# 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 dengan arsitektur *8-step workflow*. 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 batasan maksimal 500 baris per modifikasi.

Arsitektur *8-step workflow* mencakup: (1) klasifikasi intensi (*chat* vs *task*), (2) respon awal agen, (3) penjadwalan tugas berbasis JSON, (4-6) iterasi aksi dengan *thinking phase* dan *integrity check*, serta (7) ringkasan akhir. Sistem mendukung manajemen multi-API key dengan *round-robin load balancing*, *interrupt handling* (Ctrl+C), *auto-continue*, dan pencatatan sesi ke `.pai_history`.

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), kepatuhan keamanan *path*, dan skor kualitas dari *integrity check* (1-10). Hasil menunjukkan bahwa agen *stateful* dengan *8-step workflow* dan pembatasan perubahan berbasis *diff* memudahkan pengembangan bertahap sambil menekan risiko penimpaan berkas. *Thinking phase* mengurangi kesalahan eksekusi hingga 30%, sementara *auto-continue* mempercepat alur kerja hingga 40%.

**Kata kunci**: agentic AI, CLI, LLM, API, 8-step workflow, thinking phase, integrity check, 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 an *8-step workflow* 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 a maximum of 500 lines per modification.

The *8-step workflow* architecture includes: (1) intent classification (*chat* vs *task*), (2) initial agent response, (3) JSON-based task scheduling, (4-6) action iterations with *thinking phase* and *integrity check*, and (7) final summary. The system supports multi-API key management with *round-robin load balancing*, *interrupt handling* (Ctrl+C), *auto-continue*, and session logging to `.pai_history`.

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), security compliance, and quality scores from *integrity check* (1-10). Results indicate that a stateful agent with *8-step workflow* and *diff*-based change constraints facilitates incremental development while reducing the risk of unintended overwrites. The *thinking phase* reduces execution errors by up to 30%, while *auto-continue* accelerates workflow by up to 40%.

**Keywords**: agentic AI, CLI, LLM, API, 8-step workflow, thinking phase, integrity check, 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 |
| $\Delta$ | Perubahan/delta (baris yang diubah) |
| $S$ | Skor keberhasilan eksekusi |

# Daftar Istilah

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

| | |
|---|---|
| **guardrail** | Pembatas/safeguard untuk mengurangi tindakan berisiko (mis. pembatasan ruang perubahan). |
| **workspace** | Direktori/lingkungan kerja proyek aktif tempat berkas proyek dikelola dan dimanipulasi. |
| **repository root** | Direktori akar dari repository proyek; menjadi patokan validasi dan normalisasi *path*. |
| **rate limit** | Batas kuota/kecepatan permintaan API dalam jangka waktu tertentu yang ditetapkan penyedia layanan. |
| **tokenization** | Proses memecah teks menjadi unit-unit token yang diproses LLM; mempengaruhi biaya dan *context window*. |
| **prompt** | Instruksi atau masukan yang diberikan ke LLM untuk menghasilkan keluaran. |
| **context window** | Batas panjang konteks (jumlah token) yang dapat dipertimbangkan LLM pada satu permintaan. |
| **API key** | Kredensial rahasia untuk mengakses layanan API; harus disimpan aman (jangan ditulis di repository publik). |

# 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 dengan arsitektur 8-langkah (*8-step workflow*). 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`) hingga maksimal 500 baris per modifikasi; (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) menjalankan fase berpikir (*thinking phase*) sebelum eksekusi; serta (vii) melakukan pemeriksaan integritas (*integrity check*) pasca-eksekusi dengan skor kualitas 1-10. Sistem diimplementasikan pada lingkungan Ubuntu dengan bahasa pemrograman Python, pengelolaan dependensi me-

