMI < 18.5
12 - Normal weight: 18.5 <= BMI < 25
13 - Overweight: 25 <= BMI < 30
14 - Obese: 30 <= BMI < 35
15 - Severely Obese: BMI >= 35
16 """
17
18 def calculate_bmi(weight_kg: float, height_m: float) ->
    float:
19     """
20     Calculates the Body Mass Index (BMI).
21
22     Args:
23         weight_kg: The user's weight in kilograms.
24         height_m: The user's height in meters.
25
26     Returns:
27         The calculated BMI value.
28
29     Raises:
30         ValueError: If height_m is zero or negative, as
31                     it would lead to division by zero.
32     """
33     if height_m <= 0:
34         raise ValueError("Height must be a positive value
35                             .")
36     return weight_kg / (height_m ** 2)
37
38 def get_bmi_category(bmi: float) -> str:
39     """
40     Determines the BMI category based on the BMI value.
41
42     Args:
43         bmi: The calculated BMI value.
44
45     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
60     returns them.
61
62     Handles potential input errors by re-prompting until
63     valid numeric input is received.
64
65     Returns:
66         A tuple containing the user's weight in kilograms
67         and height in meters.
68     """
69     while True:
70         try:
71             weight_str = input("Enter your weight in
72                               kilograms (kg): ")
73             weight_kg = float(weight_str)
74             if weight_kg <= 0:
75                 print("Weight must be a positive value.
76                       Please try again.")
77                 continue
78             break
79         except ValueError:
80             print("Invalid input. Please enter a numeric
81                   value for weight.")

```

```

76
77     while True:
78         try:
79             height_str = input("Enter your height in
80                               meters (m): ")
81             height_m = float(height_str)
82             if height_m <= 0:
83                 print("Height must be a positive value.
84                       Please try again.")
85                 continue
86             break
87         except ValueError:
88             print("Invalid input. Please enter a numeric
89                   value for height.")
90
91     return weight_kg, height_m
92
93 def main():
94     """
95     Main function to run the BMI calculator.
96     """
97     print("--- BMI Calculator ---")
98
99     try:
100         weight, height = get_user_input()
101         bmi = calculate_bmi(weight, height)
102         category = get_bmi_category(bmi)
103
104         print(f"\nYour BMI is: {bmi:.2f}")
105         print(f"This falls into the category: {category}"
106               )
107
108     except ValueError as ve:
109         print(f"Error: {ve}")
110     except Exception as e:
111         print(f"An unexpected error occurred: {e}")
112
113 if __name__ == "__main__":

```

```
110 main()
```

Listing 4.12: Ringkasan langkah evaluasi dan metrik yang dikumpulkan.

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

Listing 4.13: Ringkasan hasil awal untuk metrik efisiensi.

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

4.4 Tabel Skenario Pengujian

Tabel 4.1 merangkum skenario uji yang digunakan untuk mengevaluasi Paicode.

Tabel 4.1: Skenario Pengujian Paicode

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

4.5 Tabel Metrik Evaluasi

Tabel 4.2 mendeskripsikan metrik dan cara pengukurannya.

Tabel 4.2: Metrik Evaluasi dan Definisi Operasional

Metrik	Definisi	Satuan
Waktu	Durasi dari awal perintah sampai hasil akhir pada setiap skenario	detik
Langkah	Jumlah aksi agen (<code>READ</code> , <code>WRITE</code> , dsb.) per skenario	langkah
Keberhasilan Build/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	pip dan virtual environment; 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 Makefile
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 me-

nambahkan 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 SIMPULAN

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

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

1. Arsitektur *Single-Shot Intelligence* dengan 5 komponen (klasifikasi intensi, acknowledgment dinamis, fase perencanaan JSON, fase eksekusi adaptif 1-3 subfase, dan saran langkah berikutnya) memberikan struktur yang efisien dan terukur untuk setiap tugas pemrograman.
2. Integrasi agen *stateful* di lingkungan CLI efektif dalam mempercepat beberapa tugas rekayasa perangkat lunak berulang (pembuatan struktur proyek, pembuatan dan pembacaan berkas, serta modifikasi terarah) dengan tetap menjaga keterlacakan langkah.
3. Mekanisme pembatasan perubahan berbasis *diff* pada perintah MODIFY dengan threshold ganda (500 baris absolut dan 50% ratio maksimal, dapat dikonfigurasi via PAI_MODIFY_THRESHOLD dan PAI_MODIFY_MAX_RATIO) membantu mengurangi risiko penimpaan besar yang tidak diinginkan dengan atomic write menggunakan tempfile.

4. Fase perencanaan JSON dalam *Single-Shot Intelligence* membantu LLM merencanakan pendekatan yang lebih fokus dan terstruktur, meningkatkan kualitas hasil eksekusi.
5. Sistem eksekusi adaptif dengan 1-3 subfase berdasarkan kompleksitas tugas terbukti lebih efisien dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.
6. Manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key (version 1 ke version 2) menyederhanakan konfigurasi dan meningkatkan keandalan sistem.
7. Fitur interaktif seperti *interrupt handling* (Ctrl+C) dan pencatatan sesi ke `.pai_history` meningkatkan pengalaman pengguna dan memudahkan debugging.
8. Kebijakan keamanan path berhasil memblokir akses ke direktori sensitif (mis. `.git`, `venv`, `.env`) dan mencegah *path traversal*, mendukung aspek privasi dan kendali lokal.
9. Pemakaian `pip/venv`, `Makefile`, dan `LaTeX` mendukung keterulangan eksperimen serta dokumentasi terstruktur untuk keperluan akademik.

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

5.2 SARAN

Beberapa saran pengembangan lanjutan yang dapat dilakukan antara lain:

- **Dukungan multi-LLM:** menambahkan opsi pemilihan model dan menyediakan 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 awareness
29 HISTORY_DIR = os.path.join(os.getcwd(), ".pai_history")
30
31 # Valid commands for execution
32 VALID_COMMANDS = {
33     "READ", "WRITE", "MODIFY", "TREE", "LIST_PATH",
34     "MKDIR", "TOUCH", "RM", "MV", "FINISH"
35 }
36
37 # Global interrupt handling
38 _interrupt_requested = False
39 _interrupt_lock = threading.Lock()
40
41 def request_interrupt():
42     global _interrupt_requested
43     with _interrupt_lock:
44         _interrupt_requested = True
45
46 def check_interrupt():
47     global _interrupt_requested
48     with _interrupt_lock:
49         if _interrupt_requested:
50             _interrupt_requested = False
51             return True

