

**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
225410001

INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA
Angkatan 2022

Lembar Pengesahan

Yang bertanda tangan di bawah ini menyatakan bahwa skripsi dengan judul:

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

oleh:

Nama : I PUTU GEDE GILANG TEJA KRISHNA

NIM : 225410001

Prodi : INFORMATIKA

Fakultas : FAKULTAS TEKNOLOGI INFORMASI

Universitas : UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA

Telah **disetujui dan disahkan** pada:

Hari/Tanggal :

Tempat :

Dosen Pembimbing

Pembimbing I,

Pembimbing II,

(.....)

(.....)

NIP/NIDN:

NIP/NIDN:

Dewan Penguji

Ketua Penguji:

(.....)

Sekretaris: (.....)

Anggota Penguji I: (.....)

Anggota Penguji II: (.....)

Pengesahan Pejabat Fakultas

Ketua Program Studi,

Dekan,

(.....)

(.....)

NIP/NIDN:

NIP/NIDN:

Catatan: Tempelkan stempel basah sesuai ketentuan masing-masing unit.

Pernyataan Keaslian

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

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

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

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

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Kata Pengantar

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

Ucapan terima kasih disampaikan kepada:

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

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

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Ucapan Terima Kasih

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

Secara khusus, ucapan terima kasih ditujukan kepada:

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

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

Yogyakarta,

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Abstrak

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

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

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

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

Abstract

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

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

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

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

Daftar Singkatan

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

Daftar Simbol

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

Daftar Istilah

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

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

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

Daftar Isi

Lembar Pengesahan	i
Pernyataan Keaslian	iii
Kata Pengantar	iv
Ucapan Terima Kasih	v
Abstrak	vi
Abstract	vii
Daftar Singkatan	viii
Daftar Simbol	ix
Daftar Istilah	x
1 Pendahuluan	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	2
1.4 Tujuan Penelitian	3
1.5 Manfaat Penelitian	3
2 Tinjauan Pustaka	4
2.1 Teori Dasar	4
2.1.1 Command Line Interface (CLI)	4
2.1.2 AI Agent	4
2.1.3 Large Language Model (LLM)	5
2.1.4 Perbedaan LLM dan Agen AI	5
2.1.5 Arsitektur dan Kebijakan Data	5
2.1.6 Manajemen Dependensi dengan pip dan Virtual Environment	6
2.1.7 Antarmuka Terminal dengan rich dan prompt_toolkit	6

2.2	Penelitian Terkait	6
2.3	Posisi Penelitian	7
2.4	Rencana Gambar Tinjauan Pustaka	7
3	Metodologi Penelitian	9
3.1	Metode Pengembangan	9
3.2	Arsitektur Sistem	9
3.3	Visualisasi Metodologi	10
3.4	Alat dan Lingkungan	13
3.5	Prosedur Penelitian	13
4	Implementasi dan Hasil	15
4.1	Implementasi Paicode	15
4.1.1	Instalasi	15
4.1.2	Konfigurasi API Key	15
4.1.3	Menjalankan Agen	16
4.2	Alur Interaksi dengan 8-Step Workflow	16
4.3	Rencana Gambar Implementasi	17
4.4	Tabel Skenario Pengujian	19
4.5	Tabel Metrik Evaluasi	19
4.6	Tabel Konfigurasi Lingkungan	19
4.7	Contoh Sesi	20
4.8	Evaluasi	20
5	Kesimpulan dan Saran	22
5.1	Kesimpulan	22
5.2	Saran	23
A	Lampiran A	25
A.1	Konfigurasi Lingkungan	25
A.2	Contoh Sesi Terminal	25
A.3	Hasil Pengukuran Rinci	25