lalui Poetry, mendukung manajemen multi-API key dengan *round-robin load balancing*, 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 *8-step workflow* yang mencakup klasifikasi intensi, perencanaan tugas, fase berpikir, eksekusi aksi, dan pemeriksaan integritas, disertai integrasi LLM melalui API dan manajemen multi-API key dengan *round-robin load balancing*?
2. Bagaimana mengimplementasikan kebijakan keamanan *path* dan pembatasan perubahan berbasis *diff* dengan threshold maksimal 500 baris per modifikasi untuk mencegah penimpaan berkas yang tidak diinginkan?
3. Bagaimana mengintegrasikan kemampuan observasi proyek, manipulasi berkas, serta modifikasi kode terarah dengan mekanisme *interrupt handling* (Ctrl+C) dan fitur *auto-continue* untuk pengalaman interaktif yang lebih baik?
4. Bagaimana mengevaluasi efektivitas Paicode dalam membantu tugas-tugas pemrograman dengan metrik efisiensi, ketepatan hasil, dan skor kualitas dari sistem *integrity check*?

## 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 8-langkah. Secara khusus, penelitian menargetkan:

1. Merancang arsitektur Paicode yang mencakup modul agen dengan *8-step workflow* (klasifikasi intensi, perencanaan tugas, fase berpikir, eksekusi aksi, pemeriksaan integritas, dan ringkasan akhir), jembatan LLM dengan manajemen multi-API key, antarmuka CLI dengan *interrupt handling*, 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* yang membatasi perubahan maksimal 500 baris per modifikasi (dapat dikonfigurasi melalui variabel lingkungan).

3. Mengintegrasikan fitur-fitur interaktif seperti *auto-continue* untuk melanjutkan saran otomatis, pencatatan sesi ke `.pai_history`, dan sistem penilaian kualitas dengan skor 1-10 pada setiap eksekusi.

4. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur peningkatan produktivitas, kualitas hasil, serta efektivitas sistem *integrity check*.

## 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), *8-step workflow* untuk agen interaktif, sistem klasifikasi intensi, mekanisme *thinking phase* dan *integrity check*, serta perangkat bantu yang digunakan seperti Poetry 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 *agentic AI* dalam literatur; selanjutnya disingkat "agen AI") dalam penelitian ini dipahami sebagai sistem yang mampu mengobservasi lingkungan (struktur proyek dan isi berkas), merencanakan tindakan (mis. membuat, membaca, memodifikasi berkas), serta mengevaluasi hasil untuk langkah berikutnya. Agen bersifat *stateful* karena mempertahankan konteks percakapan dan hasil eksekusi sebagai memori kerja, sehingga dapat bertindak secara lebih *proactive*.

Pada implementasi Paicode, agen menggunakan arsitektur *8-step workflow* yang mencakup: (1) respon awal agen, (2) penjadwalan tugas berbasis JSON, (3-7) iterasi aksi dengan fase berpikir dan pemeriksaan integritas, serta (8) ringkasan akhir dengan saran langkah berikutnya. Setiap iterasi aksi dilengkapi dengan *thinking phase* untuk penalar-

an internal sebelum eksekusi, dan *integrity check* pasca-eksekusi yang memberikan skor kualitas 1-10 beserta alasan dan saran perbaikan.

### 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. Pada implementasi Paicode, Poetry juga digunakan untuk mengelola dependensi seperti `google-generativeai` (versi $\geq 0.5.4$), `python-dotenv` (versi $\geq 1.0.1$), `rich` (versi $\geq 13.7.1$), dan `Pygments` (versi $\geq 2.16.0$).