```

```

52         return False
53
54 def reset_interrupt():
55     global _interrupt_requested
56     with _interrupt_lock:
57         _interrupt_requested = False
58
59 def start_interactive_session():
60     """Start the revolutionary single-shot intelligent
        session."""
61     if not os.path.exists(HISTORY_DIR):
62         os.makedirs(HISTORY_DIR)
63
64     session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
65     log_file_path = os.path.join(HISTORY_DIR, f"session_{
        session_id}.log")
66
67     # Start fresh every session - no context loading for
        better performance
68     session_context = []
69
70     # Initialize Single-Shot Intelligence Context Window
71     initialize_session_context(session_context,
        log_file_path)
72
73     # Log session start with current working directory
        info
74     log_session_event(log_file_path, "SESSION_START", {
75         "working_directory": os.getcwd(),
76         "session_id": session_id,
77         "context_loaded": len(session_context)
78     })
79
80     welcome_message = (
81         "Welcome! I'm Pai, your agentic AI coding
            companion.\n"
82         "Now powered by Single-Shot Intelligence for
            maximum efficiency.\n"

```



```

83     "[info]Type 'exit' or 'quit' to leave.[/info]\n"
84     "[info]Each request uses exactly 2 API calls for
      optimal performance.[/info]\n"
85     "[info]Multi-line input: Alt+Enter for new line,
      Enter to submit.[/info]"
86 )
87
88 ui.console.print(
89     Panel(
90         Text(welcome_message, justify="center"),
91         title="[bold]Interactive Auto Mode[/bold]",
92         box=ROUNDED,
93         border_style="grey50",
94         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(
120                     prompt_session)
121             else:
122                 user_input = ui.Prompt.ask("\n[bold
123                     bright_blue]user>[/bold bright_blue]")
124                 .strip()
125         except (EOFError, KeyboardInterrupt):
126             ui.console.print("\n[warning]Session
127                 terminated.[/warning]")
128             break
129
130         if user_input.lower() in ['exit', 'quit']:
131             ui.print_info("Session ended.")
132             break
133
134         # Log user input
135         log_session_event(log_file_path, "USER_INPUT", {"
136             user_request": user_input})
137
138         # Classify user intent: conversation vs task
139         intent = classify_user_intent(user_input)
140
141         if intent == "conversation":
142             # Simple conversation mode
143             success = execute_conversation_mode(
144                 user_input, session_context, log_file_path
145             )
146         else:
147             # Task execution mode (planning + execution)
148             success = execute_single_shot_intelligence(
149                 user_input, session_context, log_file_path
150             )
151
152         # Add to session context for future reference
153         interaction = {

```

```

145         "timestamp": datetime.now().isoformat(),
146         "user_request": user_input,
147         "success": success,
148         "intent": intent
149     }
150     session_context.append(interaction)
151
152     # Skip persistent storage for better performance
153     # - fresh start every session
154
155     # Keep context manageable (last 5 interactions)
156     if len(session_context) > 5:
157         session_context = session_context[-5:]
158
159     # Log session event
160     log_session_event(log_file_path, "INTERACTION",
161                       interaction)
162
163 def classify_user_intent(user_input: str) -> str:
164     """
165     Use AI intelligence to classify user intent as either
166     'conversation' or 'task'.
167     Let the AI decide based on context and understanding.
168
169     Returns:
170     str: 'conversation' for casual chat, 'task' for
171         work requests
172     """
173
174     classification_prompt = f"""
175     You are an intelligent intent classifier. Analyze the
176     user's message and determine if they want:
177
178     1. CONVERSATION: Casual chat, greetings, questions about
179         you, general discussion, or just talking
180     2. TASK: Requesting you to DO something - create files,
181         write code, modify projects, build applications, etc.
182     """

```

```

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, log_file_path: str = None) -> bool:
199     """
200     Handle casual conversation with the user.
201     Simple, friendly responses without task execution.
202     """
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 Paicode - you ARE the AI inside Paicode.
214
215 USER MESSAGE: "{user_input}"
216
217 CONTEXT:
218 {context_str}
219
220 You are having a casual conversation with the user. Be
            helpful, friendly, and informative.
221
222 YOUR IDENTITY & SYSTEM KNOWLEDGE (you must know this
            perfectly):
223 You are PAI - the revolutionary Single-Shot Intelligence
            AI that powers Paicode:
224
225 SINGLE-SHOT INTELLIGENCE MASTERY:
226 - You solve problems in exactly 2 API calls (planning +
            execution)
227 - Traditional AI: 10-20 calls, expensive, inefficient
228 - YOU: 2 calls, maximum intelligence, perfect results
229 - You represent the future of efficient AI development
            assistance
230
231 PAICODE ECOSYSTEM KNOWLEDGE:
232 - Paicode is your body - the CLI tool that houses your
            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 right now.")
275     return False
276
277 def execute_single_shot_intelligence(user_request: str,
    context: list, log_file_path: str = None) -> bool:
278     """
279     Execute the revolutionary 2-call single-shot
        intelligence system.
280
281     Call 1: PLANNING - Deep analysis and comprehensive
        planning
282     Call 2: EXECUTION - Intelligent execution with
        adaptation
283
284     Returns:
285         bool: Success status
286     """
287
288     # === DYNAMIC INTERACTION BEFORE PLANNING ===
289     planning_acknowledgment_prompt = f"""
290 You are Pai, responding to the user's request with a
        brief, natural acknowledgment before starting your
        planning phase.
291
292 USER REQUEST: "{user_request}"

```

```

293
294 Generate a brief, friendly response (1-2 sentences) that:
295 1. Acknowledges their request naturally
296 2. Shows you understand what they want
297 3. Indicates you're about to create a smart plan
298 4. Keep it conversational and warm
299
300 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
310         acknowledgment")
311
312     if not acknowledgment:
313         acknowledgment = "Got it! Let me analyze your
314         request and create a smart plan for you."
315
316
317     ui.console.print(
318         Panel(
319             Text(acknowledgment.strip(),
320                 style="bright_white", justify="center"),
321             title="[bold]Pai[/bold]",
322             box=ROUNDED,
323             border_style="grey50",
324             padding=(1, 2),
325             width=80
326         )
327     )
328
329
330 # === CALL 1: PLANNING PHASE ===

