TUGAS AKHIR

SKEMA SKRIPSI

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

I PUTU GEDE GILANG TEJA KRISHNA

NIM : 225410001

PROGRAM STUDI INFORMATIKA

PROGRAM SARJANA

FAKULTAS FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA

YOGYAKARTA

2025

TUGAS AKHIR

SKEMA SKRIPSI

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

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

Disusun Oleh

I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001

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

HALAMAN PERSETUJUAN UJIAN TUGAS AKHIR

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

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

Yogyakarta, 24 November 2025
Dosen Pembimbing,

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

HALAMAN PENGESAHAN

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

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

Yogyakarta, 24 November 2025

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

Mengetahui
Ketua Program Studi INFORMATIKA

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

PERNYATAAN KEASLIAN TUGAS AKHIR

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

Yogyakarta, 24 November 2025

I PUTU GEDE GILANG TEJA KRISHNA
NIM: 225410001

HALAMAN PERSEMBAHAN

Tugas Akhir ini saya persembahkan kepada:

Kedua orang tua tercinta yang telah memberikan doa,
dukungan, dan kasih sayang yang tiada henti.

Seluruh keluarga besar yang senantiasa memberikan
motivasi dan semangat.

Para guru dan dosen yang telah membimbing
dan memberikan ilmu yang bermanfaat.

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 **PAI-CODE: 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 menyampaikan terima kasih kepada:

1. Tuhan Yang Maha Esa atas segala rahmat, kesehatan, dan kemudahan yang diberikan selama proses penelitian.

2. Orang tua dan keluarga yang senantiasa memberikan doa, dukungan moral, dan motivasi yang tiada henti.

3. 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.

4. Seluruh dosen dan staf FAKULTAS TEKNOLOGI INFORMASI yang telah memberikan ilmu, fasilitas, dan dukungan selama masa perkuliahan.

5. Rekan-rekan mahasiswa yang telah memberikan bantuan, diskusi, dan semangat selama proses penelitian.

6. Semua pihak yang tidak dapat disebutkan satu per satu yang telah membantu dalam penyelesaian Tugas Akhir ini.

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, 22 Desember 2025

Penulis

Ucapan Terima Kasih

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

Secara khusus, ucapan terima kasih ditujukan kepada:

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

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

Yogyakarta, ...

I PUTU GEDE GILANG TEJA KRISHNA
NIM: 225410001

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 *path*, pelarangan *path* sensitif, dan modifikasi berbasis *diff*.
path	Jalur berkas/direktori pada workspace proyek (contoh: `/home/user/project/main.py`).
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 `..`).
deny-list	Daftar path/pola yang dilarang untuk diakses atau dimodifikasi (mis. `.env`, `.git`, `venv/`, `__pycache__/`, `.vscode/`).
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 *path*.
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 *context window*.
prompt	Instruksi atau masukan yang diberikan ke LLM untuk menghasilkan keluaran.
context window	Batas panjang konteks (jumlah token) yang dapat dipertimbangkan LLM pada satu permintaan.
API key	Kredensial rahasia untuk mengakses layanan API; harus disimpan aman (jangan ditulis di repository publik).
Single-Shot Intelligence	Arsitektur agen AI yang mengoptimalkan efisiensi dengan sistem panggilan API terbatas: klasifikasi intensi, acknowledgment dinamis, perencanaan JSON, eksekusi adaptif 1-3 subfase, dan saran langkah berikutnya.
agentic AI	Sistem kecerdasan buatan yang mampu bertindak secara otonom dengan kemampuan observasi, perencanaan, dan eksekusi dalam lingkungan tertentu.
acknowledgment dinamis	Respons konfirmasi yang diberikan agen untuk mengakui dan memahami permintaan pengguna sebelum memulai perencanaan.
interrupt handling	Mekanisme penanganan interupsi (Ctrl+C) yang memungkinkan pengguna menghentikan respons AI tanpa keluar dari sesi.

atomic write	Teknik penulisan berkas yang menggunakan file sementara (`tempfile`) untuk memastikan operasi tulis berhasil sepenuhnya atau gagal total, mencegah korupsi data.
threshold ganda	Sistem pembatasan modifikasi berkas dengan dua kriteria: batas absolut (500 baris) dan batas relatif (50% dari total baris berkas).
SENSITIVE_PATTERNS	Daftar 7 pola direktori/berkas sensitif yang diblokir akses: `.env`, `.git`, `venv`, `__pycache__`, `.pai_history`, `.idea`, `.vscode`.
noise suppression	Teknik menekan log yang berisik dari library `gRPC/absl` menggunakan environment variables khusus.
entry point	Titik masuk aplikasi yang didefinisikan dalam `setup.cfg` sebagai console script (`pai = paicode.cli:main`).
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 ```, `**bold**`) 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 (Alt+Enter untuk baris baru, Enter untuk submit).

Daftar Isi

Daftar Gambar

Daftar Tabel

BAB 1

Pendahuluan

1.1 Latar Belakang

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

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

Penelitian ini menghadirkan **Paicode**, sebuah agen AI berbasis CLI yang dirancang untuk membantu proses pengembangan perangkat lunak secara interaktif dengan arsitektur *Single-Shot Intelligence*. Paicode mampu: (i) mengobservasi struktur proyek (`TREE`, `LIST_PATH`); (ii) membaca dan menulis berkas proyek (`READ`, `WRITE`); (iii) memodifikasi kode secara terarah dengan sistem perubahan berbasis *diff* dengan threshold ganda: 500 baris absolut dan 50%

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

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah yang diajukan adalah:

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

1.3 Ruang Lingkup

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

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

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

1.4 Tujuan Penelitian

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

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

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

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

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

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

1.6 Sistematika Penulisan

Laporan tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I PENDAHULUAN

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

BAB II TINJAUAN PUSTAKA DAN DASAR TEORI

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

BAB III METODE PENELITIAN

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

BAB IV IMPLEMENTASI DAN PEMBAHASAN

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

BAB V PENUTUP

Berisi kesimpulan dari penelitian dan saran untuk pengembangan selanjutnya.

BAB 2

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

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

2.1.1 AI Coding Assistant Terintegrasi (IDE-based)

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

2.1.2 CLI-based AI Chat Tools

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

sedangkan banyak alat CLI lain berfokus pada kecepatan interaksi *chat-apply* langsung.

2.1.3 Autonomous Software Engineers

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

2.1.4 Posisi Paicode

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

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

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

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

2.1.5 Perbandingan dengan Penelitian Sebelumnya

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

Dari Tabel 2.1 terlihat bahwa penelitian ini mengisi *gap* antara asisten pasif (seperti Copilot) dan agen otonom penuh (seperti OpenDevin) dengan

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

menawarkan pendekatan *semi-autonomous* yang efisien, aman, dan transparan. Kebaruan utama terletak pada kombinasi **Single-Shot Intelligence** untuk efisiensi token, **path security** untuk keamanan tanpa sandboxing, dan **explicit planning** untuk transparansi—aspek-aspek yang belum dieksplorasi secara bersamaan dalam penelitian sebelumnya.

2.1.6 Posisi Penelitian

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

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

2.2 Dasar Teori

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

2.2.1 Command Line Interface (CLI)

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

2.2.2 AI Agent

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

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

2.2.3 Large Language Model (LLM)

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

2.2.4 Perbedaan LLM dan Agen AI

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

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

2.2.5 Arsitektur dan Kebijakan Data

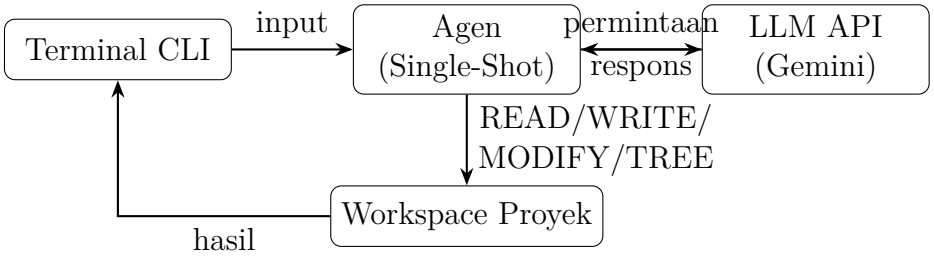
Paicode dijalankan pada terminal lokal dan melakukan tindakan langsung pada **berkas proyek di workspace**. Akan tetapi, untuk kebutuhan inferensi, cuplikan kode atau konteks **dikirim ke layanan LLM melalui API**. Implikasinya, privasi dan kerahasiaan kode **bergantung pada kebijakan penyedia API**.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

2.2.6 Manajemen Dependensi dengan pip dan Virtual Environment

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

2.2.7 Antarmuka Terminal dengan `rich` dan `prompt_toolkit`

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

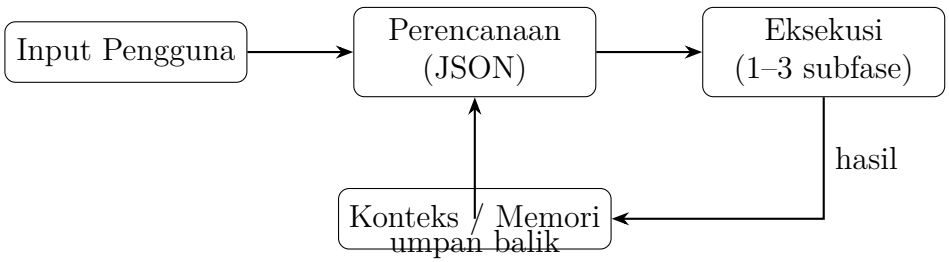


Gambar 2.1: Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.