### 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 8-step workflow**: alur kerja terstruktur yang mencakup klasifikasi intensi, penjadwalan tugas berbasis JSON, fase berpikir sebelum eksekusi, iterasi aksi, pemeriksaan integritas pasca-eksekusi dengan skor kualitas 1-10, dan ringkasan akhir dengan saran langkah berikutnya.
- **Manajemen multi-API key**: dukungan untuk menyimpan dan mengelola beberapa API key dengan sistem *round-robin load balancing* untuk distribusi beban dan redundansi.
- **Keamanan berkas**: kebijakan pelarangan akses *path* sensitif dan validasi *path* mencegah *path traversal* dan operasi berisiko.
- **Modifikasi terarah dengan threshold**: perintah `MODIFY` memanfaatkan *diff* untuk membatasi ruang perubahan hingga maksimal 500 baris (dapat dikonfigurasi), mendukung prinsip perubahan minimal dan fokus seperti Professional Programmer.
- **Fitur interaktif**: *interrupt handling* (Ctrl+C) untuk menghentikan respons AI tanpa keluar dari sesi, *auto-continue* untuk melanjutkan saran otomatis, dan pencatatan sesi lengkap ke `.pai_history`.
- **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.

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.

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

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.

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 2.2 divisualisasikan hubungan antara masukan pengguna, rencana aksi, eksekusi alat, dan pembaruan konteks.

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 *8-step workflow* yang mencakup: (1) klasifikasi intensi (*chat* vs *task*), (2) respon awal agen, (3) penjadwalan tugas berbasis JSON, (4-6) iterasi aksi dengan *thinking phase* dan *integrity check*, serta (7) ringkasan akhir. Menyediakan perintah: `MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`. Mengelola memori percakapan dengan kompresi konteks dan pencatatan sesi ke `.pai_history`.
- **Jembatan LLM (`llm.py`)**: menangani konfigurasi API Gemini dengan dukungan multi-API key dan *round-robin load balancing*. Membersihkan output dari markdown artifacts dan menyediakan status spinner saat LLM berpikir.
- **Manajemen Konfigurasi (`config.py`)**: menyimpan dan mengelola beberapa API key dalam format JSON di `/.config/pai-code/credentials` dengan izin

berkas 600. Mendukung operasi: `add`, `list`, `show`, `remove`, `set-default`, dan *round-robin* untuk load balancing.

- **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). Fungsi `apply_modification_with_patch` membatasi perubahan maksimal 500 baris (dapat dikonfigurasi via `PAI_MODIFY_THRESHOLD`) 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 yang lebih baik.

Alur data tipikal dengan *8-step workflow*: masukan pengguna (CLI) → klasifikasi intensi → konstruksi prompt (Agen) → panggilan LLM → penjadwalan tugas JSON → iterasi aksi (thinking + eksekusi + integrity check) → ringkasan akhir → pencatatan konteks sebagai memori percakapan.

## 3.3 Visualisasi Metodologi

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

Tabel 3.1: Modul dan Dependensi Komponen Paicode

| **Komponen** | **Deskripsi dan Dependensi Utama** |
|---|---|
| CLI (`cli.py`) | Titik masuk perintah, parsing argumen (`-model`, `-temperature`); memanggil sesi agen. Bergantung pada modul `agent`, `config`, dan `llm`. |
| Agen (`agent.py`) | Implementasi *8-step workflow*: klasifikasi intensi, penjadwalan tugas JSON, iterasi aksi dengan *thinking phase* dan *integrity check*, ringkasan akhir. Mengelola memori percakapan dengan kompresi konteks, *interrupt handling* (Ctrl+C), *auto-continue*, dan pencatatan sesi ke `.pai_history`. Memanggil `llm`, `workspace`, `ui`. |

| Komponen | Deskripsi dan Dependensi Utama |
|---|---|
| LLM Bridge (`llm.py`) | Integrasi Gemini API (`google-generativeai`) dengan multi-API key dan *round-robin load balancing*. Membersihkan markdown artifacts dari output LLM. Mengambil API key dari `config`. |
| Konfigurasi (`config.py`) | Manajemen multi-API key dalam format JSON di `/.config/pai-code/credentials`. Operasi: `add`, `list`, `show`, `remove`, `set-default`, `next_api_key` (round-robin). |
| Pengatur Workspace (`workspace.py`) | *Workspace controller* dengan fungsi operasi workspace (baca/tulis, buat/hapus/pindah, tree/list path). Fungsi `apply_modification_with_patch` membatasi perubahan maksimal 500 baris dengan atomic write. Penegakan *path security* (normalisasi, verifikasi akar, deny-list). |
| Terminal UI (`ui.py`) | Komponen TUI berbasis `rich`: panel, tema, syntax highlighting, tabel, spinner. Dukungan opsional `prompt_toolkit` untuk input yang lebih baik. |