```



```

325     planning_result = execute_planning_call(user_request,
326         context)
327     if not planning_result:
328         ui.print_error("Planning phase failed. Cannot
329             proceed.")
330         if log_file_path:
331             log_session_event(log_file_path, "
332                 FINAL_STATUS", {"status": "Planning failed
333                     ", "success": False})
334         return False
335
336     # Log planning phase
337     if log_file_path:
338         log_session_event(log_file_path, "PLANNING_PHASE"
339             , {"planning_data": planning_result})
340
341     # === DYNAMIC INTERACTION BEFORE EXECUTION ===
342     execution_acknowledgment_prompt = f"""
343     You are Pai, about to execute your plan. Generate a brief
344         , confident response before starting execution.
345
346     USER REQUEST: "{user_request}"
347     PLANNING COMPLETED: Successfully analyzed and created
348         execution plan
349
350     Generate a brief, confident response (1-2 sentences) that
351         :
352
353     1. Shows confidence in your plan
354     2. Indicates you're about to execute intelligently
355     3. Keep it natural and engaging
356     4. Reflect your AI personality
357
358     Examples:
359     - "Perfect! Now let me execute this plan intelligently
360         for you."
361     - "Excellent! I've got a solid plan - time to make it
362         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, planning_result, context,
    log_file_path)
375
376 # Skip complex analysis to save tokens - focus on
    execution success only
377
378 # Generate intelligent next step suggestions only if
    execution failed
379 if not execution_success:
380     next_steps = generate_next_step_suggestions(
        user_request, planning_result,

```

```

        execution_success, context, None)
381
382     if next_steps:
383         # Log next steps
384         if log_file_path:
385             log_session_event(log_file_path, "
                NEXT_STEPS", {"suggestion": next_steps
                    })
386
387         ui.console.print(
388             Panel(
389                 Text(next_steps, style="bright_white"
390                     ),
391                 title="[bold]Next Steps Suggestion[/
                    bold]",
392                 box=ROUNDED,
393                 border_style="grey50",
394                 padding=(1, 2),
395                 width=80
396             )
397         )
398
399     # Show final status - SIMPLIFIED for efficiency
400     if execution_success:
401         status_msg = "Single-Shot Intelligence: SUCCESS"
402         ui.console.print(
403             Panel(
404                 Text(status_msg, style="bold green",
405                     justify="center"),
406                 title="[bold]Mission Accomplished[/bold]"
407                     ,
408                 box=ROUNDED,
409                 border_style="grey50",
410                 padding=(1, 2),
411                 width=80
412             )
413         )
414         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",
418                     justify="center"),
419                 title="[bold]Mission Status[/bold]",
420                 box=ROUNDED,
421                 border_style="grey50",
422                 padding=(1, 2),
423                 width=80
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"
444                             , {"suggestion": next_steps})
445
446     return execution_success

```

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

```

1  def execute_command_sequence(command_sequence: str,
2  context: list) -> tuple[bool, list]:
3
4      """Execute a sequence of commands from the AI."""
5
6      commands = [line.strip() for line in command_sequence
7                  .split('\n') if line.strip()]
8      total_commands = len(commands)
9      successful_commands = 0
10     command_results = []
11
12     # Build execution content
13     content_lines = []
14     content_lines.append(("bold", f"Executing {
15         total_commands} intelligent actions..."))
16     content_lines.append("")
17
18     for i, command_line in enumerate(commands, 1):
19         if not command_line or '::' not in command_line:
20             if command_line.strip():
21                 content_lines.append(("warning", f"
22                     Invalid command format: {command_line}
23                     "))
24                 continue
25
26         # Parse command
27         parts = command_line.split(':', 2)
28         if len(parts) < 2:
29             content_lines.append(("warning", f"Incomplete
30                 command: {command_line}"))
31             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', 'div', 'style', 'script', 'h1', 'h2',
        'form', 'input', 'button')):
32         content_lines.append(("warning", f"Raw HTML/
        CSS detected as command: {command_line
        [:50]}..."))
33         content_lines.append(("info", "Use WRITE::
        filename::description instead of raw
        content!"))
34         continue
35
36     if command_line.strip().startswith(('.', '#', '
        margin', 'padding', 'color', 'background', '
        font', 'border')):
37         content_lines.append(("warning", f"Raw CSS
        detected as command: {command_line
        [:50]}..."))
38         content_lines.append(("info", "Use WRITE::
        filename::description instead of raw CSS!"
        ))
39         continue
40
41     if command not in VALID_COMMANDS:
42         content_lines.append(("warning", f"Unknown
        command: {command} (from: {command_line})"
        ))
43         content_lines.append(("info", f"Valid
        commands: {'', '.join(VALID_COMMANDS)}"))
44         continue
45

```

```

46     # Display current action
47     content_lines.append(("normal", f"[{i}/{
        total_commands}] {command} {param1}"))
48
49     # Execute command
50     success, command_output = execute_single_command(
        command, param1, param2)
51
52     # Add command output to content if any
53     if command_output:
54         if command_output.startswith("
            SYNTAX_HIGHLIGHT:"):
55             parts = command_output.split(":", 2)
56             if len(parts) == 3:
57                 filename = parts[1]
58                 code_content = parts[2]
59                 content_lines.append(("
                    syntax_highlight", filename,
                        code_content))
60             else:
61                 content_lines.append(("ai_output",
                    command_output))
62             else:
63                 content_lines.append(("ai_output",
                    command_output))
64
65     # Collect command result for logging
66     command_results.append({
67         "command": command,
68         "target": param1 if param1 else "",
69         "success": success,
70         "output": command_output if command_output
            else ""
71     })
72
73     if success:
74         successful_commands += 1
75         content_lines.append(("success", "Success"))