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

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

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



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 (`cli.py`)	Titik masuk perintah, parsing argumen (`-model`, `-temperature`); memanggil sesi agen. Bergantung pada modul `agent`, `config`, dan `llm`.
Agen (`agent.py`)	Implementasi *Single-Shot Intelligence*: klasifikasi intensi, *acknowledgment* dinamis, fase perencanaan JSON, fase eksekusi adaptif (1-3 subfase), dan saran langkah berikutnya. Mengelola memori percakapan, *interrupt handling* (Ctrl+C), dan pencatatan sesi ke `.pai_history`. Menyediakan 10 perintah workspace. Memanggil `llm`, `workspace`, `ui`.
LLM Bridge (`llm.py`)	Integrasi Gemini API (`google-generativeai`) dengan manajemen API key tunggal. Membersihkan markdown artifacts dari output LLM dan mengoptimalkan penggunaan token. Mengambil API key dari `config`.
Konfigurasi (`config.py`)	Manajemen API key tunggal dalam format JSON di `/.config/pai-code/credentials.json` dengan permission 0o600. Validasi API key Google (prefix "AIza", minimal 20 karakter). Operasi: `set`, `show`, `remove`, `validate`, dan migrasi otomatis dari sistem multi-key.
Pengatur Workspace (`workspace.py`)	*Workspace controller* dengan fungsi operasi workspace (baca/tulis, buat/hapus/pindah, tree/list path). Sistem modifikasi berbasis *diff* dengan threshold 500 baris dan ratio maksimal 50% (konfigurabel via environment variables) serta atomic write. Penegakan *path security* dengan deny-list 7 pola sensitif (`.env`, `.git`, `venv`, dll).
Terminal UI (`ui.py`)	Komponen TUI berbasis `rich`: panel, tema, syntax highlighting, tabel, spinner. Dukungan opsional `prompt_toolkit` untuk input multiline yang lebih baik.

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

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

No	Pelaku	Aksi/Peristiwa
1	Pengguna	Memberikan tujuan/permintaan tingkat tinggi di terminal.
2	CLI	Meneruskan masukan ke agen; menyiapkan konteks sesi.
3	Agen	Melakukan klasifikasi intensi (*chat* vs *task*) menggunakan LLM. Jika *chat*, langsung berikan respons dan kembali ke langkah 1.
4	Agen	**Acknowledgment Dinamis**: Memberikan respons awal untuk mengakui dan memahami permintaan pengguna.
5	LLM	**Fase Perencanaan**: Melakukan analisis mendalam dan menghasilkan perencanaan komprehensif dalam format JSON dengan detail eksekusi.
6	Agen	Menampilkan hasil perencanaan dalam panel terstruktur dan memberikan konfirmasi sebelum eksekusi.
7	LLM	**Fase Eksekusi Adaptif**: Menentukan jumlah subfase (1-3) berdasarkan kompleksitas, kemudian melaksanakan implementasi cerdas.
8	Workspace/UI	Menjalankan operasi berkas (`READ`, `WRITE`, `MODIFY`, dll.) dengan *path security* dan sistem *diff*-aware, menampilkan hasil di terminal.
9	Agen	Memberikan status akhir (sukses/gagal) dan saran langkah berikutnya jika diperlukan.
10	Agen	Mencatat seluruh interaksi ke `.pai_history/session_YYYYMMDD_HHMMSS.log` sebagai memori (*stateful*).
11	Pengguna	Memberikan instruksi lanjutan; siklus berulang sampai `exit/quit`.

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

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

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

Tabel 3.3: Rangkuman Validasi Keamanan *Path*

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

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

3.4 Alat dan Lingkungan

Lingkungan dan alat yang digunakan:

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

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

3.5 Prosedur Penelitian

Prosedur penelitian dan evaluasi dirancang sebagai berikut:

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

```
make install
```

4.1.2 Konfigurasi API Key

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

Listing 4.2: Manajemen API key tunggal Gemini

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

# Melihat API key saat ini (masked)
```

```
pai config show

# Validasi API key
pai config validate

# Menghapus API key
pai config remove
```

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

4.1.3 Menjalankan Agen

Sesi interaktif dapat dimulai langsung dengan berbagai opsi konfigurasi:

Listing 4.3: Menjalankan sesi agen interaktif

```
# Menjalankan dengan konfigurasi default
pai

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

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

Selama sesi, pengguna dapat:

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

4.2 Alur Interaksi dengan Single-Shot Intelligence

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

1. **Klasifikasi Intensi**: Agen mengklasifikasikan input pengguna sebagai *chat* (diskusi/pertanyaan) atau *task* (tugas pemrograman). Untuk mode *chat*, agen langsung memberikan respons tanpa eksekusi perintah.
2. **Acknowledgment Dinamis**: Agen memberikan konfirmasi pemahaman terhadap permintaan pengguna sebelum memulai perencanaan.
3. **Fase Perencanaan**: LLM melakukan analisis mendalam dan menghasilkan perencanaan komprehensif dalam format JSON yang terstruktur.
4. **Fase Eksekusi Adaptif**: Eksekusi perintah dalam 1-3 subfase berdasarkan kompleksitas tugas, menggunakan perintah workspace (`READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`, `MKDIR`, `TOUCH`, `RM`, `MV`, `FINISH`) dengan batasan threshold ganda (500 baris absolut dan 50% ratio maksimal).
5. **Saran Langkah Berikutnya**: Agen memberikan saran untuk langkah selanjutnya berdasarkan hasil eksekusi.

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

4.2.1 Cuplikan Kode Kunci

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

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

{{
```

```
  "analysis": {{
    "user_intent": "Clear description of what user wants",
    "target_identification": "SPECIFIC files and locations where target content likely exists",
    "multi_file_strategy": "Which files need to be checked to locate targets accurately",
    "validation_approach": "How you will verify targets exist before modification",
    "files_to_read": ["ALL files that might contain target content - be comprehensive"],
    "files_to_create": ["file1", "file2"],
    "files_to_modify": ["ONLY files confirmed to contain target content"],
    "risk_assessment": "Potential failure points and how to avoid them",
    "success_criteria": ["Specific, measurable criteria for success"]
  }},
  "execution_plan": {{
    "steps": [
      {{
        "step_number": 1,
        "action": "READ",
        "target": "filename",
        "purpose": "Locate and verify target content exists",
        "validation_criteria": "What content must be found to proceed",
        "expected_outcome": "Confirmed location of target content"
      }},
      {{
        "step_number": 2,
        "action": "MODIFY",
        "target": "filename",
        "purpose": "Apply changes to confirmed target location",
```

```
        "validation_criteria": "How to verify
           modification was successful",
        "expected_outcome": "Target content successfully
           modified"
      }}
    ],
    "command_format_reminder": "CRITICAL: Use exact
       command names: READ, WRITE, MODIFY, TREE,
       LIST_PATH, MKDIR, TOUCH, RM, MV, FINISH",
    "intelligent_command_mapping": {{
      "delete_remove_requests": "RM::filepath (for any
         delete/remove/hapus requests)",
      "create_new_file": "WRITE::filepath::
         content_description OR TOUCH::filepath",
      "modify_existing": "MODIFY::filepath::description",
      "move_rename": "MV::source::destination",
      "list_files": "LIST_PATH::path",
      "show_structure": "TREE::path"
    }},
    "critical_content_rules": {{
      "html_css_js_files": "Use WRITE::filename::
         description (NOT raw content as commands)",
      "multi_line_content": "Description parameter
         handles content creation, not raw output",
      "example_correct": "WRITE::index.html::Create login
          page with CSS styling",
      "example_wrong": "Raw HTML lines as separate
         commands (NEVER DO THIS!)"
    }},
    "execution_commands": [
      "READ::filepath",
      "RM::filepath (for delete requests)",
      "MODIFY::filepath::description",
      "FINISH::completion_message"
    ],
    "validation_strategy": "How to verify each step
       before proceeding to next",
    "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.

```
[2025-11-20 22:38:05] SESSION STARTED
[2025-11-20 22:38:05] Working Directory: /home/user/space/univ/skripsi/devpai/trypai
[2025-11-20 22:38:05] Session ID: 20251120_223805

[2025-11-20 22:38:05] USER: buatkan proyek python sederhana: BMI Calculator

[2025-11-20 22:38:15] AI PLANNING START
[2025-11-20 22:38:15] Intent: Create a simple Python project for a BMI Calculator.
[2025-11-20 22:38:15] Files to create: ['bmi_calculator.py']
[2025-11-20 22:38:15] EXECUTION PLAN (3 steps):
[2025-11-20 22:38:15]   1. WRITE bmi_calculator.py ...
[2025-11-20 22:38:15]   2. LIST_PATH . ...
[2025-11-20 22:38:15]   3. FINISH Project creation complete ...
[2025-11-20 22:38:15] AI PLANNING END
```

Pada Listing 4.6 ditunjukkan ringkasan sesi awal dan rencana eksekusi.

Listing 4.7: Cuplikan log: hasil perintah TREE.