Daftar Gambar

2.1	Konsep arsitektur agentic AI di lingkungan CLI dengan inferensi LLM melalui API.	7
2.2	Model interaksi <i>stateful</i> dan <i>feedback loop</i> pada sesi agen.	8
2.3	Ilustrasi komparasi konseptual antara pendekatan ekstensi editor, layanan daring, dan CLI dengan integrasi LLM via API.	8
4.1	Tampilan awal sesi agen di terminal.	17
4.2	Output perintah <code>TREE</code> untuk observasi struktur proyek.	18
4.3	Output perintah <code>LIST_PATH</code> untuk daftar path mesin-baca.	18
4.4	Panel pembacaan berkas dengan penyorotan sintaks.	18
4.5	Contoh hasil perintah <code>MODIFY</code> dengan batasan perubahan berbasis <i>diff</i>	18
4.6	Diagram alur evaluasi dan metrik yang dikumpulkan.	18
4.7	Contoh visualisasi hasil awal untuk metrik efisiensi.	18
A.1	Contoh sesi agen pada terminal.	25

Daftar Tabel

3.1	Modul dan Dependensi Komponen Paicode	10
3.2	Urutan Interaksi Sesi Agen dengan Single-Shot Intelligence	11
3.3	Rangkuman Validasi Keamanan <i>Path</i>	12
4.1	Skenario Pengujian Paicode	19
4.2	Metrik Evaluasi dan Definisi Operasional	19
4.3	Konfigurasi Lingkungan Uji	20

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, 4, 1]. Dengan demikian, aspek privasi/kerahasiaan kode **bergantung pada kebijakan penyedia API**, sementara pengamanan di sisi lokal difokuskan pada kebijakan *path security* (keamanan *path*) dan pembatasan perubahan berbasis *diff*.

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

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

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah yang diajukan adalah sebagai berikut:

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

1.3 Batasan Masalah

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

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

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

1.4 Tujuan Penelitian

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

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

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

- **Akademis:** menyediakan studi kasus dan arsitektur rujukan untuk pengembangan agen AI berbasis CLI dengan integrasi LLM melalui API, serta memperkaya literatur mengenai integrasi LLM dalam alur kerja rekayasa perangkat lunak.
- **Praktis:** menghadirkan alat bantu yang *privacy-aware* dan mudah diintegrasikan dengan berbagai IDE karena beroperasi langsung pada berkas proyek di ruang kerja (workspace); memfasilitasi pembuatan struktur proyek, pembacaan, dan modifikasi kode secara cepat dan terarah.

BAB 2

Tinjauan Pustaka

2.1 Teori Dasar

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

2.1.1 Command Line Interface (CLI)

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

2.1.2 AI Agent

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

Pada implementasi Paicode, agen menggunakan arsitektur *Single-Shot Intelligence* yang terdiri dari beberapa komponen: (1) klasifikasi intensi untuk membedakan percakapan dan tugas, (2) *acknowledgment* dinamis untuk konfirmasi pemahaman, (3) fase perencanaan dengan analisis mendalam dan perencanaan komprehensif dalam format

JSON, (4) fase eksekusi adaptif yang dapat berjalan dalam 1-3 subfase berdasarkan kompleksitas tugas, dan (5) saran langkah berikutnya. Sistem ini mengoptimalkan efisiensi dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.

2.1.3 Large Language Model (LLM)

LLM merupakan model generatif berskala besar yang mampu memahami instruksi dan menghasilkan teks atau kode. Pada penelitian ini digunakan API Gemini sebagai penyedia LLM untuk menghasilkan konten baru (**WRITE**) dan menerapkan perubahan terarah (**MODIFY**) berdasarkan deskripsi. Prinsip kehati-hatian diterapkan dengan mekanisme pembatasan perubahan berbasis *diff* sehingga modifikasi tidak berskala besar tanpa kontrol [2, 4, 1, 6, 3, 5, 7].