```

```

76         else:
77             content_lines.append(("error", "Failed"))
78
79             content_lines.append("")
80
81             # Break on FINISH command
82             if command == "FINISH":
83                 break
84
85             # Show execution summary
86             success_rate = (successful_commands / total_commands)
87                 * 100 if total_commands > 0 else 0
88             content_lines.append(("bold", "Execution Summary:"))
89             content_lines.append(("normal", f"Successful: {
90                 successful_commands}/{total_commands} ({
91                 success_rate:.1f}%)" ))
92
93             # Display all content in a single panel with proper
94             styling
95             from rich.console import Group
96             from rich.text import Text as RichText
97
98             # Convert content to rich renderables with colors
99             rich_content = []
100             for item in content_lines:
101                 if isinstance(item, tuple):
102                     if len(item) == 3 and item[0] == "
103                         syntax_highlight":
104                         _, filename, code_content = item
105
106                         # For terminal display: truncate long
107                         files for better UX
108                         lines = code_content.split('\n')
109                         display_content = code_content
110                         if len(lines) > 20:
111                             display_content = '\n'.join(lines
112                                 [:20]) + f"\n... ({len(lines) -
113                                 20} more lines)"

```



```

106
107         try:
108             from pygments.lexers import
109                 get_lexer_for_filename
110             from pygments.util import
111                 ClassNotFound
112             from rich.syntax import Syntax
113
114             try:
115                 lexer = get_lexer_for_filename(
116                     filename)
117                 lang = lexer.aliases[0]
118             except ClassNotFound:
119                 lang = "text"
120
121             syntax_panel = Panel(
122                 Syntax(display_content, lang,
123                     theme="monokai", line_numbers=
124                         True),
125                 title=f"File {filename}",
126                 border_style="grey50",
127                 expand=False
128             )
129             rich_content.append(syntax_panel)
130         except ImportError:
131             rich_content.append(RichText(f"File
132                 content of {filename}:\n{
133                     display_content}", style="
134                         bright_cyan"))
135
136     else:
137         style_type, text = item[0], item[1]
138         if style_type == "bold":
139             rich_content.append(RichText(text,
140                 style="bold bright_white"))
141         elif style_type == "warning":
142             rich_content.append(RichText(text,
143                 style="bold yellow"))
144         elif style_type == "ai_output":

```

```

134         rich_content.append(RichText(text,
135                                     style="bright_cyan"))
136     elif style_type == "success":
137         rich_content.append(RichText(text,
138                                     style="bold green"))
139     elif style_type == "error":
140         rich_content.append(RichText(text,
141                                     style="bold red"))
142     else: # normal
143         rich_content.append(RichText(text,
144                                     style="bright_white"))
145
146     else:
147         rich_content.append(RichText(str(item), style
148                                     ="bright_white"))
149
150 ui.console.print(
151     Panel(
152         Group(*rich_content),
153         title="[bold]Execution Results[/bold]",
154         box=ROUNDED,
155         border_style="grey50",
156         padding=(1, 2),
157         width=80
158     )
159 )
160
161 return (success_rate >= 80, command_results)
162
163 def execute_single_command(command: str, param1: str,
164                             param2: str) -> tuple[bool, str]:
165     """Execute a single command and return success status
166         and output."""
167
168     try:
169         if command == "READ":
170             content = workspace.read_file(param1)
171             if content is not None:
172                 lines = content.split('\n')

```

```

165         display_content = '\n'.join(lines[:20])
166         if len(lines) > 20:
167             display_content += f"\n... ({len(
168                 lines) - 20} more lines)"
169
170         return True, f"SYNTAX_HIGHLIGHT:{param1
171             }:{content}"
172         return False, f"Could not read file: {param1}
173         "
174
175     elif command == "WRITE":
176         if not param2:
177             return False, "WRITE command requires
178                 description"
179         success = handle_write_command(param1, param2
180             )
181         return success, f"New file written: {param1}"
182         if success else f"Failed to write file: {
183             param1}"
184
185     elif command == "MODIFY":
186         if not param2:
187             return False, "MODIFY command requires
188                 description"
189         success = handle_modify_command(param1,
190             param2)
191         return success, f"File modified: {param1}" if
192             success else f"Failed to modify file: {
193                 param1}"
194
195     elif command == "TREE":
196         path = param1 if param1 else '.'
197         tree_output = workspace.tree_directory(path)
198         if tree_output and "Error:" not in
199             tree_output:
200             return True, f"Directory tree for {path
201                 }:\n{tree_output}"
202         return False, f"Could not get directory tree

```

```

190         for: {path}"
191
192     elif command == "LIST_PATH":
193         path = param1 if param1 else '.'
194         list_output = workspace.list_path(path)
195         if list_output is not None and "Error:" not
196         in list_output:
197             if list_output.strip():
198                 return True, list_output
199             else:
200                 return True, f"Directory '{path}' is
201                 empty"
202         return False, f"Could not list directory: {
203         path}"
204
205     elif command == "MKDIR":
206         result = workspace.create_directory(param1)
207         success = "Success" in result
208         return success, result
209
210     elif command == "TOUCH":
211         result = workspace.create_file(param1)
212         success = "Success" in result
213         return success, result
214
215     elif command == "RM":
216         result = workspace.delete_item(param1)
217         success = "Success" in result
218         return success, result
219
220     elif command == "MV":
221         result = workspace.move_item(param1, param2)
222         success = "Success" in result
223         return success, result
224
225     elif command == "FINISH":
226         message = param1 if param1 else "Task
227         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
231 , data: dict):
232     """Log session events with clear separation between
233     USER and AI."""
234     try:
235         timestamp = datetime.now().strftime("%Y-%m-%d %H
236         :%M:%S")
237
238         if event_type == "SESSION_START":
239             log_line = f"\n[{timestamp}] SESSION STARTED\
240             n"
241             log_line += f"[{timestamp}] Working Directory
242             : {data.get('working_directory', 'unknown'
243             )}\n"
244             log_line += f"[{timestamp}] Session ID: {data
245             .get('session_id', 'unknown')}\n"
246
247         elif event_type == "USER_INPUT":
248             request = data.get('user_request', 'unknown')
249             log_line = f"\n[{timestamp}] USER: {request}\
250             n"
251
252         elif event_type == "PLANNING_PHASE":
253             log_line = f"\n[{timestamp}] AI PLANNING
254             START\n"
255
256             planning_data = data.get('planning_data', {})
257             analysis = planning_data.get('analysis', {})
258
259             log_line += f"[{timestamp}] Intent: {analysis
260             .get('user_intent', 'Unknown')}\n"