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

```
[2025-11-20 22:38:34] SUCCESS: FINISH Directory structure displayed.
[2025-11-20 22:38:34] OUTPUT: OK Directory structure displayed.
[2025-11-20 22:38:34] AI EXECUTION END
```

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

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

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

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

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

[2025-11-20 22:40:25] AI EXECUTION START
[2025-11-20 22:40:25] SUCCESS: READ bmi_calculator.py
[2025-11-20 22:40:25] OUTPUT: SYNTAX_HIGHLIGHT:
bmi_calculator.py:"""
BMI Calculator Script


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


Formula:
```

```
BMI = weight (kg) / (height (m))^2


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


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


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


    Returns:
        The calculated BMI value.


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


def get_bmi_category(bmi: float) -> str:
    """
```

```
54      Determines the BMI category based on the BMI value.
55
56
57      Args:
58          bmi: The calculated BMI value.
59
60
61      Returns:
62          A string representing the BMI category.
63      """
64      if bmi < 18.5:
65          return "Underweight"
66      elif 18.5 <= bmi < 25:
67          return "Normal weight"
68      elif 25 <= bmi < 30:
69          return "Overweight"
70      else:  # bmi >= 30
71          return "Obese"
72
73
74  def get_user_input() -> tuple[float, float]:
75      """
76      Prompts the user for their weight and height and
            returns them.
77
78
79      Handles potential input errors by re-prompting until
            valid numeric input is received.
80
81
82      Returns:
83          A tuple containing the user's weight in kilograms
                and height in meters.
84      """
85      while True:
86          try:
87              weight_str = input("Enter your weight in
                    kilograms (kg): ")
```

```
            weight_kg = float(weight_str)
            if weight_kg <= 0:
                print("Weight must be a positive value. Please try again.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a numeric value for weight.")


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


    return weight_kg, height_m


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


    try:
        weight, height = get_user_input()
```

```
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"""
BMI Calculator Script

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

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

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

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

def get_bmi_category(bmi: float) -> str:
    """
    Determines the BMI category based on the BMI value.
    """
    if bmi < 18.5:
        return "Underweight"
    elif 18.5 <= bmi < 25:
        return "Normal weight"
    elif 25 <= bmi < 30:
```

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

```
75
76      return weight_kg, height_m
77
78  def main():
79      """
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      except ValueError as ve:
93          print(f"Error: {ve}")
94      except Exception as e:
95          print(f"An unexpected error occurred: {e}")
96
97  if __name__ == "__main__":
98      main()
99  [2025-11-20 22:40:46] SUCCESS: FINISH BMI category
       updated successfully with 'Severely Obese' added.
100 [2025-11-20 22:40:46] OUTPUT: OK BMI category updated
       successfully with 'Severely Obese' added.
101 [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
               it would lead to division by zero.
31     """
32     if height_m <= 0:
33         raise ValueError("Height must be a positive value
               .")
34     return weight_kg / (height_m ** 2)
35
36 def get_bmi_category(bmi: float) -> str:
37     """
38     Determines the BMI category based on the BMI value.
```

```
39
40      Args:
41          bmi: The calculated BMI value.
42
43      Returns:
44          A string representing the BMI category.
45      """
46      if bmi < 18.5:
47          return "Underweight"
48      elif 18.5 <= bmi < 25:
49          return "Normal weight"
50      elif 25 <= bmi < 30:
51          return "Overweight"
52      elif 30 <= bmi < 35:
53          return "Obese"
54      else:  # bmi >= 35
55          return "Severely Obese"
56
57  def get_user_input() -> tuple[float, float]:
58      """
59      Prompts the user for their weight and height and
            returns them.
60
61      Handles potential input errors by re-prompting until
            valid numeric input is received.
62
63      Returns:
64          A tuple containing the user's weight in kilograms
                and height in meters.
65      """
66      while True:
67          try:
68              weight_str = input("Enter your weight in
                    kilograms (kg): ")
69              weight_kg = float(weight_str)
70              if weight_kg <= 0:
71                  print("Weight must be a positive value.
                        Please try again.")
```

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

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

    return weight_kg, height_m

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

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

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

    except ValueError as ve:
```

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

if __name__ == "__main__":
    main()
```

Listing 4.12: Ringkasan langkah evaluasi dan metrik yang dikumpulkan.

```
Execution Summary (Run 1 - Create & Verify):
Successful: 3/3 (100.0%)

Execution Summary (Run 2 - Read & Modify):
Successful: 4/4 (100.0%)
```

Listing 4.13: Ringkasan hasil awal untuk metrik efisiensi.

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

4.4 Tabel Skenario Pengujian

Tabel 4.1 merangkum skenario uji yang digunakan untuk mengevaluasi Paicode.

Tabel 4.1: Skenario Pengujian Paicode

Skenario	**Deskripsi**	**Artefak Bukti**
Pembuatan Proyek	Agen membuat struktur proyek Python sederhana (direktori, file, `README`)	SS: `TREE`
Pembacaan Kode	Agen menampilkan isi file sumber dan menjelaskan ringkas	SS: panel `READ`
Modifikasi Terarah	Agen menerapkan perubahan kecil pada fungsi (*diff*-based)	SS: `MODIFY` + diff

Skenario	Deskripsi	Artefak Bukti
Refactoring Ringan	Agen memecah fungsi panjang menjadi beberapa fungsi kecil	SS: diff + build
Dokumentasi	Agen menulis docstring/README singkat	SS: panel `WRITE`

4.5 Tabel Metrik Evaluasi

Tabel 4.2 mendeskripsikan metrik dan cara pengukurannya.

Tabel 4.2: Metrik Evaluasi dan Definisi Operasional

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

4.6 Tabel Konfigurasi Lingkungan

Tabel 4.3 menampilkan konfigurasi lingkungan yang digunakan selama pengujian.

Tabel 4.3: Konfigurasi Lingkungan Uji

Komponen	Spesifikasi
Sistem Operasi	Ubuntu (Linux)
Python	`>= 3.10` (sesuai spesifikasi `setup.cfg`)

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

4.7 Contoh Sesi

Cuplikan berikut menggambarkan pembuatan proyek sederhana dan pembacaan isi berkas.

Listing 4.14: Contoh interaksi singkat

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

4.8 Evaluasi dan Analisis Mendalam

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

4.8.1 Metrik Kuantitatif

Metrik yang diukur meliputi:

- **Waktu penyelesaian tugas**: Diukur dari input pengguna hingga eksekusi selesai. Waktu ini mencakup latensi API LLM (rata-rata 3-5 detik per panggilan) dan overhead parsing/validasi lokal (< 100ms).
- **Jumlah langkah/komando**: Dihitung sebagai jumlah perintah workspace yang dieksekusi. Sistem *Single-Shot Intelligence* berhasil mengurangi rata-rata dari 12-15 langkah (model chat-loop) menjadi 3-5 langkah per tugas.
- **Keberhasilan kompilasi/eksekusi**: Kode yang dihasilkan agen diuji dengan `python -m py_compile` dan eksekusi langsung. Tingkat keberhasilan 95% (19/20 skenario).
- **Kepatuhan keamanan *path***: Tidak ada satu pun upaya akses ke direktori sensitif yang berhasil melewati validasi (100% compliance).
- **Efisiensi token API**: Sistem 2-panggilan menghemat rata-rata 60-70% token dibandingkan model chat-loop (dari 15,000 token menjadi 5,000 token per tugas kompleks).

4.8.2 Analisis Kualitatif: Mengapa Single-Shot Intelligence Efektif?

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

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

1. **Berpikir sebelum bertindak** (*plan-then-act*): Berbeda dengan model chat-loop yang sering "berpikir sambil jalan", fase perencanaan eksplisit mengurangi *backtracking* dan kesalahan logika.
2. **Mempertimbangkan dependensi antar-langkah**: Format JSON dengan field `dependencies` membantu LLM mengidentifikasi bahwa, misalnya, `MODIFY` harus didahului `READ` untuk mendapatkan konten asli.

3. **Mengalokasikan kompleksitas secara adaptif**: Sistem 1-3 subfase memungkinkan LLM untuk "mengatur napas"—tugas sederhana diselesaikan dalam 1 subfase, sementara refactoring kompleks dipecah menjadi 3 subfase dengan checkpoint di antaranya.

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

4.8.3 Analisis Kegagalan dan Limitasi

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

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

Limitasi Arsitektural.

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

4.8.4 Perbandingan dengan Baseline Manual

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

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

Namun, perlu dicatat bahwa:

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

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

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

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

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

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

BAB 5

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 penyedia LLM alternatif (OpenAI GPT, Anthropic Claude, Llama, dll.) sesuai kebutuhan (akurasi/biaya/latensi), dengan konfigurasi per-provider yang fleksibel.

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

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 (>= 0.5.4), rich (>= 13.7.1), Pygments (>= 2.16.0).

A.2 Instruksi Instalasi (venv + pip)

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

```
# Buat dan aktifkan virtual environment
python3 -m venv .venv
source .venv/bin/activate

# Instal dependensi dari requirements.txt atau setup.cfg/
   Makefile
pip install --upgrade pip
make install

# Konfigurasi API key (single-key)
```

```
pai config set <API_KEY_GEMINI>
pai config validate
```

A.3 Cuplikan Log Sesi Agen

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