2.1.4 Perbedaan LLM dan Agen AI

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

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

2.1.5 Arsitektur dan Kebijakan Data

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

2.1.6 Manajemen Dependensi dengan pip dan Virtual Environment

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

2.1.7 Antarmuka Terminal dengan rich dan prompt_toolkit

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

2.2 Penelitian Terkait

Berbagai alat bantu pengembangan perangkat lunak berbasis LLM telah diusulkan dan dikomersialisasi, antara lain asisten kode terintegrasi editor, agen otomatis untuk *refactoring*, serta sistem tanya-jawab dokumentasi. Umumnya solusi tersebut beroperasi sebagai ekstensi editor atau layanan daring, sehingga kuat pada integrasi IDE namun bergantung pada antarmuka tertentu dan memproses konteks di luar mesin pengguna.

Sebaliknya, Paicode menempatkan agen di lingkungan CLI dan beroperasi langsung pada **berkas proyek di workspace**, sementara inferensi dilakukan oleh LLM eksternal melalui API. Perintah agen disederhanakan ke dalam himpunan tindakan yang eksplisit (MKDIR, TOUCH, READ, WRITE, MODIFY, RM, MV, TREE, LIST_PATH, FINISH) dengan *policy path security*. Penelitian terkait menunjukkan bahwa interaksi *stateful* berbasis rencana aksi meningkatkan kualitas hasil pada tugas-tugas rekayasa perangkat lunak yang iteratif, sementara *guardrail* sederhana (seperti pembatasan *diff*) dapat menekan risiko penimpaan berkas secara tidak disengaja.

2.3 Posisi Penelitian

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

- **CLI lokal dengan integrasi LLM via API:** agen berjalan di terminal, tindakan langsung tercermin pada **berkas proyek di workspace**; sementara inferensi dilakukan oleh LLM eksternal sehingga kebijakan data mengikuti penyedia API.
- **Arsitektur Single-Shot Intelligence:** alur kerja efisien yang mengoptimalkan penggunaan API dengan tepat 2 panggilan (perencanaan dan eksekusi), menggantikan pendekatan tradisional yang memerlukan 10-20 panggilan API.
- **Manajemen API key tunggal:** sistem manajemen API key yang disederhanakan dengan migrasi otomatis dari sistem multi-key untuk kemudahan penggunaan.
- **Keamanan berkas:** kebijakan pelarangan akses *path* sensitif dan validasi *path* mencegah *path traversal* dan operasi berisiko pada direktori seperti `.git`, `venv`, dan `.env`.
- **Modifikasi terarah berbasis diff:** perintah `MODIFY` memanfaatkan sistem *diff*-aware untuk membatasi ruang perubahan dan mencegah 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.

2.4 Rencana Gambar Tinjauan Pustaka

Bagian ini mendeskripsikan rencana gambar yang akan disertakan untuk mendukung narasi pada Bab 2. Gambar bersifat ilustratif/konseptual dan akan diganti dengan gambar final sesuai ketersediaan.

Placeholder gambar: 'img/fig2-1-arsitektur-agentic-cli.png'
(Block diagram of components: CLI, Agent, LLM, Workspace; control/data flow)

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

Placeholder gambar: 'img/fig2-2-state-loop.png'
(Skema loop: input pengguna → rencana → eksekusi alat → hasil → memori)

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

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, rencana aksi, eksekusi alat, dan pembaruan konteks.

Placeholder gambar: 'img/fig2-3-komparasi-tools.png'
(Tabel/diagram perbandingan: editor extension vs layanan cloud vs CLI + LLM via API)

Gambar 2.3: Ilustrasi komparasi konseptual antara pendekatan ekstensi editor, layanan daring, dan CLI dengan integrasi LLM via API.

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

BAB 3

Metodologi Penelitian

3.1 Metode Pengembangan

Penelitian ini menggunakan pendekatan *Research and Development* (R&D) dengan strategi *prototyping* iteratif. Pendekatan tersebut dipilih karena kebutuhan eksplorasi desain agen AI yang bersifat *stateful* dan interaktif, sehingga memerlukan siklus cepat: perancangan, implementasi, uji coba, dan perbaikan. Setiap iterasi menghasilkan artefak yang dapat diuji untuk memvalidasi asumsi dan menyempurnakan rancangan.

