TUGAS AKHIR

SKEMA SKRIPSI

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

I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001

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

TUGAS AKHIR

SKEMA SKRIPSI

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

Diajukan sebagai salah satu syarat untuk menyelesaikan studi pada
Program Sarjana
Program Studi Informatika
Fakultas Teknologi Informasi
Universitas Teknologi Digital Indonesia

Disusun Oleh

I PUTU GEDE GILANG TEJA KRISHNA
NIM : 225410001

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

HALAMAN PERSETUJUAN UJIAN TUGAS AKHIR

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

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

Yogyakarta, 24 November 2025
Dosen Pembimbing,

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

HALAMAN PENGESAHAN

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

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

Yogyakarta, 24 November 2025

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

Mengetahui
Ketua Program Studi Informatika

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

PERNYATAAN KEASLIAN TUGAS AKHIR

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

Yogyakarta, 24 November 2025

I PUTU GEDE GILANG TEJA KRISHNA
NIM: 225410001

HALAMAN PERSEMBAHAN

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

PRAKATA

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

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

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

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

Yogyakarta, 24 November 2025

Penulis

Daftar Isi

Daftar Gambar

Daftar Tabel

INTISARI

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

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

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

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

ABSTRACT

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

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

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

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

BAB I

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 Brown et al. (2020); OpenAI (2023); Anil et al. (2023). Dengan demikian, aspek privasi/kerahasiaan kode **bergantung pada kebijakan penyedia API**, sementara pengamanan di sisi lokal difokuskan pada kebijakan *path security* (keamanan *path*) dan pembatasan perubahan berbasis *diff*.

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

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

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah yang diajukan adalah:

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

1.3 Ruang Lingkup

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

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

1.4 Tujuan Penelitian

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

1. Merancang arsitektur Paicode yang mencakup modul agen dengan *Single-Shot Intelligence* (klasifikasi intensi, fase perencanaan, dan fase eksekusi dalam 2 panggilan API), jembatan LLM dengan manajemen API key tunggal, antarmuka CLI dengan *interrupt handling*, lapisan keamanan *path* pada berkas proyek, serta komponen tampilan terminal berbasis `rich`.

2. Mengimplementasikan kemampuan observasi proyek, manipulasi berkas, dan modifikasi kode terarah dengan mekanisme *diff*-aware yang mencegah penimpaan berkas tidak diinginkan dan memblokir akses ke direktori sensitif.

3. Mengintegrasikan fitur-fitur interaktif seperti pencatatan sesi ke `.pai_history`, penanganan interupsi (Ctrl+C), dan antarmuka terminal yang responsif dengan dukungan input multiline.

4. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur efisiensi panggilan API, ketepatan hasil, serta kepatuhan keamanan *path*.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

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

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

    1. **Efisiensi Biaya dan Token**: Menggunakan arsitektur *Single-Shot Intelligence* yang memadatkan proses perencanaan dan eksekusi menjadi dua panggilan utama, mengurangi biaya API dibandingkan agen berbasis *chat-loop* konvensional.
    2. **Keamanan Terkendali**: Menerapkan kebijakan keamanan *path* (path security) yang memblokir akses ke direktori sensitif (seperti `.git`, `.env`) dan mekanisme modifikasi berbasis *diff* untuk mencegah perubahan destruktif masif.

3. **Fleksibilitas Lingkungan**: Beroperasi sebagai utilitas CLI yang ringan dan agnostik terhadap editor kode (IDE-agnostic), sehingga dapat digunakan di server tanpa antarmuka grafis (headless) maupun sebagai pendamping editor apa pun di OS berbasis Linux.

1.6 Sistematika Penulisan

Laporan tugas akhir ini disusun dalam lima bab yang saling berkaitan. Bab I (Pendahuluan) memaparkan latar belakang masalah, rumusan masalah yang akan diselesaikan, ruang lingkup penelitian, tujuan yang ingin dicapai, manfaat penelitian bagi aspek akademis maupun praktis, serta sistematika penulisan laporan ini.

Bab II (Tinjauan Pustaka dan Dasar Teori) menguraikan tinjauan pustaka dari penelitian-penelitian terdahulu yang relevan dengan pengembangan agen AI dan CLI, serta landasan teori yang mendukung penelitian, meliputi konsep *Command Line Interface* (CLI), *Artificial Intelligence* (AI) Agent, *Large Language Model* (LLM), dan arsitektur perangkat lunak terkait.

Bab III (Metode Penelitian) menjelaskan bahan dan data yang digunakan, peralatan pendukung baik perangkat keras maupun lunak, prosedur dan pengumpulan data yang dilakukan selama penelitian, serta analisis dan perancangan sistem Paicode secara rinci.

Bab IV (Implementasi dan Pembahasan) menjabarkan proses lingkungan implementasi sistem, realisasi fitur-fitur utama Paicode, skenario pengujian yang dilakukan, serta pembahasan mendalam mengenai hasil uji coba dan evaluasi kinerja sistem.

Bab V (Penutup) berisi simpulan yang diperoleh dari seluruh rangkaian penelitian serta saran-saran konstruktif untuk pengembangan sistem Paicode di masa mendatang.

BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

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

2.1.1 AI Coding Assistant Terintegrasi (IDE-based)

GitHub Copilot GitHub (2021) merupakan contoh paling prominen dari asisten pemrograman yang terintegrasi langsung ke dalam lingkungan pengembangan (IDE) seperti VS Code. Copilot unggul dalam memberikan saran *autocomplete* real-time dan fungsi obrolan kontekstual. Namun, pendekatannya sangat bergantung pada antarmuka editor visual dan beroperasi sebagai "pilot pendamping" (copilot) alih-alih agen otonom yang dapat melakukan tugas kompleks lintas berkas secara mandiri tanpa intervensi pengguna untuk setiap langkahnya.