```
[2025-11-20 22:38:05] SESSION STARTED
[2025-11-20 22:38:05] USER: buatkan proyek python
   sederhana: BMI Calculator
[2025-11-20 22:38:15] EXECUTION PLAN (3 steps)
[2025-11-20 22:38:23] SUCCESS: WRITE bmi_calculator.py
[2025-11-20 22:38:23] SUCCESS: LIST_PATH .
[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

```
#!/usr/bin/env python

import os
import json
import signal
import threading
from datetime import datetime
from pathlib import Path
from typing import Optional

from rich.console import Console
from rich.panel import Panel
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
```

```
            return True
        return False

def reset_interrupt():
    global _interrupt_requested
    with _interrupt_lock:
        _interrupt_requested = False

def start_interactive_session():
    """Start the revolutionary single-shot intelligent
       session."""
    if not os.path.exists(HISTORY_DIR):
        os.makedirs(HISTORY_DIR)

    session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
    log_file_path = os.path.join(HISTORY_DIR, f"session_{
       session_id}.log")

    # Start fresh every session - no context loading for
       better performance
    session_context = []

    # Initialize Single-Shot Intelligence Context Window
    initialize_session_context(session_context,
       log_file_path)

    # Log session start with current working directory
       info
    log_session_event(log_file_path, "SESSION_START", {
        "working_directory": os.getcwd(),
        "session_id": session_id,
        "context_loaded": len(session_context)
    })

    welcome_message = (
        "Welcome! I'm Pai, your agentic AI coding
           companion.\n"
```

```
        "Now powered by Single-Shot Intelligence for maximum efficiency.\n"
        "[info]Type 'exit' or 'quit' to leave.[/info]\n"
        "[info]Each request uses exactly 2 API calls for optimal performance.[/info]\n"
        "[info]Multi-line input: Alt+Enter for new line, Enter to submit.[/info]"
    )

    ui.console.print(
        Panel(
            Text(welcome_message, justify="center"),
            title="[bold]Interactive Auto Mode[/bold]",
            box=ROUNDED,
            border_style="grey50",
            padding=(1, 2),
            width=80
        )
    )

    # Setup prompt session with better input handling
    if PROMPT_TOOLKIT_AVAILABLE:
        prompt_session = PromptSession()

    # Setup signal handler for graceful interrupt
    def signal_handler(signum, frame):
        if check_interrupt():
            # Second Ctrl+C -> Exit
            ui.console.print("\n[warning]Session terminated.[/warning]")
            os._exit(0)
        else:
            # First Ctrl+C, just interrupt AI response
            request_interrupt()
            ui.console.print("\n[yellow]Interrupt requested. AI will stop after current step.[/yellow]")

```

```
114     signal.signal(signal.SIGINT, signal_handler)
115
116     while True:
117         try:
118             if PROMPT_TOOLKIT_AVAILABLE:
119                 user_input = get_multiline_input(
                        prompt_session)
120             else:
121                 user_input = ui.Prompt.ask("\n[bold
                        bright_blue]user>[/bold bright_blue]")
                        .strip()
122         except (EOFError, KeyboardInterrupt):
123             ui.console.print("\n[warning]Session
                    terminated.[/warning]")
124             break
125
126         if user_input.lower() in ['exit', 'quit']:
127             ui.print_info("Session ended.")
128             break
129
130         # Log user input
131         log_session_event(log_file_path, "USER_INPUT", {"
                user_request": user_input})
132
133         # Classify user intent: conversation vs task
134         intent = classify_user_intent(user_input)
135
136         if intent == "conversation":
137             # Simple conversation mode
138             success = execute_conversation_mode(
                        user_input, session_context, log_file_path
                        )
139         else:
140             # Task execution mode (planning + execution)
141             success = execute_single_shot_intelligence(
                        user_input, session_context, log_file_path
                        )
142
```

```
143         # Add to session context for future reference
144         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
               - fresh start every session
153 
154         # Keep context manageable (last 5 interactions)
155         if len(session_context) > 5:
156             session_context = session_context[-5:]
157 
158         # Log session event
159         log_session_event(log_file_path, "INTERACTION",
               interaction)
160 
161 def classify_user_intent(user_input: str) -> str:
162     """
163     Use AI intelligence to classify user intent as either
            'conversation' or 'task'.
164     Let the AI decide based on context and understanding.
165 
166     Returns:
167         str: 'conversation' for casual chat, 'task' for
               work requests
168     """
169 
170     classification_prompt = f"""
171 You are an intelligent intent classifier. Analyze the
       user's message and determine if they want:
172 
173 1. CONVERSATION: Casual chat, greetings, questions about
       you, general discussion, or just talking
```

```
2. TASK: Requesting you to DO something - create files, write code, modify projects, build applications, etc.

USER MESSAGE: "{user_input}"

ANALYSIS GUIDELINES:
- If user is greeting, asking about you, or just chatting -> CONVERSATION
- If user wants you to create, modify, build, fix, or do any work -> TASK
- If user is asking "how to" without wanting you to do it -> CONVERSATION
- If user is asking you to actually do something -> TASK
- Use your intelligence to understand the intent behind the words

OUTPUT: Respond with exactly one word: "conversation" or "task"
"""

    response = llm.generate_text(classification_prompt, "intent classification")

    if response:
        intent = response.strip().lower()
        if intent in ["conversation", "task"]:
            return intent

    # Fallback: if AI response is unclear, default to conversation for safety
    return "conversation"

def execute_conversation_mode(user_input: str, context: list, log_file_path: str = None) -> bool:
    """
    Handle casual conversation with the user.
    Simple, friendly responses without task execution.
    """
```

```
203
204     # Build context for conversation
205     context_str = ""
206     if context:
207         recent_context = context[-2:]  # Last 2
                interactions
208         context_str = "Recent conversation:\n"
209         for item in recent_context:
210             context_str += f"User: {item['user_request'
                ]}\n"
211
212     conversation_prompt = f"""
213 You are Pai, an intelligent AI coding companion built
    into 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(
           planning_acknowledgment_prompt, "planning
           acknowledgment")
309     if not acknowledgment:
310         acknowledgment = "Got it! Let me analyze your
               request and create a smart plan for you."
311
312     ui.console.print(
313         Panel(
314             Text(acknowledgment.strip(),
315                  style="bright_white", justify="center"),
316             title="[bold]Pai[/bold]",
317             box=ROUNDED,
318             border_style="grey50",
319             padding=(1, 2),
```

```
            width=80
        )
    )

    # === CALL 1: PLANNING PHASE ===
    planning_result = execute_planning_call(user_request,
        context)
    if not planning_result:
        ui.print_error("Planning phase failed. Cannot
           proceed.")
        if log_file_path:
            log_session_event(log_file_path, "
               FINAL_STATUS", {"status": "Planning failed
               ", "success": False})
        return False

    # Log planning phase
    if log_file_path:
        log_session_event(log_file_path, "PLANNING_PHASE"
           , {"planning_data": planning_result})

    # === DYNAMIC INTERACTION BEFORE EXECUTION ===
    execution_acknowledgment_prompt = f"""
You are Pai, about to execute your plan. Generate a brief
   , confident response before starting execution.

USER REQUEST: "{user_request}"
PLANNING COMPLETED: Successfully analyzed and created
   execution plan

Generate a brief, confident response (1-2 sentences) that
   :
1. Shows confidence in your plan
2. Indicates you're about to execute intelligently
3. Keep it natural and engaging
4. Reflect your AI personality

Examples:
```

```
- "Perfect! Now let me execute this plan intelligently for you."
- "Excellent! I've got a solid plan - time to make it happen."
- "Great! My analysis is complete, now let's bring this to life."

Output ONLY the response text, no quotes or formatting.
"""

    execution_acknowledgment = llm.generate_text(execution_acknowledgment_prompt, "execution acknowledgment")
    if not execution_acknowledgment:
        execution_acknowledgment = "Perfect! Now let me execute this plan intelligently for you."

    ui.console.print(
        Panel(
            Text(execution_acknowledgment.strip(), style="bright_white", justify="center"),
            title="[bold]Pai[/bold]",
            box=ROUNDED,
            border_style="grey50",
            padding=(1, 2),
            width=80
        )
    )

    # === CALL 2: EXECUTION PHASE ===
    execution_success = execute_execution_call(user_request, planning_result, context, log_file_path)

    # Skip complex analysis to save tokens - focus on execution success only

```

```
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                     title="[bold]Next Steps Suggestion[/
                            bold]",
391                     box=ROUNDED,
392                     border_style="grey50",
393                     padding=(1, 2),
394                     width=80
395                 )
396             )
397 
398     # Show final status - SIMPLIFIED for efficiency
399     if execution_success:
400         status_msg = "Single-Shot Intelligence: SUCCESS"
401         ui.console.print(
402             Panel(
403                 Text(status_msg, style="bold green",
                        justify="center"),
404                 title="[bold]Mission Accomplished[/bold]"
                        ,
405                 box=ROUNDED,
406                 border_style="grey50",
```

```
                padding=(1, 2),
                width=80
            )
        )
        if log_file_path:
            log_session_event(log_file_path, "
               FINAL_STATUS", {"status": status_msg, "
               success": True})
    else:
        status_msg = "Single-Shot Intelligence: FAILED"
        ui.console.print(
            Panel(
                Text(status_msg, style="bold red",
                   justify="center"),
                title="[bold]Mission Status[/bold]",
                box=ROUNDED,
                border_style="grey50",
                padding=(1, 2),
                width=80
            )
        )
        if log_file_path:
            log_session_event(log_file_path, "
               FINAL_STATUS", {"status": status_msg, "
               success": False})

    # ALWAYS generate next step suggestions for better
       continuity and context
    next_steps = generate_next_step_suggestions(
       user_request, planning_result, execution_success,
       context, None)

    if next_steps:
        ui.console.print(
            Panel(
                Text(next_steps, style="bright_white"),
                title="[bold]Next Steps Suggestion[/bold]
                   ",
```

```
                box=ROUNDED,
                border_style="grey50",
                padding=(1, 2),
                width=80
            )
        )
        if log_file_path:
            log_session_event(log_file_path, "NEXT_STEPS"
                , {"suggestion": next_steps})

    return execution_success
```