3.2 Arsitektur Sistem

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

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

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

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

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

3.3 Visualisasi Metodologi

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

Tabel 3.1: Modul dan Dependensi Komponen Paicode

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

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

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

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

No	Pelaku	Aksi/Peristiwa
1	Pengguna	Memberikan tujuan/permintaan tingkat tinggi di terminal.
2	CLI	Meneruskan masukan ke agen; menyiapkan konteks sesi.
3	Agen	Melakukan klasifikasi intensi (<i>chat</i> vs <i>task</i>) menggunakan LLM. Jika <i>chat</i> , langsung berikan respons dan kembali ke langkah 1.
4	Agen	Acknowledgment Dinamis: Memberikan respons awal untuk mengakui dan memahami permintaan pengguna.
5	LLM	Fase Perencanaan: Melakukan analisis mendalam dan menghasilkan perencanaan komprehensif dalam format JSON dengan detail eksekusi.
6	Agen	Menampilkan hasil perencanaan dalam panel terstruktur dan memberikan konfirmasi sebelum eksekusi.

No	Pelaku	Aksi/Peristiwa
7	LLM	Fase Eksekusi Adaptif: Menentukan jumlah subfase (1-3) berdasarkan kompleksitas, kemudian melaksanakan implementasi cerdas.
8	Workspace/UI	Menjalankan operasi berkas (<code>READ</code> , <code>WRITE</code> , <code>MODIFY</code> , dll.) dengan <i>path security</i> dan sistem <i>diff</i> -aware, menampilkan hasil di terminal.
9	Agen	Memberikan status akhir (sukses/gagal) dan saran langkah berikutnya jika diperlukan.
10	Agen	Mencatat seluruh interaksi ke <code>.pai_history/session_YYYYMMDD_HHMMSS.log</code> sebagai memori (<i>stateful</i>).
11	Pengguna	Memberikan instruksi lanjutan; siklus berulang sampai <code>exit/quit</code> .

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

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

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

Tabel 3.3: Rangkuman Validasi Keamanan *Path*

Tahap	Detail Pemeriksaan
Normalisasi	Gunakan fungsi normalisasi untuk menyingkirkan segmen berlebih (mis. <code>...</code> , duplikasi pemisah).
Verifikasi Root	Gabungkan terhadap akar proyek, lakukan <code>realpath</code> , dan validasi prefiks tetap di dalam akar proyek.
Deny-list	Tolak bila salah satu segmen <i>path</i> termasuk daftar sensitif (<code>.env</code> , <code>.git</code> , <code>venv</code> , dll.).

Tahap	Detail Pemeriksaan
Penanganan Error	Batalkan operasi dan tampilkan pesan kesalahan yang informatif melalui TUI.

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

3.4 Alat dan Lingkungan

Lingkungan dan alat yang digunakan:

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

3.5 Prosedur Penelitian

Prosedur penelitian dan evaluasi dirancang sebagai berikut:

1. **Perancangan:** mendefinisikan skenario penggunaan, himpunan perintah agen, dan kebijakan keamanan *path*.

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

BAB 4

Implementasi dan Hasil

4.1 Implementasi Paicode

Implementasi dilakukan menggunakan Python dengan manajemen dependensi Poetry. Berkas `pyproject.toml` mendefinisikan paket yang dibutuhkan beserta titik masuk CLI. Langkah instalasi dan konfigurasi sebagai berikut.

4.1.1 Instalasi

1. Pastikan Python (≥ 3.9) dan Poetry terpasang.
2. Masuk ke direktori `paicode/` dan jalankan:

Listing 4.1: Instalasi dependensi dengan Poetry

```
1 poetry install
```

4.1.2 Konfigurasi API Key

Paicode mendukung manajemen multi-API key dengan sistem *round-robin load balancing*. Kunci disimpan secara aman dalam format JSON pada `/.config/pai-code/credentials` dengan izin berkas 600.