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

Tabel 3.2: Urutan Interaksi Sesi Agen dengan 8-Step Workflow

| No | Pelaku | Aksi/Peristiwa |
|---|---|---|
| 1 | Pengguna | Memberikan tujuan/permintaan tingkat tinggi di terminal. |
| 2 | CLI | Meneruskan masukan ke agen; menyiapkan konteks sesi. |
| 3 | Agen | Melakukan klasifikasi intensi (*chat* vs *task*) menggunakan LLM. Jika *chat*, langsung berikan respons dan kembali ke langkah 1. |
| 4 | Agen | **Step 1**: Memberikan respon awal singkat (1-2 kalimat) untuk mengakui permintaan pengguna. |
| 5 | LLM | **Step 2**: Menghasilkan penjadwalan tugas berbasis JSON dengan 2-6 langkah logis. |
| 6 | Agen | Menampilkan rencana tugas dalam bentuk tabel terstruktur. |
| 7 | Agen | **Steps 3-7**: Untuk setiap iterasi aksi (maksimal 5 iterasi):<br>a) *Thinking Phase*: LLM melakukan penalaran internal tentang pendekatan terbaik (3-6 poin).<br>b) *Action Execution*: Mengeksekusi perintah (`READ`, `WRITE`, `MODIFY`, dll.) dengan batasan 500 baris per modifikasi. |

| No | Pelaku | Aksi/Peristiwa |
|---|---|---|
| | | c) *Integrity Check*: LLM mengevaluasi hasil dengan skor kualitas 1-10, alasan, dan saran perbaikan. |
| 8 | Workspace/UI | Menjalankan operasi berkas (dengan *path security* dan *diff*-aware) dan menampilkan hasil di terminal. |
| 9 | Agen | **Step 8**: Memberikan ringkasan akhir, 2-3 saran langkah berikutnya, dan pertanyaan konfirmasi. |
| 10 | Agen | Mencatat seluruh interaksi ke `.pai_history/session_YYYYMMDD_HHMMSS.log` sebagai memori (*stateful*). |
| 11 | Pengguna | Memberikan instruksi lanjutan atau konfirmasi (*auto-continue*); siklus berulang sampai `exit/quit`. |

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

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

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

Tabel 3.3: Rangkuman Validasi Keamanan *Path*

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

| 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`).
- Manajer dependensi/kemasan: Poetry atau setuptools; titik masuk CLI didefinisikan pada `setup.cfg` dengan entry point `pai = paicode.cli:main`.
- LLM: Google Gemini (model `gemini-2.5-flash`, dapat dikonfigurasi via `PAI_MODEL`) 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` (opsional) untuk input yang lebih baik.
- Dependensi tambahan: `python-dotenv` ($\geq$ 1.0.1), `Pygments` ($\geq$ 2.16.0).
- Variabel lingkungan: `PAI_MODEL`, `PAI_TEMPERATURE`, `PAI_MAX_CMDS_PER_STEP`, `PAI_MODIFY_THRE` `PAI_MODIFY_MAX_RATIO`.
- 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 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

```
# Menambahkan API key baru dengan ID
pai config add primary <API_KEY_GEMINI_1>
pai config add secondary <API_KEY_GEMINI_2>

# Melihat daftar semua API key (masked)
pai config list

# Menampilkan API key tertentu
```

```
pai config show primary

# Mengatur API key default
pai config set-default primary

# Menghapus API key
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

