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

2025

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. 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.

Metode yang digunakan adalah *Research and Development* (R&D) dengan pendekatan *prototyping* iteratif. Evaluasi awal dilakukan melalui skenario tugas representatif, dengan metrik efisiensi (waktu/ langkah), ketepatan hasil (kompilasi/ eksekusi), serta kepatuhan keamanan *path*. Hasil menunjukkan bahwa agen *stateful* dengan pembatasan perubahan berbasis *diff* memudahkan pengembangan bertahap sambil menekan risiko penimpaan berkas.

Kata kunci: agentic AI, CLI, LLM, API, 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. 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.

We adopt a Research and Development approach with iterative prototyping. The initial evaluation uses representative programming scenarios and measures efficiency (time/steps), correctness (build/run), and security compliance. Results indicate that a stateful agent with *diff*-based change constraints facilitates incremental development while reducing the risk of unintended overwrites.

Keywords: agentic AI, CLI, LLM, API, 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

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.
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.
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).

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, 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. Paicode mampu: (i) mengobservasi struktur proyek (`TREE`, `LIST_PATH`); (ii) membaca dan menulis berkas proyek (`READ`, `WRITE`); (iii) memodifikasi kode secara terarah dengan batasan perubahan berbasis diff (`MODIFY`); serta (iv) menegakkan kebijakan keamanan path pada berkas proyek (memblokir akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`). Sistem diimplementasikan pada lingkungan Ubuntu dengan bahasa pemrograman Python, pengelolaan dependensi melalui Poetry, 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 yang *stateful* dan *proactive* dengan integrasi LLM melalui API, disertai kebijakan keamanan *path* dan pembatasan perubahan berbasis *diff* untuk mendukung proses pengembangan perangkat lunak?
2. Bagaimana mengintegrasikan kemampuan observasi proyek, manipulasi berkas, serta modifikasi kode terarah berbasis deskripsi pengguna dengan pengamanan terhadap *path* dan direktori sensitif?
3. Bagaimana mengevaluasi efektivitas Paicode dalam membantu tugas-tugas pemrograman dibandingkan proses manual atau alat pembanding yang relevan?

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. Secara khusus, penelitian menargetkan:

1. Merancang arsitektur Paicode yang mencakup modul agen, jembatan LLM, antarmuka CLI, lapisan keamanan path pada berkas proyek, serta komponen tampilan terminal.
2. Mengimplementasikan kemampuan observasi proyek, manipulasi berkas, dan modifikasi kode terarah dengan mekanisme *patch/diff* untuk membatasi ruang perubahan.
3. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur peningkatan produktivitas atau kualitas hasil.

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), serta perangkat bantu yang digunakan seperti Poetry untuk manajemen dependensi dan `rich` 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 *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*.

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 Poetry

Poetry menyediakan manajemen dependensi dan kemasan proyek Python yang deterministik. Berkas `pyproject.toml` mendeskripsikan dependensi dan titik masuk CLI (`pai`). Pendekatan ini memudahkan replikasi lingkungan dan distribusi alat.

2.1.7 Antarmuka Terminal dengan `rich`

Paket `rich` dimanfaatkan untuk menyajikan hasil eksekusi secara terstruktur dan mudah dibaca (panel, warna, penyorotan sintaks). Penyajian output yang jelas mendukung pengalaman interaktif dan penelusuran hasil tindakan agen.

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.
- **Keamanan berkas**: kebijakan pelarangan akses *path* sensitif dan validasi *path* mencegah *path traversal* dan operasi berisiko.
- **Modifikasi terarah**: perintah `MODIFY` memanfaatkan *diff* untuk membatasi ruang perubahan, mendukung prinsip perubahan minimal.
- **Keterulangan eksperimen**: penggunaan Poetry dan Makefile memudahkan replikasi lingkungan dan dokumentasi langkah.

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.

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

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 2.2 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`). Secara default, CLI memanggil sesi interaktif agen.
- **Agen (`agent.py`)**: menyusun prompt, mengelola memori percakapan, dan mengeksekusi rencana aksi hasil LLM. Menyediakan perintah: `MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`.
- **Jembatan LLM (`llm.py`)**: menangani konfigurasi API Gemini dan penyederhanaan hasil keluaran.
- **Pengatur Workspace (`workspace.py`)**: bertindak sebagai *workspace controller* yang menyediakan fungsi-fungsi terpusat untuk menjalankan operasi tingkat-aplikasi pada ruang kerja proyek (membaca/menulis, membuat/menghapus/memindahkan, menampilkan pohon/list path, dan modifikasi berbasis *diff*). Sebelum aksi dieksekusi, modul ini menegakkan kebijakan *path security* (normalisasi dan verifikasi akar, serta deny-list direktori sensitif). Seluruh operasi dibatasi pada workspace proyek secara terkontrol.

- **Tampilan Terminal (`ui.py`)**: penyajian hasil eksekusi menggunakan `rich` (panel, warna, penomoran baris).

Alur data tipikal: masukan pengguna (CLI) → konstruksi prompt (Agen) → panggilan LLM → rencana aksi → eksekusi tindakan (Workspace/UI) → pelaporan dan 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; memanggil sesi agen. Bergantung pada modul `agent` dan `config`.
Agen (`agent.py`)	Penyusun prompt, eksekusi rencana aksi, memori percakapan; memanggil `llm`, `workspace`, `ui`.
LLM Bridge (`llm.py`)	Integrasi Gemini API (`google-generativeai`); mengambil API key dari `config`.
Pengatur Workspace (`workspace.py`)	Bertindak sebagai *workspace controller* yang menyediakan fungsi-fungsi operasi workspace (baca/tulis, buat/hapus/pindah, tree/list path, dan modifikasi berbasis *diff*) dengan penegakan *path security* (normalisasi, verifikasi akar, dan deny-list). Seluruh operasi dibatasi pada workspace proyek, bukan driver filesystem OS.
Terminal UI (`ui.py`)	Komponen TUI berbasis `rich`: panel, tema, syntax highlighting.