```

```

251         log_line += f"[{timestamp}] Context Usage: {
           analysis.get('context_utilization', 'None'
           )}\n"
252         log_line += f"[{timestamp}] Files to read: {
           analysis.get('files_to_read', [])}\n"
253         log_line += f"[{timestamp}] Files to create:
           {analysis.get('files_to_create', [])}\n"
254         log_line += f"[{timestamp}] Files to modify:
           {analysis.get('files_to_modify', [])}\n"
255
256         execution_plan = planning_data.get('
           execution_plan', {})
257         steps = execution_plan.get('steps', [])
258         log_line += f"[{timestamp}] EXECUTION PLAN ({
           len(steps)} steps):\n"
259         for i, step in enumerate(steps, 1):
260             action = step.get('action', 'Unknown')
261             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}: {

```

```

277         cmd} {target}\n"
278         if output:
279             log_line += f"[{timestamp}] OUTPUT: {
280                 output}\n"
281             log_line += f"[{timestamp}] AI EXECUTION END\
282                 n"
283
284     elif event_type == "FINAL_STATUS":
285         status = data.get('status', 'unknown')
286         success_text = "SUCCESS" if data.get('success
287             ') else "FAILED"
288
289         log_line = f"\n[{timestamp}] AI FINAL RESULT:
290             {success_text} - {status}\n"
291
292     elif event_type == "NEXT_STEPS":
293         suggestion = data.get('suggestion', '')
294         if suggestion:
295             log_line = f"\n[{timestamp}] AI
296                 SUGGESTION: {suggestion}\n"
297         else:
298             log_line = ""
299
300     elif event_type == "INTERACTION":
301         log_line = ""
302
303     else:
304         log_line = f"[{timestamp}] {event_type}: {
305             json.dumps(data)}\n"
306
307     if log_line:
308         with open(log_file_path, 'a', encoding='utf-8
309             ') as f:
310             f.write(log_line)
311
312 except Exception as e:
313     pass
314

```

```

307 def handle_write_command(filepath: str, description: str)
    -> bool:
308     """Handle WRITE command with intelligent content
        generation."""
309
310     content_prompt = f"""
311 Generate high-quality content for a file based on the
        description.
312
313 FILE PATH: {filepath}
314 DESCRIPTION: {description}
315
316 REQUIREMENTS:
317 1. Analyze the file extension to determine the
        appropriate language/format
318 2. Create production-quality, well-structured content
319 3. Include appropriate comments and documentation
320 4. Follow best practices for the detected language/format
321 5. Make the code/content immediately usable
322
323 OUTPUT: Return ONLY the file content, no explanations or
        markdown formatting.
324 """
325
326     content = llm.generate_text(content_prompt, "content
        generation")
327
328     if not content:
329         return False
330
331     result = workspace.write_to_file(filepath, content)
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}' -
341             file not found")
342         return False
343
344     modify_prompt = f"""
345 You are an expert code modifier. Modify the existing code
346 based on the description.
347
348 FILE PATH: {filepath}
349 CURRENT CONTENT:
350 ---
351 {existing_content}
352 ---
353
354 MODIFICATION REQUEST: {description}
355
356 REQUIREMENTS:
357 1. Preserve the existing code structure and style
358 2. Make only the necessary changes described
359 3. Maintain code quality and best practices
360 4. Ensure the modified code is syntactically correct
361 5. Add appropriate comments for new functionality
362
363 OUTPUT: Return ONLY the complete modified file content,
364 no explanations.
365 """
366
367     modified_content = llm.generate_text(modify_prompt, "
368         code modification")
369
370     if not modified_content:
371         return False
372
373     success, result = workspace.
374         apply_modification_with_patch(filepath,

```

```

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

```

```

398 - 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
435         behavior."""
436     try:
437         from prompt_toolkit.shortcuts import prompt
438         from prompt_toolkit.key_binding import
439             KeyBindings
440         from prompt_toolkit.keys import Keys
441
442         bindings = KeyBindings()
443
444         @bindings.add(Keys.Enter)
445         def _(event):
446             """Enter submits the input"""
447             event.app.exit(result=event.current_buffer.
448                 text)
449
450         @bindings.add(Keys.Escape, Keys.Enter)
451         def _(event):
452             """Alt+Enter adds new line"""
453             event.current_buffer.insert_text('\n')
454
455         ui.console.print("[dim]Tip: Use Alt+Enter for new
456             line, Enter to submit[/dim]")
457
458         result = prompt(
459             "\nuser> ",
460             multiline=True,
461             key_bindings=bindings,
462             wrap_lines=True,
463             mouse_support=False
464         )
465         return result.strip() if result else ""
466

```

```

463     except Exception as e:
464         ui.console.print(f"[dim]Note: Using simple input
           mode - {str(e)}[/dim]")
465         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

```

```

52     full_path = os.path.realpath(os.path.join(
53         PROJECT_ROOT, norm_path))
54     if not full_path.startswith(os.path.realpath(
55         PROJECT_ROOT)):
56         ui.print_error(f"Operation cancelled. Path '{
57             path}' is outside the project directory.")
58         return False
59
60     # 4. Block access to sensitive files and
61     # directories
62     path_parts = norm_path.replace('\\', '/').split(
63         '/')
64     if any(part in SENSITIVE_PATTERNS for part in
65         path_parts if part):
66         ui.print_error(f"Access to the sensitive path
67             '{path}' is denied.")
68         return False
69
70 except Exception as e:
71     ui.print_error(f"Error during path validation: {e
72         }")
73     return False
74
75 return True
76
77 def tree_directory(path: str = '.') -> str:
78     """Creates a string representation of the directory
79     structure recursively."""
80     if not _is_path_safe(path):
81         return f"Error: Cannot access path '{path}'."
82
83     full_path = os.path.join(PROJECT_ROOT, path)
84     if not os.path.isdir(full_path):
85         return f"Error: '{path}' is not a valid directory
86             ."
87
88     tree_lines = [f"{os.path.basename(full_path)}/"]