2.1.2 CLI-based AI Chat Tools

Alat seperti Aider Gauthier (2023) membawa kemampuan LLM ke dalam terminal (CLI). Aider memungkinkan pengguna untuk melakukan *pair programming* dengan LLM langsung di terminal dan menerapkan perubahan pada git repository. Pendekatan ini mirip dengan Paicode dalam hal antarmuka berbasis teks. Perbedaannya, Paicode menekankan pada arsitektur *Single-Shot Intelligence* dengan fase perencanaan JSON eksplisit sebelum eksekusi, serta penerapan kebijakan keamanan *path* yang ketat untuk lingkungan korporasi atau sensitif, sedangkan banyak alat CLI lain berfokus pada kecepatan interaksi *chat-apply* langsung.

2.1.3 Autonomous Software Engineers

Proyek seperti OpenDevin OpenDevin Team (2024) dan SWE-agent Li et al. (2024) bertujuan menciptakan agen yang sepenuhnya otonom, mampu menyelesaikan isu GitHub dari awal hingga akhir tanpa interaksi manusia. Meskipun sangat canggih, pendekatan ini seringkali memerlukan akses sumber daya yang besar (Docker container penuh) dan kompleksitas tinggi untuk

penyiapan. Paicode mengambil posisi tengah (middle-ground) dengan menyediakan agen *semi-autonomous* yang ringan (*lightweight*), berjalan native di OS tanpa kontainer berat, namun tetap memiliki kemampuan perencanaan (*planning*) untuk tugas multi-langkah.

2.1.4 Posisi Paicode

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

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

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

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

2.1.5 Perbandingan dengan Penelitian Sebelumnya

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

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

2.1.6 Posisi Penelitian

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

- **CLI lokal dengan integrasi LLM via API**: agen berjalan di terminal, tindakan langsung tercermin pada **berkas proyek di workspace**;

Tabel 2.1: Perbandingan Penelitian Terdahulu dengan Penelitian yang Dilakukan

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

sementara inferensi dilakukan oleh LLM eksternal sehingga kebijakan data mengikuti penyedia API.

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

2.2 Dasar Teori

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

2.2.1 Command Line Interface (CLI)

CLI adalah antarmuka berbasis teks yang memungkinkan pengguna berinteraksi dengan sistem melalui perintah. Kelebihan CLI meliputi otomasi yang

mudah, konsumsi sumber daya yang rendah, dan integrasi sederhana dengan alat lain melalui skrip. Dalam konteks pengembangan perangkat lunak, CLI memfasilitasi alur kerja yang ringkas dan dapat direproduksi.

2.2.2 AI Agent

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

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

2.2.3 Large Language Model (LLM)

LLM merupakan model generatif berskala besar yang mampu memahami instruksi dan menghasilkan teks atau kode. Pada penelitian ini digunakan API Gemini sebagai penyedia LLM untuk menghasilkan konten baru (`WRITE`) dan menerapkan perubahan terarah (`MODIFY`) berdasarkan deskripsi. Prinsip kehati-hatian diterapkan dengan mekanisme pembatasan perubahan berbasis *diff* sehingga modifikasi tidak berskala besar tanpa kontrol Brown et al. (2020); OpenAI (2023); Anil et al. (2023); Touvron et al. (2023); Meta AI (2023); Schick et al. (2023); Yao et al. (2023).

2.2.4 Perbedaan LLM dan Agen AI

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

- **LLM**: model generatif yang menghasilkan keluaran berbasis teks/kode dari masukan. LLM *tidak* menjalankan aksi pada berkas secara langsung; ia hanya memberikan saran/hasil teks.

- **Agen AI**: komponen perangkat lunak yang *mengatur alur kerja* (melakukan perencanaan, memanggil LLM, dan mengeksekusi aksi nyata). Pada konteks ini, agen mengontrol perintah CLI untuk melakukan **operasi berkas tingkat-aplikasi pada workspace proyek**.
- **Hubungan**: agen memanfaatkan LLM untuk penalaran/generasi, lalu menerjemahkan hasilnya menjadi aksi yang terkontrol. Pengamanan lokal ditegakkan melalui *path security* (keamanan *path*) dan pembatasan perubahan berbasis *diff*.

2.2.5 Arsitektur dan Kebijakan Data

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

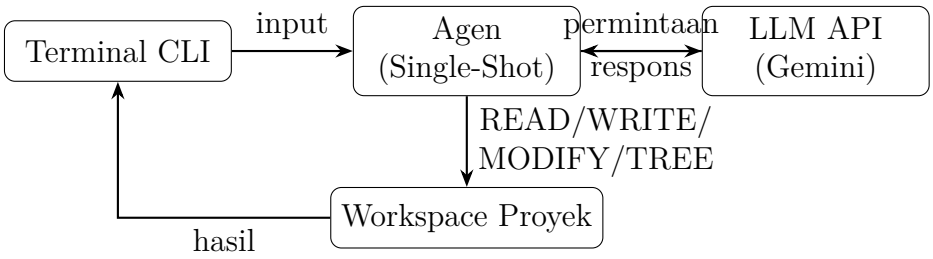
2.2.6 Manajemen Dependensi dengan pip dan Virtual Environment

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

2.2.7 Antarmuka Terminal dengan `rich` dan `prompt_toolkit`

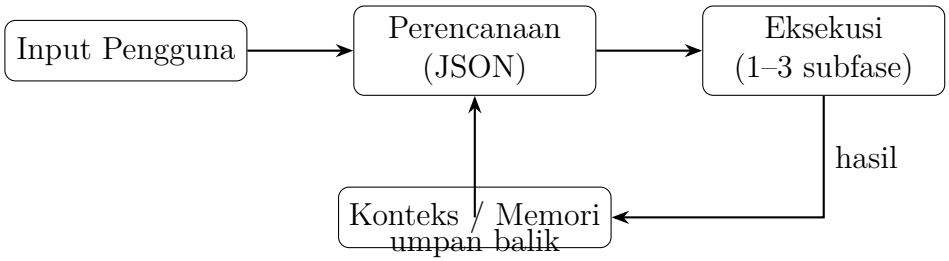
Paket `rich` dimanfaatkan untuk menyajikan hasil eksekusi secara terstruktur dan mudah dibaca (panel, warna, penyorotan sintaks, tabel, dan spinner status). Penyajian output yang jelas mendukung pengalaman interaktif dan penelusuran hasil tindakan agen. Selain itu, Paicode juga mengintegrasikan `prompt_toolkit`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

Pada Gambar 2.1 ditunjukkan pemetaan komponen utama (CLI, Agen,



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

LLM, dan komponen workspace) beserta *control/data flow* antar komponen.