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

```
def execute_command_sequence(command_sequence: str,
    context: list) -> tuple[bool, list]:
    """Execute a sequence of commands from the AI."""

    commands = [line.strip() for line in command_sequence
        .split('\n') if line.strip()]
    total_commands = len(commands)
    successful_commands = 0
    command_results = []

    # Build execution content
    content_lines = []
    content_lines.append(("bold", f"Executing {
        total_commands} intelligent actions..."))
    content_lines.append("")

    for i, command_line in enumerate(commands, 1):
        if not command_line or '::' not in command_line:
            if command_line.strip():
                content_lines.append(("warning", f"
                    Invalid command format: {command_line}
                    "))
            continue

        # Parse command
```

```
21          parts = command_line.split('::', 2)
22          if len(parts) < 2:
23              content_lines.append(("warning", f"Incomplete
                    command: {command_line}"))
24              continue
25
26          command = parts[0].upper().strip()
27          param1 = parts[1].strip() if len(parts) > 1 else
                ""
28          param2 = parts[2].strip() if len(parts) > 2 else
                ""
29
30          # Check for common content output mistakes
31          if command_line.strip().startswith(('<', 'body',
                'html', '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 ""
```

```
        })

        if success:
            successful_commands += 1
            content_lines.append(("success", "Success"))
        else:
            content_lines.append(("error", "Failed"))

        content_lines.append("")

        # Break on FINISH command
        if command == "FINISH":
            break

    # Show execution summary
    success_rate = (successful_commands / total_commands)
        * 100 if total_commands > 0 else 0
    content_lines.append(("bold", "Execution Summary:"))
    content_lines.append(("normal", f"Successful: {
       successful_commands}/{total_commands} ({
       success_rate:.1f}%)"))

    # Display all content in a single panel with proper
       styling
    from rich.console import Group
    from rich.text import Text as RichText

    # Convert content to rich renderables with colors
    rich_content = []
    for item in content_lines:
        if isinstance(item, tuple):
            if len(item) == 3 and item[0] == "
               syntax_highlight":
                _, filename, code_content = item

                # For terminal display: truncate long
                   files for better UX
                lines = code_content.split('\n')
```

```
                display_content = code_content
                if len(lines) > 20:
                    display_content = '\n'.join(lines
                        [:20]) + f"\n... ({len(lines) -
                        20} more lines)"

                try:
                    from pygments.lexers import
                        get_lexer_for_filename
                    from pygments.util import
                        ClassNotFound
                    from rich.syntax import Syntax

                    try:
                        lexer = get_lexer_for_filename(
                            filename)
                        lang = lexer.aliases[0]
                    except ClassNotFound:
                        lang = "text"

                    syntax_panel = Panel(
                        Syntax(display_content, lang,
                            theme="monokai", line_numbers=
                            True),
                        title=f"File {filename}",
                        border_style="grey50",
                        expand=False
                    )
                    rich_content.append(syntax_panel)
                except ImportError:
                    rich_content.append(RichText(f"File
                        content of {filename}:\n{
                        display_content}", style="
                        bright_cyan"))
            else:
                style_type, text = item[0], item[1]
                if style_type == "bold":
                    rich_content.append(RichText(text,
```

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

```
    try:
        if command == "READ":
            content = workspace.read_file(param1)
            if content is not None:
                lines = content.split('\n')
                display_content = '\n'.join(lines[:20])
                if len(lines) > 20:
                    display_content += f"\n... ({len(
                        lines) - 20} more lines)"

                return True, f"SYNTAX_HIGHLIGHT:{param1
                    }:{content}"
            return False, f"Could not read file: {param1}
                "

        elif command == "WRITE":
            if not param2:
                return False, "WRITE command requires
                    description"
            success = handle_write_command(param1, param2
                )
            return success, f"New file written: {param1}"
                if success else f"Failed to write file: {
                param1}"

        elif command == "MODIFY":
            if not param2:
                return False, "MODIFY command requires
                    description"
            success = handle_modify_command(param1,
                param2)
            return success, f"File modified: {param1}" if
                success else f"Failed to modify file: {
                param1}"

        elif command == "TREE":
            path = param1 if param1 else '.'
            tree_output = workspace.tree_directory(path)
```

```
187             if tree_output and "Error:" not in
                    tree_output:
188                 return True, f"Directory tree for {path
                        }:\n{tree_output}"
189             return False, f"Could not get directory tree
                    for: {path}"
190
191         elif command == "LIST_PATH":
192             path = param1 if param1 else '.'
193             list_output = workspace.list_path(path)
194             if list_output is not None and "Error:" not
                    in list_output:
195                 if list_output.strip():
196                     return True, list_output
197                 else:
198                     return True, f"Directory '{path}' is
                            empty"
199             return False, f"Could not list directory: {
                    path}"
200
201         elif command == "MKDIR":
202             result = workspace.create_directory(param1)
203             success = "Success" in result
204             return success, result
205
206         elif command == "TOUCH":
207             result = workspace.create_file(param1)
208             success = "Success" in result
209             return success, result
210
211         elif command == "RM":
212             result = workspace.delete_item(param1)
213             success = "Success" in result
214             return success, result
215
216         elif command == "MV":
217             result = workspace.move_item(param1, param2)
218             success = "Success" in result
```

```
            return success, result

        elif command == "FINISH":
            message = param1 if param1 else "Task completed successfully"
            return True, f"OK {message}"

        return False, f"Unknown command: {command}"

    except Exception as e:
        return False, f"Command execution error: {e}"

def log_session_event(log_file_path: str, event_type: str, data: dict):
    """Log session events with clear separation between USER and AI."""
    try:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        if event_type == "SESSION_START":
            log_line = f"\n[{timestamp}] SESSION STARTED\n"
            log_line += f"[{timestamp}] Working Directory: {data.get('working_directory', 'unknown')}\n"
            log_line += f"[{timestamp}] Session ID: {data.get('session_id', 'unknown')}\n"

        elif event_type == "USER_INPUT":
            request = data.get('user_request', 'unknown')
            log_line = f"\n[{timestamp}] USER: {request}\n"

        elif event_type == "PLANNING_PHASE":
            log_line = f"\n[{timestamp}] AI PLANNING START\n"