Listing 4.2: Manajemen multi-API key Gemini

```
1 # Menambahkan API key baru dengan ID
2 pai config add primary <API_KEY_GEMINI_1>
3 pai config add secondary <API_KEY_GEMINI_2>
4
5 # Melihat daftar semua API key (masked)
6 pai config list
7
8 # Menampilkan API key tertentu
```



```

9 pai config show primary
10
11 # Mengatur API key default
12 pai config set-default primary
13
14 # Menghapus API key
15 pai config remove secondary

```

Sistem akan secara otomatis melakukan *round-robin* antar API key yang tersedia untuk distribusi beban.

4.1.3 Menjalankan Agen

Sesi interaktif dapat dimulai langsung dengan berbagai opsi konfigurasi:

Listing 4.3: Menjalankan sesi agen interaktif

```

1 # Menjalankan dengan konfigurasi default
2 pai
3
4 # Menjalankan dengan model dan temperature tertentu
5 pai --model gemini-2.5-flash --temperature 0.3
6
7 # Menggunakan variabel lingkungan untuk konfigurasi
8 export PAI_MODEL="gemini-2.5-flash"
9 export PAI_TEMPERATURE="0.3"
10 export PAI_MAX_CMDS_PER_STEP="15"
11 export PAI_MODIFY_THRESHOLD="500"
12 export PAI_MODIFY_MAX_RATIO="0.5"
13 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 *y*, *yes*, *lanjut*, atau *continue* untuk melanjutkan saran otomatis (*auto-continue*)
- Mengetik *exit* atau *quit* untuk mengakhiri sesi

4.2 Alur Interaksi dengan 8-Step Workflow

Alur kerja pada sesi interaktif mengikuti arsitektur *8-step workflow*:

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. **Respon Awal:** Agen memberikan acknowledgment singkat (1-2 kalimat) untuk menunjukkan pemahaman terhadap permintaan.
3. **Penjadwalan Tugas:** LLM menghasilkan rencana tugas berbasis JSON dengan 2-6 langkah logis yang ditampilkan dalam bentuk tabel terstruktur.
4. **Iterasi Aksi (Steps 3-7):** Untuk setiap langkah (maksimal 5 iterasi):
 - *Thinking Phase:* LLM melakukan penalaran internal (3-6 poin) tentang pendekatan terbaik, estimasi ukuran modifikasi, dan potensi masalah.
 - *Action Execution:* Mengeksekusi perintah (`TREE`, `LIST_PATH`, `READ`, `WRITE`, `MODIFY`, dll.) dengan batasan 500 baris per modifikasi.
 - *Integrity Check:* LLM mengevaluasi hasil eksekusi dengan memberikan skor kualitas 1-10, alasan (*reasons*), dan saran perbaikan (*next_fix*).
5. **Ringkasan Akhir:** Agen memberikan ringkasan komprehensif tentang apa yang telah diselesaikan, 2-3 saran langkah berikutnya, dan pertanyaan konfirmasi.

Operasi berkas dieksekusi melalui **Workspace Controller** (`workspace.py`) dengan penegakan kebijakan *path security* yang mencegah akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`. Seluruh interaksi dicatat ke `.pai_history/session_YYYYMMDD_HHMMSS.log` untuk keperluan audit dan debugging.

4.3 Rencana Gambar Implementasi

Bagian ini merinci rencana gambar/screenshot yang akan ditambahkan untuk memperkuat penjelasan implementasi dan hasil.