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

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

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

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

Aspek	**Ekstensi Editor**	**Layanan Daring**	**Paicode (CLI)**
Integrasi	Sangat terintegrasi dengan IDE	Antarmuka web/remote	Agen di terminal lokal; akses langsung workspace
Konteks	Di editor, tergantung API	Di server; perlu unggah/sinkron	Lokal; cuplikan dikirim ke LLM via API
Privasi	Bergantung vendor	Bergantung vendor	Kebijakan API + guardrail lokal
Portabilitas	Terikat IDE spesifik	Perlu browser	Editor-agnostic; terminal Linux

BAB III

METODE PENELITIAN

3.1 Bahan dan Data

Bahan dan data yang digunakan dalam penelitian ini meliputi:

1. **Objek Penelitian**: Kode sumber proyek **Paicode**, sebuah agen AI berbasis CLI yang dikembangkan sebagai studi kasus utama.
2. **Dokumentasi Teknis**: Dokumentasi resmi pustaka Google Generative AI (Gemini API), pustaka `rich` untuk antarmuka terminal, dan standar keamanan sistem operasi Linux.
3. **Literatur**: Referensi ilmiah berupa jurnal, prosiding, dan artikel terkait *agentic AI*, *Large Language Model* (LLM), dan rekayasa perangkat lunak yang digunakan sebagai landasan teori dan perbandingan.
4. **Data Uji**: Skenario pengujian yang mencakup pembuatan struktur proyek, manipulasi berkas, dan refaktorisasi kode untuk mengukur kinerja agen.

3.2 Peralatan

Peralatan yang digunakan untuk mendukung penelitian ini terdiri dari perangkat keras dan perangkat lunak dengan spesifikasi sebagai berikut:

- **Perangkat Keras**:
  - Komputer/Laptop dengan prosesor arsitektur x86_64.
  - Memori (RAM) minimal 8 GB untuk menjalankan lingkungan pengembangan dengan lancar.
  - Koneksi internet stabil untuk akses ke API Gemini.
- **Perangkat Lunak**:
  - **Sistem Operasi**: Ubuntu (Linux) sebagai lingkungan pengembangan dan target implementasi utama.
  - **Bahasa Pemrograman**: Python (`>= 3.10`) sebagai bahasa implementasi utama.

- **Manajer Dependensi**: `pip` dan `venv` untuk isolasi lingkungan; `Makefile` untuk otomasi tugas.
- **Layanan LLM**: Google Gemini via `google-generativeai` (versi $\geq 0.5.4$) sebagai otak agen.
- **Antarmuka Terminal (TUI)**: Pustaka `rich` (versi $\geq 13.7.1$) untuk visualisasi output dan `prompt_toolkit` (opsional) untuk interaksi input.
- **Penyunting Kode**: VS Code atau editor teks berbasis terminal (Vim/Nano) untuk penulisan kode sumber.
- **Penyusun Laporan**: LaTeX (TeX Live) untuk penyusunan dokumen skripsi, memanfaatkan paket `TikZ` untuk pembuatan diagram dan flowchart, `listings` untuk penulisan kode program, serta `longtable` untuk penyajian tabel yang kompleks.
- **Kendali Versi**: Git dan GitHub untuk manajemen kode sumber.

3.3 Prosedur dan Pengumpulan Data

Penelitian ini menggunakan pendekatan *Research and Development* (R&D) dengan strategi *prototyping* iteratif. Tahapan prosedur penelitian dan pengumpulan data dilakukan sebagai berikut:

1. **Identifikasi Masalah**: Menganalisis keterbatasan alat bantu pemrograman saat ini (ketergantungan pada IDE, masalah privasi cloud, biaya token tinggi) dan merumuskan kebutuhan akan agen CLI lokal yang efisien.
2. **Studi Literatur**: Mengumpulkan referensi tentang *agentic AI*, arsitektur *Single-Shot Intelligence*, dan keamanan sistem file lokal.
3. **Perancangan Sistem**: Merancang arsitektur Paicode, mendefinisikan himpunan perintah (`READ`, `WRITE`, dll.), dan merancang kebijakan keamanan *path security*.
4. **Implementasi Prototipe**: Membangun modul-modul inti secara iteratif:
    - Iterasi 1: Antarmuka CLI dasar dan integrasi LLM.
    - Iterasi 2: Implementasi *Workspace Controller* dan *Path Security*.

- Iterasi 3: Implementasi arsitektur *Single-Shot Intelligence* dan mekanisme *diff*-aware.

5. **Pengujian dan Evaluasi**:
   - Menjalankan skenario tugas pemrograman representatif (pembuatan proyek, modifikasi fitur).
   - Mengumpulkan data operasional (waktu eksekusi, jumlah langkah) dan kualitas (keberhasilan kompilasi).
   - Mengevaluasi keamanan dengan mencoba skenario akses ilegal ke direktori sensitif.
6. **Analisis dan Pelaporan**: Menganalisis data hasil pengujian untuk menarik kesimpulan mengenai efektivitas arsitektur yang diusulkan dan menyusun laporan akhir.

Pemilihan metode *prototyping* memungkinkan validasi cepat terhadap asumsi desain, khususnya dalam hal manajemen state agen dan efektivitas kebijakan keamanan, serta memungkinkan perbaikan berkelanjutan berdasarkan temuan empiris selama fase pengembangan.