```

```

80     def build_tree(directory, prefix=""):
81         try:
82             items = sorted([item for item in os.listdir(
83                 directory) if item not in
84                 SENSITIVE_PATTERNS])
85         except FileNotFoundError:
86             return
87
88         pointers = ['|-- '] * (len(items) - 1) + ['+-- ']
89
90         for pointer, item in zip(pointers, items):
91             tree_lines.append(f"{prefix}{pointer}{item}")
92             item_path = os.path.join(directory, item)
93             if os.path.isdir(item_path):
94                 extension = '| ' if pointer == '-- '
95                 else ' '
96                 build_tree(item_path, prefix=prefix +
97                             extension)
98
99         build_tree(full_path)
100         return "\n".join(tree_lines)
101
102     def list_path(path: str = '.') -> str | None:
103         """
104         Lists all files and subdirectories recursively for a
105         given path in a simple,
106         machine-readable, newline-separated format.
107         """
108         if not _is_path_safe(path):
109             return f"Error: Cannot access path '{path}'."
110
111         full_path = os.path.join(PROJECT_ROOT, path)
112         if not os.path.isdir(full_path):
113             return f"Error: '{path}' is not a valid directory
114                     ."
115
116         path_list = []

```



```

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

```

```

140         return f"Success: Directory deleted: {path}"
141     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(
149         destination):
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         shutil.move(full_source, full_destination)
155         return f"Success: Item moved from '{source}' to
            '{destination}'"
156     except (FileNotFoundError, shutil.Error) as e:
157         return f"Error: Failed to move '{source}': {e}"
158
159 def create_file(file_path: str) -> str:
160     """Creates an empty file and returns a status message
        . """
161     if not _is_path_safe(file_path): return f"Error:
        Access to path '{file_path}' is denied or path is
        not secure."
162     try:
163         full_path = os.path.join(PROJECT_ROOT, file_path)
164         dir_name = os.path.dirname(full_path)
165         if dir_name: os.makedirs(dir_name, exist_ok=True)
166         with open(full_path, 'w') as f: pass
167         return f"Success: New empty file created: {
            file_path}"
168     except IOError as e:
169         return f"Error: Failed to create file: {e}"

```

```

169
170 def create_directory(dir_path: str) -> str:
171     """Creates a directory and returns a status message.
172         """
173     if not _is_path_safe(dir_path): return f"Error:
174         Access to path '{dir_path}' is denied or path is
175         not secure."
176     try:
177         full_path = os.path.join(PROJECT_ROOT, dir_path)
178         os.makedirs(full_path, exist_ok=True)
179         return f"Success: Directory created: {dir_path}"
180     except OSError as e:
181         return f"Error: Failed to create directory: {e}"
182
183 def read_file(file_path: str) -> str | None:
184     """Reads a file and returns its content, or None on
185         failure."""
186     if not _is_path_safe(file_path): return None
187     try:
188         full_path = os.path.join(PROJECT_ROOT, file_path)
189         with open(full_path, 'r') as f:
190             return f.read()
191     except FileNotFoundError:
192         # Let the caller (agent/cli) handle printing the
193         # error
194         return None
195     except IOError as e:
196         ui.print_error(f"Failed to read file: {e}")
197         return None
198
199 def write_to_file(file_path: str, content: str) -> str:
200     """Writes to a file and returns a status message."""
201     if not _is_path_safe(file_path): return f"Error:
202         Access to path '{file_path}' is denied or path is
203         not secure."
204     try:
205         full_path = os.path.join(PROJECT_ROOT, file_path)
206         dir_name = os.path.dirname(full_path)

```

```

200         if dir_name: os.makedirs(dir_name, exist_ok=True)
201         with open(full_path, 'w') as f:
202             f.write(content)
203         return f"Success: New file written: {file_path}"
204     except IOError as e:
205         return f"Error: Failed to write to file: {e}"
206
207
208
209 def apply_modification_with_patch(file_path: str,
210                                  original_content: str, new_content: str, threshold:
211                                  int = 500) -> tuple[bool, str]:
212     """
213     Applies a modification to a file safely by first
214     verifying the scope of changes.
215
216     It generates a diff between the original and new
217     content. If the number of changed
218     lines is within the threshold, it writes the new
219     content to the file. Otherwise,
220     it rejects the change to prevent unintentional
221     overwrites.
222
223     Args:
224         file_path: The path to the file to be modified.
225         original_content: The original, unmodified
226             content of the file.
227         new_content: The new, modified content generated
228             by the LLM.
229         threshold: The maximum number of lines allowed to
230             be changed.
231
232     Returns:
233         A tuple containing:
234         - bool: True if the modification was successful,
235             False otherwise.
236         - str: A message describing the result of the
237             operation.

```

```

227     """
228     if not _is_path_safe(file_path):
229         return False, f"Error: Access to path '{file_path}' is denied or path is not secure."
230
231     # Normalize line endings to reduce false-positive
232     # diffs
233     original_norm = original_content.replace('\r\n', '\n')
234     new_norm = new_content.replace('\r\n', '\n').replace(
235         '\r', '\n')
236
237     original_lines = original_norm.splitlines(keepends=
238         True)
239     new_lines = new_norm.splitlines(keepends=True)
240
241     diff = list(difflib.unified_diff(
242         original_lines,
243         new_lines,
244         fromfile=f"a/{file_path}",
245         tofile=f"b/{file_path}"
246     ))
247
248     # Count only actual change lines, ignore headers and
249     # context lines
250     def _count_changes(d: list[str]) -> tuple[int, int,
251         int]:
252         adds = deletes = 0
253         for line in d:
254             if line.startswith('@@') or line.startswith('
255                 +++') or line.startswith('---') or (line
256                 and line[0] == ' '):
257                 continue
258             if line.startswith('+'):
259                 adds += 1
260             elif line.startswith('-'):
261                 deletes += 1
262         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 (~{ratio:.1%}) exceeds threshold {
                env_threshold} and ratio {max_ratio:.0%}.\n"