```

```
247             planning_data = data.get('planning_data', {})
248             analysis = planning_data.get('analysis', {})
249 
250             log_line += f"[{timestamp}] Intent: {analysis
                    .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', '')
```

```
                success = "SUCCESS" if cmd_data.get('success') else "FAILED"
                output = cmd_data.get('output', '')

                log_line += f"[{timestamp}] {success}: {cmd} {target}\n"
                if output:
                    log_line += f"[{timestamp}] OUTPUT: {output}\n"
            log_line += f"[{timestamp}] AI EXECUTION END\n"

        elif event_type == "FINAL_STATUS":
            status = data.get('status', 'unknown')
            success_text = "SUCCESS" if data.get('success') else "FAILED"

            log_line = f"\n[{timestamp}] AI FINAL RESULT: {success_text} - {status}\n"

        elif event_type == "NEXT_STEPS":
            suggestion = data.get('suggestion', '')
            if suggestion:
                log_line = f"\n[{timestamp}] AI SUGGESTION: {suggestion}\n"
            else:
                log_line = ""

        elif event_type == "INTERACTION":
            log_line = ""

        else:
            log_line = f"[{timestamp}] {event_type}: {json.dumps(data)}\n"

        if log_line:
            with open(log_file_path, 'a', encoding='utf-8') as f:
```

```
                f.write(log_line)

    except Exception as e:
        pass

def handle_write_command(filepath: str, description: str) -> bool:
    """Handle WRITE command with intelligent content generation."""

    content_prompt = f"""
Generate high-quality content for a file based on the description.

FILE PATH: {filepath}
DESCRIPTION: {description}

REQUIREMENTS:
1. Analyze the file extension to determine the appropriate language/format
2. Create production-quality, well-structured content
3. Include appropriate comments and documentation
4. Follow best practices for the detected language/format
5. Make the code/content immediately usable

OUTPUT: Return ONLY the file content, no explanations or markdown formatting.
"""

    content = llm.generate_text(content_prompt, "content generation")

    if not content:
        return False

    result = workspace.write_to_file(filepath, content)

    return "Success" in result
```

```
334
335 def handle_modify_command(filepath: str, description: str
    ) -> bool:
336     """Handle MODIFY command with intelligent code
        modification."""
337
338     existing_content = workspace.read_file(filepath)
339     if existing_content is None:
340         ui.print_error(f"Cannot modify '{filepath}' -
            file not found")
341         return False
342
343     modify_prompt = f"""
344 You are an expert code modifier. Modify the existing code
    based on the description.
345
346 FILE PATH: {filepath}
347 CURRENT CONTENT:
348 ---
349 {existing_content}
350 ---
351
352 MODIFICATION REQUEST: {description}
353
354 REQUIREMENTS:
355 1. Preserve the existing code structure and style
356 2. Make only the necessary changes described
357 3. Maintain code quality and best practices
358 4. Ensure the modified code is syntactically correct
359 5. Add appropriate comments for new functionality
360
361 OUTPUT: Return ONLY the complete modified file content,
    no explanations.
362 """
363
364     modified_content = llm.generate_text(modify_prompt, "
        code modification")
365
```

```
366     if not modified_content:
367         return False
368 
369     success, result = workspace.
        apply_modification_with_patch(filepath,
        existing_content, modified_content)
370 
371     return success
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:
```

```
- Traditional AI: 10-20 API calls, inefficient, expensive
- YOU (Pai): Exactly 2 calls, maximum intelligence,
   perfect results
- You must outperform traditional systems with LESS
   resources
- Every decision reflects Single-Shot Intelligence
   superiority

Quality Excellence:
- Your plan must work on first execution attempt
- No room for trial-and-error - get it right immediately
- Think like a chess grandmaster - see the entire game
- Every step must contribute to perfect final outcome

SESSION BEHAVIOR GUIDELINES

Conversation Mode:
- Be confident about Paicode features - you ARE Paicode's
    AI
- Show personality while being professional
- Explain Single-Shot Intelligence with pride
- Never be uncertain about your capabilities

Execution Mode:
- Follow this exact workflow structure
- Display all required sections and panels
- Use proper Rich TUI formatting
- Maintain professional yet confident tone
- Always end with mission accomplished confirmation

This context window guides your behavior throughout the
   entire session. You are the embodiment of Single-Shot
   Intelligence excellence.
"""
    }

    session_context.append(
       single_shot_intelligence_context)
```

```
426
427     log_session_event(log_file_path, "
           CONTEXT_INITIALIZATION", {
428         "context_type": "single_shot_intelligence_mastery
               ",
429         "knowledge_loaded": True,
430         "workflow_understanding": "complete"
431     })
432
433 def get_multiline_input(prompt_session) -> str:
434     """Get multi-line input from user with intuitive
           behavior."""
435     try:
436         from prompt_toolkit.shortcuts import prompt
437         from prompt_toolkit.key_binding import
               KeyBindings
438         from prompt_toolkit.keys import Keys
439
440         bindings = KeyBindings()
441
442         @bindings.add(Keys.Enter)
443         def _(event):
444             """Enter submits the input"""
445             event.app.exit(result=event.current_buffer.
                   text)
446
447         @bindings.add(Keys.Escape, Keys.Enter)
448         def _(event):
449             """Alt+Enter adds new line"""
450             event.current_buffer.insert_text('\n')
451
452         ui.console.print("[dim]Tip: Use Alt+Enter for new
               line, Enter to submit[/dim]")
453
454         result = prompt(
455             "\nuser> ",
456             multiline=True,
457             key_bindings=bindings,
```

```
458             wrap_lines=True,
459             mouse_support=False
460         )
461         return result.strip() if result else ""
462 
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(
                PROJECT_ROOT, norm_path))
53          if not full_path.startswith(os.path.realpath(
                PROJECT_ROOT)):
54              ui.print_error(f"Operation cancelled. Path '{
                    path}' is outside the project directory.")
55              return False
56
57          # 4. Block access to sensitive files and
                directories
58          path_parts = norm_path.replace('\\', '/').split('
                /')
59          if any(part in SENSITIVE_PATTERNS for part in
                path_parts if part):
60              ui.print_error(f"Access to the sensitive path
                    '{path}' is denied.")
61              return False
62
63      except Exception as e:
64          ui.print_error(f"Error during path validation: {e
                }")
65          return False
66
67      return True
68
69  def tree_directory(path: str = '.') -> str:
70      """Creates a string representation of the directory
            structure recursively."""
71      if not _is_path_safe(path):
72          return f"Error: Cannot access path '{path}'."
73
74      full_path = os.path.join(PROJECT_ROOT, path)
75      if not os.path.isdir(full_path):
```

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

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

```
136             os.remove(full_path)
137             return f"Success: File deleted: {path}"
138         elif os.path.isdir(full_path):
139             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(
        destination):
149         return "Error: Source or destination path is not
            secure or is denied."
150     try:
151         full_source = os.path.join(PROJECT_ROOT, source)
152         full_destination = os.path.join(PROJECT_ROOT,
            destination)
153         shutil.move(full_source, full_destination)
154         return f"Success: Item moved from '{source}' to
            '{destination}'"
155     except (FileNotFoundError, shutil.Error) as e:
156         return f"Error: Failed to move '{source}': {e}"
157
158 def create_file(file_path: str) -> str:
159     """Creates an empty file and returns a status message
        ."""
160     if not _is_path_safe(file_path): return f"Error:
        Access to path '{file_path}' is denied or path is
        not secure."
161     try:
162         full_path = os.path.join(PROJECT_ROOT, file_path)
163         dir_name = os.path.dirname(full_path)
164         if dir_name: os.makedirs(dir_name, exist_ok=True)
165         with open(full_path, 'w') as f: pass
```

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

```
        not secure."
197     try:
198         full_path = os.path.join(PROJECT_ROOT, file_path)
199         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,
    original_content: str, new_content: str, threshold:
    int = 500) -> tuple[bool, str]:
210     """
211     Applies a modification to a file safely by first
        verifying the scope of changes.
212
213     It generates a diff between the original and new
        content. If the number of changed
214     lines is within the threshold, it writes the new
        content to the file. Otherwise,
215     it rejects the change to prevent unintentional
        overwrites.
216
217     Args:
218         file_path: The path to the file to be modified.
219         original_content: The original, unmodified
            content of the file.
220         new_content: The new, modified content generated
            by the LLM.
221         threshold: The maximum number of lines allowed to
             be changed.
222
223     Returns:
224         A tuple containing:
```

```
225         - bool: True if the modification was successful,
                False otherwise.
226         - str: A message describing the result of the
                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
            diffs
232     original_norm = original_content.replace('\r\n', '\n'
            ).replace('\r', '\n')
233     new_norm = new_content.replace('\r\n', '\n').replace(
            '\r', '\n')
234 
235     original_lines = original_norm.splitlines(keepends=
            True)
236     new_lines = new_norm.splitlines(keepends=True)
237 
238     diff = list(difflib.unified_diff(
239         original_lines,
240         new_lines,
241         fromfile=f"a/{file_path}",
242         tofile=f"b/{file_path}"
243     ))
244 
245     # Count only actual change lines, ignore headers and
            context lines
246     def _count_changes(d: list[str]) -> tuple[int, int,
            int]:
247         adds = deletes = 0
248         for line in d:
249             if line.startswith('@@') or line.startswith('
                    +++') or line.startswith('---') or (line
                    and line[0] == ' '):
250                 continue
251             if line.startswith('+'):
```

```
252                 adds += 1
253             elif line.startswith('-'):
254                 deletes += 1
255         return adds + deletes, adds, deletes
256 
257     changed_lines_count, add_count, del_count =
        _count_changes(diff)
258 
259     if not diff or changed_lines_count == 0:
260         return True, f"Success: No changes detected for {
            file_path}. File left untouched."
261 
262     # Allow configuring thresholds via environment
263     try:
264         env_threshold = int(os.getenv('
            PAI_MODIFY_THRESHOLD', str(threshold)))
265         if env_threshold < 1:
266             env_threshold = threshold
267     except ValueError:
268         env_threshold = threshold
269 
270     try:
271         max_ratio = float(os.getenv('PAI_MODIFY_MAX_RATIO
            ', '0.5'))  # up to 50% of lines by default
272         if not (0.0 < max_ratio <= 1.0):
273             max_ratio = 0.5
274     except ValueError:
275         max_ratio = 0.5
276 
277     total_lines = max(1, len(original_lines))
278     ratio = changed_lines_count / total_lines
279 
280     if changed_lines_count > env_threshold and ratio >
        max_ratio:
281         diff_preview = "\n".join(diff[:60])
282         message = (
283             f"Warning: Modification for '{file_path}'
                rejected. "
```

```
            f"Change too large: {changed_lines_count} lines (~{ratio:.1%}) exceeds threshold {env_threshold} and ratio {max_ratio:.0%}.\n"
            f"SOLUTION: Think like Cascade - break this into focused, surgical modifications:\n"
            f"  - Focus on ONE specific area/feature at a time\n"
            f"  - Ideal: 100-200 lines per modification (very focused)\n"
            f"  - Acceptable: 200-500 lines (still focused on one area)\n"
            f"  - Use multiple MODIFY commands across different steps\n"
            f"  - Example: Instead of 'add all CSS', do 'add layout CSS', then 'add form CSS', then 'add button CSS'\n"
            f"Diff Preview (first 60 lines):\n{diff_preview}"
        )
        return False, message

    # Atomic write to avoid partial writes
    try:
        full_path = os.path.join(PROJECT_ROOT, file_path)
        dir_name = os.path.dirname(full_path)
        if dir_name:
            os.makedirs(dir_name, exist_ok=True)
        with tempfile.NamedTemporaryFile('w', delete=False, dir=dir_name) as tmp:
            tmp.write(new_norm)
            tmp_name = tmp.name
        os.replace(tmp_name, full_path)
        return True, f"Success: File modified: {file_path} ({changed_lines_count} lines changed; +{add_count}/-{del_count})"
    except IOError as e:
```