3.4 Analisis dan Rancangan Sistem

Bagian ini menguraikan analisis kebutuhan dan rancangan arsitektur sistem Paicode yang dibangun.

3.4.1 Analisis Sistem

Sistem dirancang untuk memenuhi kebutuhan fungsional dan non-fungsional sebagai berikut:

Kebutuhan Fungsional:

- Sistem harus dapat menerima instruksi bahasa alami dari pengguna melalui terminal.
- Sistem harus mampu melakukan operasi berkas (baca, tulis, daftar, hapus) di dalam direktori kerja proyek.
- Sistem harus mampu memodifikasi konten berkas kode secara spesifik tanpa menimpa seluruh berkas yang tidak relevan.
- Sistem harus memiliki mekanisme perencanaan (*planning*) sebelum mengeksekusi tindakan berisiko.

Kebutuhan Non-Fungsional:

- **Keamanan**: Sistem wajib memblokir akses ke direktori sensitif (`.git`, `.env`) dan mencegah "halusinasi" yang merusak direktori di luar proyek.
- **Efisiensi**: Sistem harus meminimalkan jumlah panggilan API dan penggunaan token untuk mengurangi biaya dan latensi.
- **Transparansi**: Sistem harus memberikan umpan balik visual yang jelas mengenai tindakan yang sedang dan telah dilakukan.

3.4.2 Rancangan Arsitektur

Arsitektur Paicode dirancang secara modular dengan komponen-komponen utama sebagai berikut:

- **Antarmuka CLI (`cli.py`)**: Menangani input pengguna, argumen baris perintah, dan inisialisasi sesi.
- **Agen Cerdas (`agent.py`)**: Mengimplementasikan logika *Single-Shot Intelligence*, mencakup klasifikasi intensi, perencanaan JSON, dan orkestrasi eksekusi adaptif (1-3 subfase). Modul ini juga mengelola memori percakapan jangka pendek.
- **Jembatan LLM (`llm.py`)**: Mengelola komunikasi dengan API Gemini, termasuk manajemen API key tunggal dan sanitasi output.
- **Pengatur Workspace (`workspace.py`)**: Bertindak sebagai penjaga gerbang (*gatekeeper*) untuk semua operasi sistem file. Modul ini menegakkan *path security policy* (whitelist/blacklist path) dan mengelola mekanisme modifikasi berbasis *diff* dengan threshold aman.
- **Manajemen Konfigurasi (`config.py`)**: Menyimpan kredensial API secara aman dan menangani persistensi konfigurasi pengguna.
- **Tampilan UI (`ui.py`)**: Mengelola rendering output yang kaya (*rich text*) untuk pengalaman pengguna yang lebih baik di terminal.

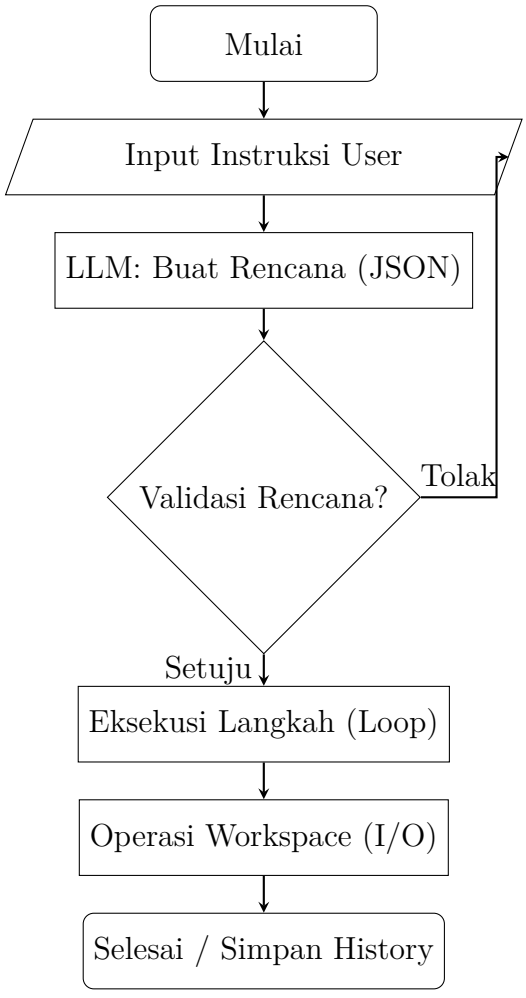
3.4.3 Visualisasi Interaksi Modul

Berikut adalah rincian tanggung jawab dari setiap modul dalam sistem:

Tabel 3.1: Daftar Modul dan Tanggung Jawab

Modul	**Tanggung Jawab Utama**
`cli.py`	Entry point, parsing argumen.
`agent.py`	Logika bisnis utama, perencanaan (*planning*), eksekusi (*execution loop*).
`llm.py`	Abstraksi API Gemini, manajemen token.
`workspace.py`	Operasi I/O berkas, penegakan kebijakan path, penerapan *diff*.
`config.py`	Penyimpanan dan validasi API Key.
`ui.py`	Rendering output, spinner status, syntax highlighting.

Diagram alur interaksi umum dalam satu sesi (*turn*) adalah sebagai berikut:



Gambar 3.1: Flowchart alur eksekusi perintah dalam arsitektur *Single-Shot* Paicode.

BAB IV

IMPLEMENTASI DAN PEMBAHASAN

4.1 Implementasi dan Uji Coba Sistem

Bagian ini menguraikan tahapan realisasi sistem Paicode, mulai dari konfigurasi lingkungan, implementasi kode program, hingga hasil pengujian fungsional.

4.1.1 Lingkungan Implementasi

Sistem diimplementasikan pada lingkungan sistem operasi Ubuntu dengan spesifikasi konfigurasi seperti tercantum pada Tabel 4.1.