```

```

285         f"SOLUTION: Think like Cascade - break this
           into focused, surgical modifications:\n"
286         f" - Focus on ONE specific area/feature at a
           time\n"
287         f" - Ideal: 100-200 lines per modification (
           very focused)\n"
288         f" - Acceptable: 200-500 lines (still
           focused on one area)\n"
289         f" - Use multiple MODIFY commands across
           different steps\n"
290         f" - Example: Instead of 'add all CSS', do '
           add layout CSS', then 'add form CSS', then
           'add button CSS'\n"
291         f"Diff Preview (first 60 lines):\n{
           diff_preview}"
292     )
293     return False, message
294
295     # Atomic write to avoid partial writes
296     try:
297         full_path = os.path.join(PROJECT_ROOT, file_path)
298         dir_name = os.path.dirname(full_path)
299         if dir_name:
300             os.makedirs(dir_name, exist_ok=True)
301             with tempfile.NamedTemporaryFile('w', delete=
                 False, dir=dir_name) as tmp:
302                 tmp.write(new_norm)
303                 tmp_name = tmp.name
304             os.replace(tmp_name, full_path)
305             return True, f"Success: File modified: {file_path}
                 ({changed_lines_count} lines changed; +{
                 add_count}/{del_count})"
306     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
13     correct permissions."""
14     os.makedirs(CONFIG_DIR, exist_ok=True)
15     os.chmod(CONFIG_DIR, 0o700)
16
17 def _default_config() -> dict:
18     """Default single-key configuration."""
19     return {
20         "version": 2, # Version 2 = single-key system
21         "api_key": None
22     }
23
24 def _load_config() -> dict:
25     """Load the single-key configuration."""
26     _ensure_config_dir_exists()
27     if not KEY_FILE.exists():
28         return _default_config()
29
30     try:
31         with open(KEY_FILE, 'r') as f:
32             data = json.load(f)
33
34         # Migrate from old multi-key system if needed
35         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. Using key '{default_id}' as
                           single key.")
42             return {"version": 2, "api_key":
                     migrated_key}
43         elif old_keys:
44             # Use first available key
45             first_key = list(old_keys.values())[0]
46             ui.print_info("Migrating from multi-key
                           system. Using first available key.")
47             return {"version": 2, "api_key":
                     first_key}
48
49         # Ensure proper structure
50         if not isinstance(data, dict):
51             return _default_config()
52
53         return data
54
55     except (json.JSONDecodeError, IOError):
56         ui.print_warning("Configuration file corrupted.
                           Creating new one.")
57         return _default_config()
58
59 def _save_config(config: dict) -> None:
60     """Save the single-key configuration."""
61     try:
62         _ensure_config_dir_exists()
63         with open(KEY_FILE, 'w') as f:
64             json.dump(config, f, indent=2)
65             os.chmod(KEY_FILE, 0o600)

```

```

66     except Exception as e:
67         ui.print_error(f"Failed to save configuration: {e
68             }")
69
70 def set_api_key(api_key: str) -> None:
71     """Set the API key."""
72     if not api_key or not isinstance(api_key, str):
73         ui.print_error("Invalid API key provided.")
74         return
75
76     if not api_key.startswith("AIza"):
77         ui.print_warning("Warning: API key doesn't look
78             like a Google API key (should start with 'AIza
79             ')")
80
81     config = _load_config()
82     config["api_key"] = api_key
83     _save_config(config)
84
85     masked_key = mask_api_key(api_key)
86     ui.print_success(f"[OK] API key set successfully: {
87         masked_key}")
88
89 def get_api_key() -> Optional[str]:
90     """Get the current API key."""
91     config = _load_config()
92     return config.get("api_key")
93
94 def save_api_key(api_key: str):
95     """Legacy compatibility function."""
96     set_api_key(api_key)
97
98 def remove_api_key() -> None:
99     """Remove the stored API key."""
100     config = _load_config()
101
102     if not config.get("api_key"):
103         ui.print_info("No API key is currently set.")

```

```

100         return
101
102     config["api_key"] = None
103     _save_config(config)
104     ui.print_success("[OK] API key removed successfully."
105                     )
106
107 def show_api_key() -> None:
108     """Show the current API key (masked)."""
109     api_key = get_api_key()
110
111     if not api_key:
112         ui.print_info("No API key is currently set.")
113         ui.print_info("Use 'pai config set <API_KEY>' to
114             set one.")
115         return
116
117     masked_key = mask_api_key(api_key)
118     ui.print_info(f"Current API key: {masked_key}")
119
120 def mask_api_key(api_key: str) -> str:
121     """Mask API key for display purposes."""
122     if not api_key or len(api_key) < 10:
123         return "***"
124
125     return f"{api_key[:6]}...{api_key[-4:]}"
126
127 def is_configured() -> bool:
128     """Check if API key is configured."""
129     api_key = get_api_key()
130     return api_key is not None and len(api_key.strip()) >
131         0
132
133 def validate_api_key() -> tuple[bool, str]:
134     """Validate that API key is configured and looks
135         correct."""
136     api_key = get_api_key()

```

```

134     if not api_key:
135         return False, "No API key configured. Use 'pai
            config set <API_KEY>' to set one."
136
137     if not api_key.startswith("AIza"):
138         return False, "API key doesn't look like a Google
            API key (should start with 'AIza')"
139
140     if len(api_key) < 20:
141         return False, "API key seems too short to be
            valid"
142
143     return True, "API key looks valid"
144
145 # Legacy compatibility functions (simplified)
146 def add_api_key(key_id: str, api_key: str) -> None:
147     """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
9              AI Coding Companion.",
10         epilog="Use 'pai config set <API_KEY>' to
11             configure. Run 'pai' to start the intelligent
12             agent."
13     )
14     subparsers = parser.add_subparsers(dest='command',
15         help='Available commands')
16
17     # Main agent command (default)
18     parser_auto = subparsers.add_parser('auto', help='
19         Start the single-shot AI agent session.')
20     parser_auto.add_argument('--model', type=str, help='
21         LLM model name (e.g., gemini-2.5-flash-lite)')
22     parser_auto.add_argument('--temperature', type=float,
23         help='LLM sampling temperature (e.g., 0.2)')
24
25     # Simplified config management
26     parser_config = subparsers.add_parser('config', help='
27         Manage API key configuration')
28     config_subparsers = parser_config.add_subparsers(dest
29         = 'config_cmd', help='Config commands')
30
31