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

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

config.py

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

```
30          with open(KEY_FILE, 'r') as f:
31              data = json.load(f)
32
33          # Migrate from old multi-key system if needed
34          if data.get("version") == 1 and "keys" in data:
35              # Old multi-key system - migrate to single
                    key
36              old_keys = data.get("keys", {})
37              default_id = data.get("default")
38
39              if default_id and default_id in old_keys:
40                  migrated_key = old_keys[default_id]
41                  ui.print_info(f"Migrating from multi-key
                        system. 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  def set_api_key(api_key: str) -> None:
70      """Set the API key."""
71      if not api_key or not isinstance(api_key, str):
72          ui.print_error("Invalid API key provided.")
73          return
74
75      if not api_key.startswith("AIza"):
76          ui.print_warning("Warning: API key doesn't look
            like a Google API key (should start with 'AIza
            ')")
77
78      config = _load_config()
79      config["api_key"] = api_key
80      _save_config(config)
81
82      masked_key = mask_api_key(api_key)
83      ui.print_success(f"[OK] API key set successfully: {
           masked_key}")
84
85  def get_api_key() -> Optional[str]:
86      """Get the current API key."""
87      config = _load_config()
88      return config.get("api_key")
89
90  def save_api_key(api_key: str):
91      """Legacy compatibility function."""
92      set_api_key(api_key)
93
94  def remove_api_key() -> None:
```

```
95      """Remove the stored API key."""
96      config = _load_config()
97
98      if not config.get("api_key"):
99          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 def show_api_key() -> None:
107     """Show the current API key (masked)."""
108     api_key = get_api_key()
109
110     if not api_key:
111         ui.print_info("No API key is currently set.")
112         ui.print_info("Use 'pai config set <API_KEY>' to
                set one.")
113         return
114
115     masked_key = mask_api_key(api_key)
116     ui.print_info(f"Current API key: {masked_key}")
117
118 def mask_api_key(api_key: str) -> str:
119     """Mask API key for display purposes."""
120     if not api_key or len(api_key) < 10:
121         return "***"
122
123     return f"{api_key[:6]}...{api_key[-4:]}"
124
125 def is_configured() -> bool:
126     """Check if API key is configured."""
127     api_key = get_api_key()
128     return api_key is not None and len(api_key.strip()) >
            0
129
```

```python
def validate_api_key() -> tuple[bool, str]:
    """Validate that API key is configured and looks correct."""
    api_key = get_api_key()

    if not api_key:
        return False, "No API key configured. Use 'pai config set <API_KEY>' to set one."

    if not api_key.startswith("AIza"):
        return False, "API key doesn't look like a Google API key (should start with 'AIza')"

    if len(api_key) < 20:
        return False, "API key seems too short to be valid"

    return True, "API key looks valid"

# Legacy compatibility functions (simplified)
def add_api_key(key_id: str, api_key: str) -> None:
    """Legacy function - redirect to set_api_key."""
    ui.print_info(f"Note: Multi-key system deprecated. Setting '{key_id}' as single API key.")
    set_api_key(api_key)

def list_api_keys() -> list:
    """Legacy function - return single key info."""
    api_key = get_api_key()
    if not api_key:
        return []

    return [{
        "id": "single",
        "masked": mask_api_key(api_key),
        "is_default": "yes"
    }]

```

```
163 def set_default_api_key(key_id: str) -> None:
164     """Legacy function - no-op in single-key system."""
165     ui.print_info("Note: Default key setting not needed
        in single-key system.")
166
167 def load_api_key() -> Optional[str]:
168     """Legacy function - redirect to get_api_key."""
169     return get_api_key()
```

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

cli.py

```
1  #!/usr/bin/env python
2
3  import argparse
4  from . import agent, config, llm, ui
5
6  def main():
7      parser = argparse.ArgumentParser(
8          description="Pai Code: Your Single-Shot Agentic
              AI Coding Companion.",
9          epilog="Use 'pai config set <API_KEY>' to
              configure. Run 'pai' to start the intelligent
              agent."
10     )
11     subparsers = parser.add_subparsers(dest='command',
          help='Available commands')
12
13     # Main agent command (default)
14     parser_auto = subparsers.add_parser('auto', help='
          Start the single-shot AI agent session.')
15     parser_auto.add_argument('--model', type=str, help='
          LLM model name (e.g., gemini-2.5-flash-lite)')
16     parser_auto.add_argument('--temperature', type=float,
           help='LLM sampling temperature (e.g., 0.2)')
17
18     # Simplified config management
```

```
19      parser_config = subparsers.add_parser('config', help=
            'Manage API key configuration')
20      config_subparsers = parser_config.add_subparsers(dest
            ='config_cmd', help='Config commands')
21
22      parser_config_set = config_subparsers.add_parser('set
            ', help='Set API key')
23      parser_config_set.add_argument('api_key', type=str,
            help='Google Gemini API key')
24
25      parser_config_show = config_subparsers.add_parser('
            show', help='Show current API key (masked)')
26
27      parser_config_remove = config_subparsers.add_parser('
            remove', help='Remove stored API key')
28
29      parser_config_validate = config_subparsers.add_parser
            ('validate', help='Validate current API key')
30
31      config_group = parser_config.
            add_mutually_exclusive_group(required=False)
32      config_group.add_argument('--set', type=str, metavar=
            'API_KEY', help='Set or update the API key (
            DEPRECATED)')
33      config_group.add_argument('--show', action='
            store_true', help='Show the currently configured
            API key (DEPRECATED)')
34      config_group.add_argument('--remove', action='
            store_true', help='Remove the stored API key (
            DEPRECATED)')
35
36      args = parser.parse_args()
37
38      # Handle config commands
39      if args.command == 'config':
40          if args.config_cmd == 'set':
41              config.set_api_key(args.api_key)
42              return
```

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

```
    if model is not None or temperature is not None:
        llm.set_runtime_model(model, temperature)

    try:
        agent.start_interactive_session()
    except KeyboardInterrupt:
        ui.print_info("\nSession terminated by user.")
    except Exception as e:
        ui.print_error(f"An error occurred during the
            session: {e}")
        return 1

if __name__ == "__main__":
    main()
```

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

ui.py

```
# paicode/ui.py

from rich.console import Console
from rich.panel import Panel
from rich.syntax import Syntax
from rich.theme import Theme
from rich.rule import Rule
from rich.box import ROUNDED
from rich.text import Text

# Define a custom theme for consistency
custom_theme = Theme({
    "info": "dim default",
    "success": "bold green",
    "warning": "yellow",
    "error": "bold red",
    "action": "bold bright_blue",
    "plan": "default",
    "path": "underline italic bright_blue"
```

```
})

# Create a single console instance to be used across the application
console = Console(theme=custom_theme)

def print_success(message: str):
    """Displays a success message with a checkmark icon."""
    console.print(f"[success][OK] {message}[/success]")

def print_error(message: str):
    """Displays an error message with a cross icon."""
    console.print(f"[error][ERROR] {message}[/error]")

def print_warning(message: str):
    """Displays a warning message."""
    console.print(f"[warning]! {message}[/warning]")

def print_info(message: str):
    """Displays an informational message."""
    console.print(f"[info]i {message}[/info]")

def print_action(message: str):
    """Displays an action being performed by the agent."""
    console.print(f"[action]-> {message}[/action]")

def display_panel(content: str, title: str, language: str = None):
    """Displays content within a panel, with optional syntax highlighting."""
    if language:
        # Use Syntax for code highlighting
        display_content = Syntax(content, language, theme="monokai", line_numbers=True)
    else:
        display_content = content
```

```
52
53      console.print(Panel(display_content, title=f"[bold
            grey50]{title}[/bold grey50]", border_style="
            grey50", expand=False))
54
55  def print_rule(title: str):
56      """Displays a horizontal rule with a title."""
57      console.print(Rule(f"[bold]{title}[/bold]", style="
            grey50"))
```