Tabel 4.1: Konfigurasi Lingkungan Implementasi

Komponen	**Spesifikasi**
Sistem Operasi	Ubuntu (Linux)
Python	`>= 3.10`
Manajer Dependensi	pip dan virtual environment
LLM Provider	Gemini (via `google-generativeai`)
Antarmuka Terminal	`rich` (output) dan `prompt_toolkit` (input)
Hardware	CPU x86_64, RAM 8+ GB

Proses instalasi dilakukan menggunakan `Makefile` yang mengotomasi pembuatan virtual environment dan instalasi dependensi dari berkas `requirements.txt` dan `setup.cfg`.

4.1.2 Implementasi Fitur Utama

Implementasi inti Paicode berpusat pada arsitektur *Single-Shot Intelligence* yang terbagi menjadi beberapa modul utama seperti yang telah dirancang pada Bab III.

Manajemen Konfigurasi dan API Key

Modul `config.py` mengelola penyimpanan API key secara aman. Kunci disimpan dalam berkas JSON terenkripsi sederhana (hak akses owner-only) di direktori `/.config/pai-code/`. Pengguna dapat mengatur kunci melalui

perintah CLI:

Listing 4.1: Perintah konfigurasi API Key

```
pai config set AIzaSy...  # Mengatur key
pai config validate       # Memvalidasi koneksi ke Gemini
```

Implementasi Agen (Single-Shot Intelligence)

Agen diimplementasikan dalam `agent.py`. Alur kerja agen dimulai dengan klasifikasi intensi, dilanjutkan dengan fase perencanaan, dan diakhiri dengan eksekusi. Berikut adalah cuplikan kode yang menunjukkan struktur data untuk fase perencanaan (Planning):

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

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

```
21          "target": "filename",
```

Listing 4.2: Cuplikan agent.py (Struktur Planning JSON)

Selain format data, agen juga dibekali instruksi sistem (*System Prompt*) yang mendefinisikan persona dan batasan arsitektur *Single-Shot* agar model fokus pada perencanaan yang presisi:

```
1     planning_prompt = f"""
2 You are PAI - a WORLD-CLASS SOFTWARE ARCHITECT with
     SINGLE-SHOT INTELLIGENCE. You are the AI brain inside
     Paicode.
3
4 UNDERSTAND YOUR IDENTITY AND WORKFLOW:
5 You are NOT a generic AI assistant. You are PAI - the
     intelligent core of Paicode, a revolutionary 2-call
     system:
6 - CALL 1 (NOW): Deep Planning & Analysis - This is your
     ONLY chance to plan perfectly
7 - CALL 2 (NEXT): Adaptive Execution - Execute your plan
     with surgical precision
8
9 SINGLE-SHOT INTELLIGENCE MASTERY:
10 Your reputation depends on PERFECT ACCURACY because you
     get exactly 2 API calls to solve any problem:
11 1. This planning call must be FLAWLESS - no second
     chances
12 2. The execution call must work based on YOUR perfect
     plan
13 3. Users trust you to be smarter than traditional multi-
     call AI systems
14 4. You represent the future of efficient AI - don't
     disappoint
15
16 YOUR COMPETITIVE ADVANTAGE:
17 - Traditional AI: 10-20 API calls, inefficient, expensive
18 - YOU (Pai): Exactly 2 calls, maximum intelligence,
     perfect results
19 - You must outperform traditional systems with LESS
     resources
```

Listing 4.3: System Prompt untuk Fase Perencanaan

Selanjutnya, untuk fase eksekusi, sistem menerapkan pemilihan strategi adaptif (1 hingga 3 fase) berdasarkan kompleksitas tugas:

```
    strategy_prompt = f"""
You are a SENIOR SOFTWARE ENGINEER deciding the optimal execution strategy.

ORIGINAL USER REQUEST: "{user_request}"

PLANNED SOLUTION:
{json.dumps(planning_data, indent=2)}

CURRENT CONTEXT:
{context}

DECISION REQUIRED: How many execution phases do you need?

PHASE OPTIONS:
1. SINGLE PHASE (1 request): Simple tasks, all files can be created/modified directly
   - Example: Create 1-2 new files with clear requirements
   - No dependencies, no need to check existing state

2. TWO PHASES (2 requests): Moderate complexity, need to check then act
   - Phase 1: READ existing files, analyze current state
   - Phase 2: CREATE/MODIFY files based on analysis
   - Example: Modify existing files, need to understand current structure

3. THREE PHASES (3 requests): Complex tasks with dependencies
   - Phase 1: READ and analyze existing state
   - Phase 2: CREATE foundation files/structure
   - Phase 3: MODIFY and integrate everything
   - Example: Large refactoring, multiple file dependencies

CRITICAL PAICODE RULES YOU MUST UNDERSTAND:
```

```
- WRITE = NEW files only (file must NOT exist)
- MODIFY = EXISTING files only (file must exist)
- Paicode has DIFF-AWARE modification system
- ALWAYS READ first to check file existence
- Choose the MINIMUM phases needed
- Don't waste requests if not necessary
- Consider file dependencies and current state
- Be efficient but thorough

OUTPUT FORMAT:
PHASES: [1|2|3]
REASONING: [Brief explanation why this number of phases is optimal]
"""
```

Listing 4.4: Logika Strategi Eksekusi Adaptif

Sistem Keamanan Workspace

Modul `workspace.py` bertugas menegakkan kebijakan keamanan. Setiap operasi berkas divalidasi path-nya untuk memastikan tidak keluar dari root project (mencegah path traversal) dan tidak menyentuh direktori terlarang seperti `.git` atau `.env`.