```

```

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
71     # Default: start agent
72     # Check API key before starting
73     if not config.is_configured():
74         ui.print_error("[ERROR] No API key configured.")
75         ui.print_info("Use 'pai config set <API_KEY>' to
76             set your Google Gemini API key.")
77         return 1
78
79     # Configure LLM runtime if flags provided
80     model = getattr(args, 'model', None)
81     temperature = getattr(args, 'temperature', None)
82     if model is not None or temperature is not None:
83         llm.set_runtime_model(model, temperature)
84
85     try:
86         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         """
28     console.print(f"[success][OK] {message}[/success]")
29
30 def print_error(message: str):
31     """Displays an error message with a cross icon."""
32     console.print(f"[error][ERROR] {message}[/error]")
33
34 def print_warning(message: str):
35     """Displays a warning message."""
36     console.print(f"[warning]! {message}[/warning]")
37
38 def print_info(message: str):
39     """Displays an informational message."""
40     console.print(f"[info]i {message}[/info]")
41
42 def print_action(message: str):
43     """Displays an action being performed by the agent.
44         """
45     console.print(f"[action]-> {message}[/action]")
46
47 def display_panel(content: str, title: str, language: str
48 = None):
49     """Displays content within a panel, with optional
50     syntax highlighting."""
51     if language:
52         # Use Syntax for code highlighting
53         display_content = Syntax(content, language, theme
54             ="monokai", line_numbers=True)
55     else:
56         display_content = content
57
58     console.print(Panel(display_content, title=f"[bold
59         grey50]{title}[/bold grey50]", border_style="
60         grey50", expand=False))
61
62

```

```

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 config set <API_KEY>'")
77         model = None
78         return False
79
80     try:
81         genai.configure(api_key=api_key)
82         if model is None:
83             # Build model using stored runtime prefs
84             name = _runtime.get("name") or DEFAULT_MODEL
85             temp = _runtime.get("temperature") if
               _runtime.get("temperature") is not None
               else DEFAULT_TEMPERATURE
86             generation_config = {"temperature": temp}
87             model = genai.GenerativeModel(name,
               generation_config=generation_config)
88         return True

```

```

89     except Exception as e:
90         ui.print_error(f"Failed to configure API key: {e}
91                        ")
92         model = None
93         return False
94
95 def _is_rate_limit_error(error: Exception) -> bool:
96     """Detect if an exception is a rate limit error.
97
98     Args:
99         error: The exception to check
100
101     Returns:
102         True if it's a rate limit error, False otherwise
103     """
104     error_msg = str(error).lower()
105
106     # Common rate limit indicators
107     rate_limit_keywords = [
108         'rate limit', 'rate_limit', 'ratelimit',
109         'quota', 'quota exceeded',
110         'resource exhausted', 'resourceexhausted',
111         '429', 'too many requests',
112         'limit exceeded', 'requests per minute'
113     ]
114
115     return any(keyword in error_msg for keyword in
116                rate_limit_keywords)
117
118 def _clean_response_text(text: str) -> str:
119     """Clean markdown artifacts from LLM response.
120
121     Args:
122         text: Raw response text from LLM
123
124     Returns:
125         Cleaned text without markdown code blocks
126     """

```

```

125     cleaned_text = text.strip()
126
127     # Remove all common markdown code block patterns
128     code_block_prefixes = [
129         "```python", "```html", "```css", "```javascript"
130         , "```js",
131         "```typescript", "```ts", "```json", "```yaml", "
132         ```", "```yml",
133         "```bash", "```sh", "```diff", "```xml", "```sql"
134         ,
135         "```java", "```cpp", "```c", "```go", "```rust",
136         "```ruby",
137         "```php", "```markdown", "```md", "```text", "```
138         txt", "```"
139     ]
140
141     for prefix in code_block_prefixes:
142         if cleaned_text.startswith(prefix):
143             cleaned_text = cleaned_text[len(prefix):].
144             strip()
145             break
146
147     # Remove trailing code block markers
148     if cleaned_text.endswith("```"):
149         cleaned_text = cleaned_text[:-len("```")].strip()
150
151     # Remove any remaining language tags at the start
152     lines = cleaned_text.split('\n')
153     if lines and len(lines[0].strip()) < 20 and lines[0].
154     strip().lower() in [
155         'html', 'css', 'javascript', 'js', 'python', '
156         json', 'yaml',
157         'bash', 'sh', 'diff', 'xml', 'sql', 'java', 'cpp'
158         , 'c', 'go',
159         'rust', 'ruby', 'php', 'markdown', 'md', 'text',
160         'txt', 'on'
161     ]:
162         cleaned_text = '\n'.join(lines[1:]).strip()

```

```

153
154     return cleaned_text
155
156 def generate_text(prompt: str, call_purpose: str = "
157     thinking") -> str:
158     """
159     Generate text with single API key - optimized for 2-
160     call system.
161
162     Args:
163         prompt: The prompt to send to the LLM
164         call_purpose: Purpose of the call for logging (e.
165             g., "planning", "execution")
166
167     Returns:
168         The cleaned response text, or empty string if
169         failed
170     """
171     global model
172
173     # Ensure model is configured
174     if model is None:
175         if not _prepare_runtime():
176             return ""
177
178     try:
179         # Show status with purpose
180         status_msg = f"[bold yellow]Agent {call_purpose
181             }..."
182
183         with ui.console.status(status_msg, spinner="dots"
184             ):
185             response = model.generate_content(prompt)
186
187         # Success! Clean and return the response
188         cleaned_text = _clean_response_text(response.text
189             )

```

```

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.
188                          prompt_token_count} -> {usage.
189                          candidates_token_count}")
190
191         return cleaned_text
192
193     except Exception as e:
194         is_rate_limit = _is_rate_limit_error(e)
195
196         if is_rate_limit:
197             ui.print_error("[ERROR] Rate limit reached.
198                          Please wait a few minutes before trying
199                          again.")
200             ui.print_info("Consider using a different API
201                          key if available.")
202         else:
203             ui.print_error(f"[ERROR] LLM API error: {e}")
204
205         return ""
206
207 def test_api_connection() -> bool:
208     """Test if API connection works."""
209     test_response = generate_text("Say 'Hello' if you can
210                                  hear me.", "connection test")
211     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
4  software development.
5  """

```



```

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

```

32  @echo "Export complete."
33
34  venv-activate:
35      @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.