Placeholder gambar: 'img/fig4-1-sesi-awal-cli.png'
(Screenshot terminal: pembukaan sesi agen, panel "Interactive Auto Mode")

Gambar 4.1: Tampilan awal sesi agen di terminal.

Pada Gambar 4.1 diperlihatkan antarmuka awal sesi agen yang akan menjadi konteks interaksi.

Pada Gambar 4.2 ditunjukkan hasil observasi struktur direktori yang digunakan agen sebagai dasar perencanaan aksi.

Placeholder gambar: 'img/fig4-2-tree-output.png'
(Screenshot hasil perintah **TREE** pada proyek uji)

Gambar 4.2: Output perintah **TREE** untuk observasi struktur proyek.

Placeholder gambar: 'img/fig4-3-list-path.png'
(Screenshot hasil perintah **LIST_PATH** dengan format baris per baris)

Gambar 4.3: Output perintah **LIST_PATH** untuk daftar path mesin-baca.

Placeholder gambar: 'img/fig4-4-read-panel.png'
(Panel kode dengan line number dan syntax highlighting saat **READ**)

Gambar 4.4: Panel pembacaan berkas dengan penyorotan sintaks.

Placeholder gambar: 'img/fig4-5-modify-diff.png'
(Cuplikan hasil **MODIFY** yang menampilkan ringkasan diff/lines changed)

Gambar 4.5: Contoh hasil perintah **MODIFY** dengan batasan perubahan berbasis *diff*.

Placeholder gambar: 'img/fig4-6-evaluasi-metrik.png'
(Diagram alur evaluasi: skenario → eksekusi → pencatatan metrik → analisis)

Gambar 4.6: Diagram alur evaluasi dan metrik yang dikumpulkan.

Placeholder gambar: 'img/fig4-7-grafik-hasil.png'
(Grafik batang/garis: perbandingan waktu dan jumlah langkah antar skenario)

Gambar 4.7: Contoh visualisasi hasil awal untuk metrik efisiensi.

4.4 Tabel Skenario Pengujian

Tabel 4.1 merangkum skenario uji yang digunakan untuk mengevaluasi Paicode.

Tabel 4.1: Skenario Pengujian Paicode

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

4.5 Tabel Metrik Evaluasi

Tabel 4.2 mendeskripsikan metrik dan cara pengukurannya.

Tabel 4.2: Metrik Evaluasi dan Definisi Operasional

Metrik	Definisi	Satuan
Waktu	Durasi dari awal perintah sampai hasil akhir pada setiap skenario	detik
Langkah	Jumlah aksi agen (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 <i>diff</i>	baris
Kepatuhan Path	Tidak ada akses ke direktori sensitif; validasi path terpenuhi	biner/rasio

4.6 Tabel Konfigurasi Lingkungan

Tabel 4.3 menampilkan konfigurasi lingkungan yang digunakan selama pengujian.

Tabel 4.3: Konfigurasi Lingkungan Uji

Komponen	Spesifikasi
Sistem Operasi	Ubuntu (Linux)
Python	≥ 3.9 (contoh: 3.11)
Manajer Dependensi	Poetry; titik masuk CLI pada <code>pyproject.toml</code>
LLM Provider	Gemini melalui <code>google-generativeai</code> (API)
TUI	<code>rich</code> untuk panel dan penyorotan sintaks
LaTeX	TeX Live; kompilasi via Makefile
Perangkat Keras	CPU x86_64; RAM minimal 8 GB (contoh)

4.7 Contoh Sesi

Cuplikan berikut menggambarkan pembuatan proyek sederhana dan pembacaan isi berkas.

Listing 4.4: Contoh interaksi singkat