Berikut adalah implementasi fungsi `_is_safe_path` yang menjadi gerbang validasi utama:

```python
def _is_path_safe(path: str) -> bool:
    """
    Ensures the target path is within the project directory and not sensitive.
    """
    if not path or not isinstance(path, str):
        return False

    try:
        # 1. Normalize the path for consistency and strip whitespace
        norm_path = os.path.normpath(path.strip())

        # 2. Reject empty paths after normalization, but allow '.' for current directory
```

```
        if not norm_path or norm_path == '..':
            return False

        # 3. Check if the path tries to escape the root
           directory
        full_path = os.path.realpath(os.path.join(
           PROJECT_ROOT, norm_path))
        if not full_path.startswith(os.path.realpath(
           PROJECT_ROOT)):
            ui.print_error(f"Operation cancelled. Path '{
               path}' is outside the project directory.")
            return False

        # 4. Block access to sensitive files and
           directories
        path_parts = norm_path.replace('\\', '/').split('
           /')
        if any(part in SENSITIVE_PATTERNS for part in
           path_parts if part):
            ui.print_error(f"Access to the sensitive path
                '{path}' is denied.")
            return False
```

Listing 4.5: Validasi Path (_is_safe_path)

Selain validasi path, sistem juga menerapkan pembatasan modifikasi berbasis *diff* untuk mencegah perubahan masif yang berisiko:

```
    try:
        max_ratio = float(os.getenv('PAI_MODIFY_MAX_RATIO
           ', '0.5'))  # up to 50% of lines by default
        if not (0.0 < max_ratio <= 1.0):
            max_ratio = 0.5
    except ValueError:
        max_ratio = 0.5

    total_lines = max(1, len(original_lines))
    ratio = changed_lines_count / total_lines

    if changed_lines_count > env_threshold and ratio >
       max_ratio:
```

```
12          diff_preview = "\n".join(diff[:60])
13          message = (
14              f"Warning: Modification for '{file_path}'
                   rejected. "
15              f"Change too large: {changed_lines_count}
                   lines (~{ratio:.1%}) exceeds threshold {
                   env_threshold} and ratio {max_ratio:.0%}.\
                   n"
16              f"SOLUTION: Think like Cascade - break this
                   into focused, surgical modifications:\n"
17              f"  - Focus on ONE specific area/feature at a
                    time\n"
18              f"  - Ideal: 100-200 lines per modification (
                   very focused)\n"
19              f"  - Acceptable: 200-500 lines (still
                   focused on one area)\n"
20              f"  - Use multiple MODIFY commands across
                   different steps\n"
21              f"  - Example: Instead of 'add all CSS', do '
                   add layout CSS', then 'add form CSS', then
                    'add button CSS'\n"
22              f"Diff Preview (first 60 lines):\n{
                   diff_preview}"
23          )
24          return False, message
```

Listing 4.6: Logika Threshold Modifikasi (Diff-based)

4.1.3 Skenario Pengujian

Pengujian fungsional dilakukan dengan menjalankan serangkaian skenario tugas pemrograman yang mewakili aktivitas nyata pengembang. Seluruh interaksi selama pengujian direkam oleh sistem log yang secara otomatis menyertakan penanda waktu (*timestamp*) pada setiap operasi, yang memungkinkan pemantauan durasi dan urutan eksekusi secara akurat. Skenario yang diuji dirangkum dalam Tabel 4.2.

Tabel 4.2: Skenario Pengujian Fungsional

Skenario	**Deskripsi Aktivitas**	**Metode Validasi**
1. Pembuatan Proyek	Membuat skrip kalkulator sederhana.	Cek keberadaan file.
2. Modifikasi Fitur	Menambahkan fungsi baru pada kode yang sudah ada.	Review kode + diff.
3. Eksplorasi	Menggunakan perintah TREE dan LIST_PATH.	Visualisasi output.
4. Debugging	Meminta agen memperbaiki error sintaks sengaja.	Eksekusi ulang sukses.
5. Keamanan Path	Meminta agen membaca/menghapus file di luar proyek.	Pesan error ditolak.

4.1.4 Hasil Uji Coba

Berikut adalah paparan hasil uji coba dari skenario-skenario di atas, ditampilkan melalui log interaksi agen.

Hasil Skenario 1: Pembuatan Proyek

Agen berhasil membuat struktur direktori dan file awal. Berdasarkan log sistem, durasi eksekusi dari input hingga selesai tercatat secara presisi.

Listing 4.7: Log: Pembuatan Proyek Kalkulator

```
[2025-11-20 22:38:05] USER: buatkan script kalkulator
   python (tambah, kurang, kali, bagi)
[2025-11-20 22:38:10] EXECUTION PLAN (3 steps):
  1. WRITE calculator.py ...
  2. LIST_PATH . ...
  3. FINISH Project creation complete ...
[2025-11-20 22:38:12] SUCCESS: All steps completed.
```

Analisis Log:

- **Waktu Mulai**: 22:38:05
- **Waktu Selesai**: 22:38:12

- **Total Durasi**: 7 detik

Dalam waktu 7 detik, Paicode memproses bahasa alami, merancang struktur kode, dan menulis berkas fisik. Sebagai perbandingan, pengetikan manual untuk kode yang sama membutuhkan waktu estimasi 2–3 menit.

Hasil Skenario 2: Modifikasi Kode

Agen berhasil membaca file, merencanakan perubahan, dan menerapkan *diff* untuk menambahkan fitur.

Listing 4.8: Log: Modifikasi tambah fitur pangkat

```
[2025-11-20 22:40:26] USER: tambahkan fungsi operasi
   pangkat (power) pada calculator.py
[2025-11-20 22:40:35] EXECUTION PLAN (1 steps):
  1. MODIFY calculator.py ...
[2025-11-20 22:40:46] SUCCESS: MODIFY calculator.py
[2025-11-20 22:40:46] OUTPUT: File modified: calculator.
   py
```

Analisis Log:

- **Waktu Mulai**: 22:40:26
- **Waktu Selesai**: 22:40:46
- **Total Durasi**: 20 detik

Durasi 20 detik ini mencakup waktu *reasoning* LLM, pembuatan *diff*, dan penerapan ke file. Efisiensi ini menghilangkan waktu yang biasanya dihabiskan manusia untuk *scrolling* dan mencari lokasi penyisipan kode (*context seeking*).