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

Tabel 3.2: Urutan Interaksi Sesi Agen (Stateful Feedback Loop)

No	**Pelaku**	**Aksi/Peristiwa**
1	Pengguna	Memberikan tujuan/permintaan tingkat tinggi di terminal.
2	CLI	Meneruskan masukan ke agen; menyiapkan konteks sesi.
3	Agen	Menyusun prompt dengan memori percakapan (*conversation history*).

No	Pelaku	Aksi/Peristiwa
4	LLM	Menghasilkan rencana aksi (urutan perintah) berdasarkan prompt.
5	Agen	Mengeksekusi rencana: `TREE`/`LIST_PATH`/`READ`/`WRITE`/`MODIFY`/dst.
6	Workspace/UI	Menjalankan operasi berkas (dengan *path security* dan *diff-aware*) dan menampilkan hasil di terminal.
7	Agen	Mencatat hasil (`System Response`) sebagai memori untuk langkah berikutnya (*stateful*).
8	Pengguna	Memberikan instruksi lanjutan; siklus berulang sampai `FINISH`.

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 (`>= 3.9`).
- Manajer dependensi/kemasan: Poetry; titik masuk CLI didefinisikan pada `pyproject.toml`.
- LLM: Google Gemini (model `gemini-2.5-flash`) melalui paket `google-generativeai`.
- TUI: `rich` untuk panel, warna, dan penyorotan sintaks.
- 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

```
poetry install
```

4.1.2 Konfigurasi API Key

Paicode memerlukan API key Gemini untuk akses LLM. Kunci disimpan secara aman pada `/.config/pai-code/credentials` dengan izin berkas 600.

Listing 4.2: Set dan verifikasi API key Gemini

```
poetry run pai config --set <API_KEY_GEMINI>
poetry run pai config --show
```

4.1.3 Menjalankan Agen

Sesi interaktif dapat dimulai langsung:

Listing 4.3: Menjalankan sesi agen interaktif

```
poetry run pai
```

4.2 Alur Interaksi

Alur kerja pada sesi interaktif meliputi: (i) pengguna memberikan tujuan tingkat tinggi; (ii) agen mengobservasi struktur proyek menggunakan perintah `TREE`/`LIST_PATH`; (iii) agen membaca/menulis/memodifikasi berkas; (iv) hasil dievaluasi dan menjadi konteks untuk langkah berikutnya. Kebijakan keamanan *path* mencegah akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`.

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.

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.

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

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.

4.4 Tabel Skenario Pengujian

Tabel 4.1 merangkum skenario uji yang digunakan untuk mengevaluasi Paicode.

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*.

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
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

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.

Metrik	Definisi	Satuan
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

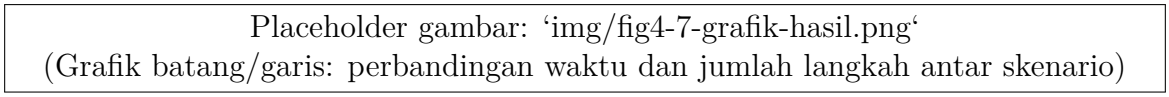
Komponen	Spesifikasi
Sistem Operasi	Ubuntu (Linux)
Python	`>= 3.9` (contoh: 3.11)
Manajer Dependensi	Poetry; titik masuk CLI pada `pyproject.toml`
LLM Provider	Gemini melalui `google-generativeai` (API)
TUI	`rich` untuk panel dan penyorotan sintaks
LaTeX	TeX Live; kompilasi via Makefile
Perangkat Keras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

4.7 Contoh Sesi

Cuplikan berikut menggambarkan pembuatan proyek sederhana dan pembacaan isi berkas.

Listing 4.4: Contoh interaksi singkat

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



Gambar 4.7: Contoh visualisasi hasil awal untuk metrik efisiensi.

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).

Hasil awal menunjukkan bahwa pendekatan agen *stateful* dengan batasan perubahan berbasis *diff* memudahkan penulisan dan pengembangan bertahap sambil menekan risiko penimpaan berkas yang tidak diinginkan. 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. 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.

2. Mekanisme pembatasan perubahan berbasis *diff* pada perintah `MODIFY` membantu mengurangi risiko penimpaan besar yang tidak diinginkan, sehingga sejalan dengan prinsip perubahan minimal.

3. Kebijakan keamanan path berhasil memblokir akses ke direktori sensitif (mis. `.git`, `venv`, `.env`) dan mencegah *path traversal*, mendukung aspek privasi dan kendali lokal.

4. Pemakaian Poetry, 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 sesuai kebutuhan (akurasi/biaya/latensi).
- **Integrasi editor**: menyediakan jembatan ringan ke IDE (mis. VS Code) 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, dan memperketat validasi konten sebelum penulisan berkas.
- **Memori jangka panjang**: menambahkan ringkasan sesi dan penyimpanan konteks terkurasi agar agen dapat mempelajari preferensi proyek pengguna secara berkelanjutan.
- **Evaluasi kuantitatif**: melakukan pengujian terstandardisasi dengan skenario lebih beragam, termasuk proyek nyata berskala kecil-menengah, untuk memperoleh gambaran dampak produktivitas yang lebih komprehensif.

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

Placeholder: ‘img/appA-1-sesi-terminal.png‘
(Screenshot sesi terminal lengkap)

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.