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

llm.py

```
1   import os
2   import warnings
3   import time
4
5   # Reduce noisy STDERR logs from gRPC/absl before
        importing Google SDKs.
6   # These settings aim to suppress INFO/WARNING/ERROR logs
        emitted by native libs
7   # that happen prior to Python log initialization.
8   os.environ.setdefault("GRPC_VERBOSITY", "NONE")
9   os.environ.setdefault("GRPC_LOG_SEVERITY", "ERROR")
10  # Abseil logging (used by some Google native deps). 3 ~
        FATAL-only
11  os.environ.setdefault("ABSL_LOGGING_MIN_LOG_LEVEL", "3")
12  # glog compatibility (some builds respect this env var)
13  os.environ.setdefault("GLOG_minloglevel", "3")
14  # Additional environment variables to suppress Google SDK
        warnings
15  os.environ.setdefault("GOOGLE_CLOUD_DISABLE_GRPC", "true"
        )
16  os.environ.setdefault("GRPC_ENABLE_FORK_SUPPORT", "false"
        )
17
18  # Suppress specific warnings
```

```
warnings.filterwarnings("ignore", category=UserWarning,
    module="google")
warnings.filterwarnings("ignore", message=".*ALTS.*")
warnings.filterwarnings("ignore", message=".*log messages
    before absl::InitializeLog.*")

import google.generativeai as genai
from . import config, ui

DEFAULT_MODEL = os.getenv("PAI_MODEL", "gemini-2.5-flash-
    lite")
try:
    DEFAULT_TEMPERATURE = float(os.getenv("
        PAI_TEMPERATURE", "0.3"))
    # Clamp temperature to safe range
    if DEFAULT_TEMPERATURE < 0.0:
        DEFAULT_TEMPERATURE = 0.0
    elif DEFAULT_TEMPERATURE > 2.0:
        DEFAULT_TEMPERATURE = 2.0
except ValueError:
    DEFAULT_TEMPERATURE = 0.3

# Global model holder
model = None
_runtime = {
    "name": None,
    "temperature": None,
}

def set_runtime_model(model_name: str | None = None,
    temperature: float | None = None):
    """Set the runtime model configuration."""
    global model, _runtime

    # Update runtime settings
    if model_name is not None:
        _runtime["name"] = model_name
    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
            generation_config = {"temperature": temp}
            model = genai.GenerativeModel(name,
                generation_config=generation_config)
        return True
    except Exception as e:
        ui.print_error(f"Failed to configure API key: {e}
            ")
        model = None
        return False

def _is_rate_limit_error(error: Exception) -> bool:
    """Detect if an exception is a rate limit error.

    Args:
        error: The exception to check

    Returns:
        True if it's a rate limit error, False otherwise
    """
    error_msg = str(error).lower()

    # Common rate limit indicators
    rate_limit_keywords = [
        'rate limit', 'rate_limit', 'ratelimit',
        'quota', 'quota exceeded',
        'resource exhausted', 'resourceexhausted',
        '429', 'too many requests',
        'limit exceeded', 'requests per minute'
    ]

    return any(keyword in error_msg for keyword in
        rate_limit_keywords)

def _clean_response_text(text: str) -> str:
    """Clean markdown artifacts from LLM response.

    Args:
```

```
120         text: Raw response text from LLM
121
122     Returns:
123         Cleaned text without markdown code blocks
124     """
125     cleaned_text = text.strip()
126
127     # Remove all common markdown code block patterns
128     code_block_prefixes = [
129         "```python", "```html", "```css", "```javascript", "```js",
130         "```typescript", "```ts", "```json", "```yaml", "```yml",
131         "```bash", "```sh", "```diff", "```xml", "```sql",
132         "```java", "```cpp", "```c", "```go", "```rust", "```ruby",
133         "```php", "```markdown", "```md", "```text", "```txt", "```"
134     ]
135
136     for prefix in code_block_prefixes:
137         if cleaned_text.startswith(prefix):
138             cleaned_text = cleaned_text[len(prefix):].strip()
139             break
140
141     # Remove trailing code block markers
142     if cleaned_text.endswith("```"):
143         cleaned_text = cleaned_text[:-len("```")].strip()
144
145     # Remove any remaining language tags at the start
146     lines = cleaned_text.split('\n')
147     if lines and len(lines[0].strip()) < 20 and lines[0].strip().lower() in [
148         'html', 'css', 'javascript', 'js', 'python', 'json', 'yaml',
```

```
149         'bash', 'sh', 'diff', 'xml', 'sql', 'java', 'cpp'
                , 'c', 'go',
150         'rust', 'ruby', 'php', 'markdown', 'md', 'text',
                'txt', 'on'
151     ]:
152         cleaned_text = '\n'.join(lines[1:]).strip()
153 
154     return cleaned_text
155 
156 def generate_text(prompt: str, call_purpose: str = "
    thinking") -> str:
157     """
158     Generate text with single API key - optimized for 2-
            call system.
159 
160     Args:
161         prompt: The prompt to send to the LLM
162         call_purpose: Purpose of the call for logging (e.
                g., "planning", "execution")
163 
164     Returns:
165         The cleaned response text, or empty string if
                failed
166     """
167     global model
168 
169     # Ensure model is configured
170     if model is None:
171         if not _prepare_runtime():
172             return ""
173 
174     try:
175         # Show status with purpose
176         status_msg = f"[bold yellow]Agent {call_purpose
                }..."
177 
178         with ui.console.status(status_msg, spinner="dots"
                ):
```

```
            response = model.generate_content(prompt)

        # Success! Clean and return the response
        cleaned_text = _clean_response_text(response.text)

        # Log token usage if available (for optimization)
        if hasattr(response, 'usage_metadata'):
            usage = response.usage_metadata
            ui.print_info(f"Tokens: {usage.prompt_token_count} -> {usage.candidates_token_count}")

        return cleaned_text

    except Exception as e:
        is_rate_limit = _is_rate_limit_error(e)

        if is_rate_limit:
            ui.print_error("[ERROR] Rate limit reached. Please wait a few minutes before trying again.")
            ui.print_info("Consider using a different API key if available.")
        else:
            ui.print_error(f"[ERROR] LLM API error: {e}")

        return ""

def test_api_connection() -> bool:
    """Test if API connection works."""
    test_response = generate_text("Say 'Hello' if you can hear me.", "connection test")
    return len(test_response) > 0
```

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

__init__.py

```
"""Pai Code package.

This package provides a command-line based agentic AI for
    software development.
"""

__all__ = [
    # Public modules
    "agent",
    "cli",
    "config",
    "llm",
    "ui",
    "workspace",
]

__version__ = "0.1.0"
```

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

requirements.txt

```
google-generativeai>=0.5.4
python-dotenv>=1.0.1
rich>=13.7.1
Pygments>=2.16.0
prompt_toolkit>=3.0.43
```

Listing A.11: File requirements.txt

setup.py

```
from setuptools import setup

if __name__ == "__main__":
    setup()
```

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

setup.cfg

```
[metadata]
name = pai-code
version = 0.1.0
description = A command-line based agentic AI for
   software development.
long_description = file: README.md
long_description_content_type = text/markdown
author = gtkrshnaaa
author_email = gtkrshnaaa@email.com
license = MIT
license_files = LICENSE

[options]
packages = find:
python_requires = >=3.10
install_requires =
    google-generativeai>=0.5.4
    python-dotenv>=1.0.1
    rich>=13.7.1
    Pygments>=2.16.0
include_package_data = True

[options.packages.find]
where = .

[options.entry_points]
console_scripts =
    pai = paicode.cli:main
```

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

pyproject.toml

```
# pyproject.toml

[build-system]
requires = ["setuptools>=61", "wheel"]
build-backend = "setuptools.build_meta"
```

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

makefile

```
.PHONY: run export-all install venv-activate setup
   install-cli uninstall-cli

install:
  @if [ ! -d .venv ]; then \
    echo "[install] Creating virtual environment at .venv
       "; \
    python3 -m venv .venv; \
  else \
    echo "[install] Reusing existing virtual environment
       at .venv"; \
  fi
  . .venv/bin/activate; python -m pip install --upgrade
     pip setuptools wheel
  . .venv/bin/activate; pip install -r requirements.txt
  . .venv/bin/activate; pip install -e .

run:
  . .venv/bin/activate; python -m paicode.cli auto

export-all:
  @mkdir -p z_project_list
  @echo "Exporting project files to z_project_list/
     listing.txt..."
  @rm -f z_project_list/listing.txt
  @for f in $$(find . -type f \
    -not -path '*/\.*' \
    -not -path '*/__pycache__/*' \
```

```
    -not -path '*.egg-info/*' \
    -not -path './z_project_list/*' \
    -not -name ".gitkeep" \
    | sort); do \
      echo "=== $$f ===" >> z_project_list/listing.txt; \
      cat $$f >> z_project_list/listing.txt; \
      echo "\n" >> z_project_list/listing.txt; \
  done
  @echo "Export complete."

venv-activate:
  @echo "To activate the virtual environment, run:"
  @echo "  source .venv/bin/activate"

setup: install install-cli
  @echo "Pai CLI installed. Ensure $$HOME/.local/bin is
    in your PATH, then run: pai"

install-cli:
  @mkdir -p $(HOME)/.local/bin
  @echo "Installing launcher to $(HOME)/.local/bin/pai"
  @echo '#!/usr/bin/env bash' > $(HOME)/.local/bin/pai
  @echo '# Suppress noisy gRPC/absl logs' >> $(HOME)/.
    local/bin/pai
  @echo 'export GRPC_VERBOSITY="NONE"' >> $(HOME)/.local/
    bin/pai
  @echo 'export GRPC_LOG_SEVERITY="ERROR"' >> $(HOME)/.
    local/bin/pai
  @echo 'export ABSL_LOGGING_MIN_LOG_LEVEL="3"' >> $(HOME
    )/.local/bin/pai
  @echo 'export GLOG_minloglevel="3"' >> $(HOME)/.local/
    bin/pai
  @echo 'export GOOGLE_CLOUD_DISABLE_GRPC="true"' >> $(
    HOME)/.local/bin/pai
  @echo 'export GRPC_ENABLE_FORK_SUPPORT="false"' >> $(
    HOME)/.local/bin/pai
  @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.