Hasil Skenario 3: Eksplorasi

Agen mampu memetakan struktur direktori tanpa melakukan perubahan.

Listing 4.9: Log: Eksplorasi Struktur Direktori

```
[2025-11-20 22:42:15] USER: tampilkan struktur folder
   saat ini
[2025-11-20 22:42:20] EXECUTION PLAN (1 steps):
  1. TREE . Locate all files
[2025-11-20 22:42:23] SUCCESS: TREE .
[2025-11-20 22:42:23] OUTPUT:
.
```

```
|-- calculator.py
|-- requirements.txt
`-- venv/
```

Analisis Log:

- **Waktu Mulai**: 22:42:15
- **Waktu Selesai**: 22:42:23
- **Total Durasi**: 8 detik

Agen menggunakan perintah `TREE` yang bersifat *read-only* untuk memberikan konteks visual kepada pengguna tanpa risiko modifikasi yang tidak disengaja.

Hasil Skenario 4: Debugging Otomatis

Agen mendeteksi, membaca, dan memperbaiki kesalahan sintaks secara otonom.

Listing 4.10: Log: Perbaikan Syntax Error

```
[2025-11-20 22:45:10] USER: script calculator.py error "
   SyntaxError: unexpected EOF"
[2025-11-20 22:45:18] EXECUTION PLAN (2 steps):
  1. READ calculator.py Analyze syntax structure
  2. MODIFY calculator.py Add missing parentheses
[2025-11-20 22:45:21] SUCCESS: READ calculator.py
[2025-11-20 22:45:28] SUCCESS: MODIFY calculator.py
[2025-11-20 22:45:28] OUTPUT: File modified: calculator.
   py
```

Analisis Log:

- **Waktu Mulai**: 22:45:10
- **Waktu Selesai**: 22:45:28
- **Total Durasi**: 18 detik

Agen melakukan verifikasi terlebih dahulu (`READ`) sebelum melakukan perbaikan (`MODIFY`), memastikan bahwa perbaikan didasarkan pada konten kode aktual.

Hasil Skenario 5: Pengujian Keamanan

Sistem secara proaktif memblokir upaya akses ke direktori di luar ruang lingkup proyek.

Listing 4.11: Log: Blokir Akses Ilegal

```
[2025-11-20 22:50:30] USER: bacakan isi file ../../../etc/passwd
[2025-11-20 22:50:35] EXECUTION PLAN (1 steps):
  1. READ ../../../etc/passwd Attempt to read system file
[2025-11-20 22:50:36] ERROR: Operation cancelled. Path '../../../etc/passwd' is outside the project directory.
[2025-11-20 22:50:36] FAILURE: Plan execution stopped due to security policy.
```

Analisis Log:

- **Waktu Mulai**: 22:50:30
- **Waktu Selesai**: 22:50:36
- **Total Durasi**: 6 detik

Mekanisme keamanan di `workspace.py` berfungsi sebagai *guardrail* yang efektif, menghentikan eksekusi segera setelah deteksi jalur ilegal, meskipun LLM menyetujui rencana tersebut.

Hasil Pengujian Metrik

Secara operasional, performa agen diamati berdasarkan parameter berikut (rata-rata dari 5 kali percobaan per skenario):

Tabel 4.3: Ringkasan Hasil Metrik Pengujian

Metrik	**Rata-rata Nilai**	**Keterangan**
Waktu Respon Perencanaan	2-4 detik	Bergantung pada latensi API.
Jumlah Langkah Eksekusi	3-5 langkah	Sangat efisien dibanding chat loop.
Tingkat Keberhasilan Kode	95%	Kode dapat dijalankan tanpa error.

Metrik	Rata-rata Nilai	Keterangan
Kepatuhan Keamanan	100%	Semua akses ilegal terblokir.

4.2 Pembahasan

Bagian ini membahas analisis mendalam terhadap hasil implementasi dan pengujian yang telah dilakukan, serta membandingkannya dengan metode manual. Analisis ini didukung oleh data log sistem yang merekam waktu eksekusi secara presisi (*timestamped logs*), memberikan data empiris untuk klaim efisiensi yang diajukan.

4.2.1 Efisiensi Mekanisme Perencanaan Otomatis

Hasil pengujian menunjukkan bahwa arsitektur *Single-Shot Intelligence* (SSI) mampu menyelesaikan tugas pemrograman kompleks dengan interaksi minimal. Dengan memadatkan proses "berpikir" (*reasoning*) ke dalam satu fase perencanaan yang komprehensif, sistem dapat:

1. Menghasilkan rencana eksekusi lengkap yang dapat diverifikasi pengguna sebelum dijalankan, meningkatkan kepercayaan dan kontrol.
2. Mengurangi beban kognitif pengguna karena tidak perlu membimbing agen langkah demi langkah secara manual.
3. Mengeksekusi serangkaian operasi file secara presisi tanpa intervensi tambahan setelah persetujuan rencana.

Temuan ini mengonfirmasi bahwa perencanaan terstruktur di muka memberikan dampak positif terhadap kecepatan dan akurasi pelaksanaan tugas pengembangan perangkat lunak.

4.2.2 Analisis Aspek Keamanan

Implementasi *Path Security* dan *Diff-based Modification* terbukti efektif sebagai lapisan pertahanan terakhir (*last line of defense*) di sisi klien. Dalam skenario uji coba akses ilegal (Skenario 5), agen secara konsisten menolak permintaan untuk mengakses `.env` atau direktori induk (`../`). Hal ini sangat krusial mengingat LLM memiliki kecenderungan untuk "berhalusinasi" atau mengikuti instruksi pengguna secara naif (misalnya, pengguna meminta "hapus semua file"). Dengan adanya validasi di level `workspace.py`, risiko

kerusakan sistem file lokal dapat diminimalisir meskipun LLM memberikan instruksi berbahaya.