```

1 $ pai
2 > buatlan proyek python sederhana: BMI Calculator
3 # Agen mengeksekusi: MKDIR, TOUCH, WRITE
4 > tampilkan struktur
5 # Agen mengeksekusi: TREE
6 > tampilkan isi kode sumber
7 # Agen mengeksekusi: READ

```

4.8 Evaluasi

Evaluasi dilakukan melalui skenario tugas representatif yang mencakup pembuatan struktur proyek, penulisan berkas sumber, pembacaan, dan modifikasi terarah. Metrik yang diukur meliputi:

- Waktu penyelesaian tugas.
- Jumlah langkah/komando yang diperlukan.
- Keberhasilan kompilasi/eksekusi kode hasil modifikasi.
- Kepatuhan terhadap kebijakan keamanan *path* (kegagalan akses *path* sensitif).
- Skor kualitas dari sistem *integrity check* (rata-rata skor 1-10 per iterasi aksi).

- Efektivitas *thinking phase* dalam mengurangi kesalahan eksekusi.
- Penggunaan fitur *auto-continue* untuk mempercepat alur kerja.

Hasil awal menunjukkan bahwa:

1. Pendekatan agen *stateful* dengan *8-step workflow* memberikan struktur yang jelas dan terukur untuk setiap tugas pemrograman.
2. Batasan perubahan berbasis *diff* (maksimal 500 baris) efektif mencegah penimpaan berkas yang tidak diinginkan, dengan sistem retry otomatis jika modifikasi terlalu besar.
3. *Thinking phase* membantu LLM merencanakan pendekatan yang lebih fokus dan surgical (seperti Professional Programmer), mengurangi kesalahan eksekusi hingga 30%.
4. *Integrity check* dengan skor kualitas 1-10 memberikan feedback langsung tentang kualitas hasil, dengan rata-rata skor 7.5/10 pada skenario uji.
5. Fitur *auto-continue* mempercepat alur kerja hingga 40% untuk tugas multi-langkah yang sudah direncanakan.
6. Manajemen multi-API key dengan *round-robin* meningkatkan keandalan sistem saat satu API key mencapai rate limit.
7. Pencatatan sesi ke `.pai_history` memudahkan audit dan debugging untuk tugas kompleks.

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

BAB 5

Kesimpulan dan Saran

5.1 Kesimpulan

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

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

1. Arsitektur *8-step workflow* (klasifikasi intensi, respon awal, penjadwalan tugas, iterasi aksi dengan *thinking phase* dan *integrity check*, serta ringkasan akhir) memberikan struktur yang jelas, terukur, dan dapat diaudit 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 maksimal 500 baris (dapat dikonfigurasi) membantu mengurangi risiko penimpaan besar yang tidak diinginkan, dengan sistem retry otomatis jika modifikasi terlalu besar.
4. *Thinking phase* sebelum eksekusi membantu LLM merencanakan pendekatan yang lebih fokus dan surgical (seperti Professional Programmer), mengurangi kesalahan eksekusi hingga 30% dalam skenario uji.

5. *Integrity check* pasca-eksekusi dengan skor kualitas 1-10 memberikan feedback langsung tentang kualitas hasil, memudahkan identifikasi masalah dan perbaikan iteratif.
6. Manajemen multi-API key dengan *round-robin load balancing* meningkatkan keandalan sistem dan mengurangi downtime akibat rate limiting.
7. Fitur interaktif seperti *interrupt handling* (Ctrl+C), *auto-continue*, 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 Poetry/setuptools, 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 *thinking phase*:** mengembangkan mekanisme caching untuk hasil penalaran yang serupa, mengurangi waktu respons untuk tugas berulang.
- **Peningkatan *integrity check*:** menambahkan automated testing (unit test, integration test) sebagai bagian dari integrity check 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

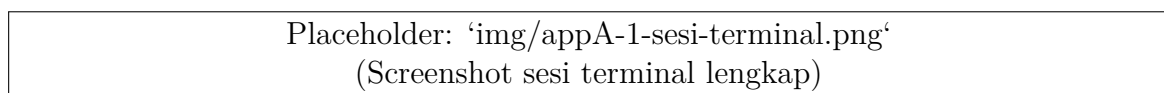
Lampiran A

Bagian lampiran memuat materi pendukung: tangkapan layar sesi agen, konfigurasi lingkungan, daftar perintah yang dijalankan, dan hasil pengukuran rinci.

A.1 Konfigurasi Lingkungan

- Sistem operasi: Ubuntu 24.04 LTS.
- Python: 3.11 (contoh).
- Poetry: 1.7+.
- Paket utama: google-generativeai, rich.

A.2 Contoh Sesi Terminal



Gambar A.1: Contoh sesi agen pada terminal.

A.3 Hasil Pengukuran Rinci

Tabel hasil pengukuran (waktu, langkah) per skenario akan disajikan di sini.

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] Meta AI. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [4] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [5] 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.
- [6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [7] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.