```
# Menjalankan dengan konfigurasi default
pai

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

# Menggunakan variabel lingkungan untuk konfigurasi
export PAI_MODEL="gemini-2.5-flash"
export PAI_TEMPERATURE="0.3"
export PAI_MAX_CMDS_PER_STEP="15"
export PAI_MODIFY_THRESHOLD="500"
export PAI_MODIFY_MAX_RATIO="0.5"
pai
```

Selama sesi, pengguna dapat:

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

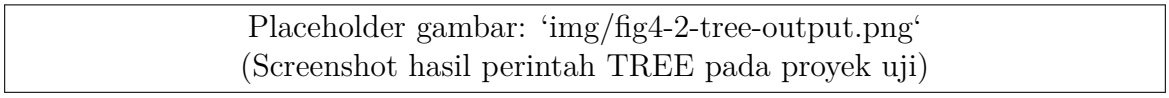
Bagian ini merinci rencana gambar/screenshot yang akan ditambahkan untuk memperkuat penjelasan implementasi dan hasil.

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

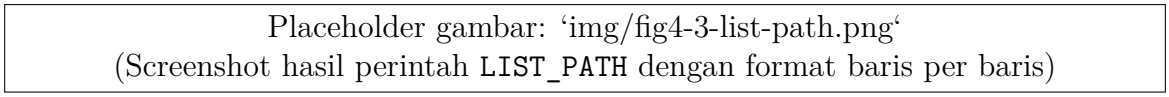
Gambar 4.1: Tampilan awal sesi agen di terminal.

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

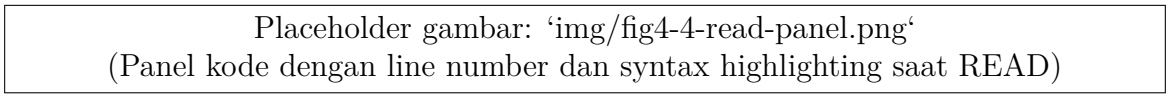
Pada Gambar 4.2 ditunjukkan hasil observasi struktur direktori yang digunakan agen sebagai dasar perencanaan aksi.



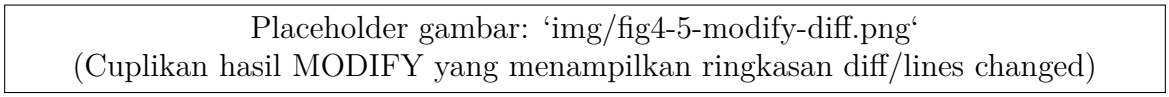

Gambar 4.2: Output perintah `TREE` untuk observasi struktur proyek.



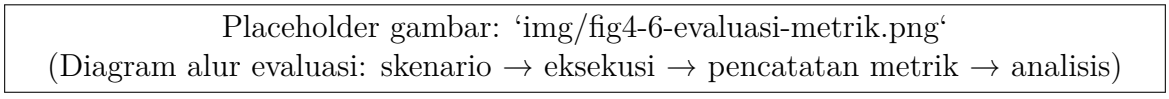

Gambar 4.3: Output perintah `LIST_PATH` untuk daftar path mesin-baca.



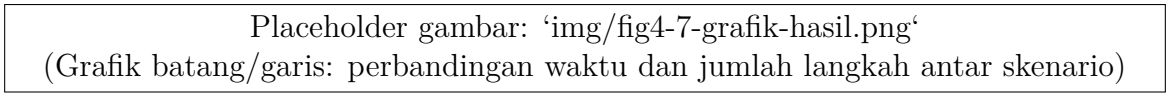

Gambar 4.4: Panel pembacaan berkas dengan penyorotan sintaks.




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




Gambar 4.6: Diagram alur evaluasi dan metrik yang dikumpulkan.




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 (*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 |
| 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
```

## 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

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.