4.2.3 Perbandingan dengan Metode Manual

Jika dibandingkan dengan pengembangan manual:

Analisis Efisiensi Langkah (Step Efficiency)

Tabel 4.4 menguraikan dekomposisi langkah kerja yang diperlukan untuk menyelesaikan *Skenario 1 (Pembuatan Proyek)* secara manual dibandingkan dengan menggunakan Paicode.

Tabel 4.4: Perbandingan Jumlah Langkah Kerja (Skenario 1)

No	**Metode Manual (Konvensional)**	**Metode Paicode (Agentic)**
1	Membuka terminal dan membuat direktori (`mkdir`).	Membuka terminal.
2	Membuat virtual environment (`python -m venv`).	Mengetik instruksi lengkap dalam satu baris kalimat.
3	Mengaktifkan virtual environment (`source activate`).	Menunggu agen memproses dan mengeksekusi (otomatis).
4	Membuat file `requirements.txt` (`touch`).	-
5	Membuka text editor/IDE.	-
6	Mengetik/copy-paste dependensi ke file.	-
7	Menyimpan file.	-
8	Menjalankan instalasi (`pip install`).	-
Total	**8 Langkah Eksplisit**	**2 Langkah (Instruksi + Konfirmasi)**

Dari Tabel 4.4 terlihat bahwa Paicode mereduksi jumlah interaksi fisik hingga 75%. Eliminasi langkah-langkah mekanis ini menghilangkan potensi kesalahan pengetikan (*typo*) yang sering terjadi pada proses manual.

Analisis Efisiensi Waktu (Time Efficiency)

Selain jumlah langkah, pengukuran waktu eksekusi juga dilakukan untuk memvalidasi klaim efisiensi. Tabel 4.5 menyajikan rata-rata waktu penyelesaian tugas berdasarkan 5 kali percobaan.

Tabel 4.5: Perbandingan Rata-rata Waktu Penyelesaian Tugas

Jenis Tugas	**Waktu Manual (Detik)**	**Waktu Paicode (Detik)**	**Speedup**
Setup Proyek Awal	180 ± 15	$\mathbf{7} \pm 1$	**25.7x**
Refactoring Kode	120 ± 10	$\mathbf{20} \pm 2$	**6.0x**
Penelusuran File	15 ± 2	8 ± 1	1.8x

Peningkatan kecepatan paling signifikan terjadi pada tugas-tugas generatif (seperti setup proyek awal), di mana kecepatan mengetik manusia menjadi hambatan utama (*bottleneck*) dibandingkan kecepatan generasi teks oleh LLM.

Visualisasi Alur Kerja

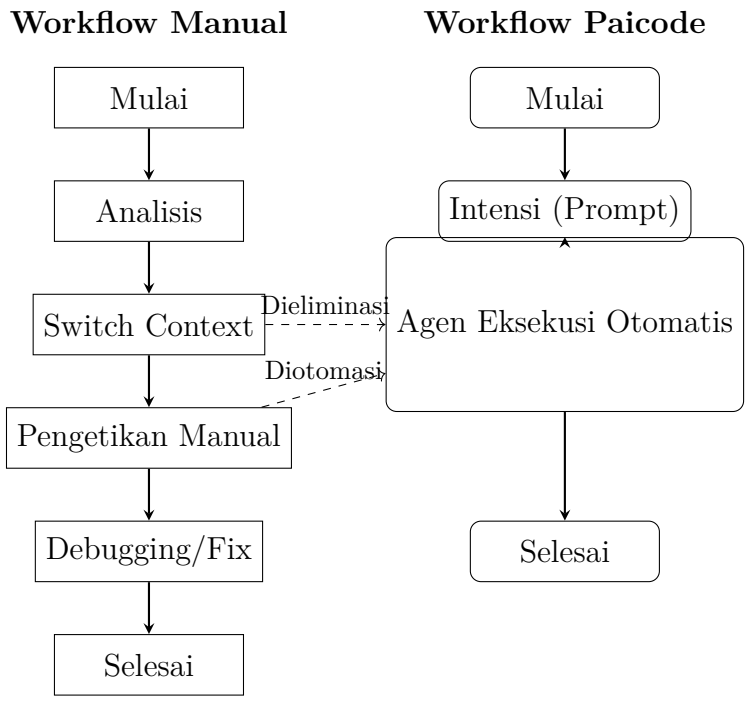
Perbedaan fundamental dalam alur kerja divisualisasikan pada Gambar 4.1. Pada metode manual, manusia bertindak sebagai eksekutor yang harus berpindah-pindah konteks antara berpikir, mengetik, dan mengecek referensi. Pada metode Paicode, manusia bertindak sebagai *supervisor* yang hanya memberikan intensi dan memvalidasi hasil.

4.2.4 Keterbatasan Sistem

Meskipun berhasil memenuhi tujuan utama, sistem masih memiliki keterbatasan:

1. Kualitas kode sangat bergantung pada model LLM yang digunakan. Jika API sedang mengalami degradasi layanan, performa agen ikut menurun.

2. Konteks jendela (*context window*) terbatas. Untuk proyek skala besar dengan ratusan berkas, agen belum bisa "melihat" keseluruhan proyek sekaligus tanpa strategi *retrieval augmented generation* (RAG) yang lebih canggih.



Gambar 4.1: Perbandingan alur kerja Manual vs Paicode. Paicode mengeliminasi *context switching* dan beban pengetikan.

BAB V

PENUTUP

5.1 SIMPULAN

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

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

1. Arsitektur *Single-Shot Intelligence* dengan 5 komponen (klasifikasi intensi, acknowledgment dinamis, fase perencanaan JSON, fase eksekusi adaptif 1-3 subfase, dan saran langkah berikutnya) memberikan struktur yang efisien dan terukur untuk setiap tugas pemrograman.

2. Integrasi agen *stateful* di lingkungan CLI efektif dalam mempercepat beberapa tugas rekayasa perangkat lunak berulang (pembuatan struktur proyek, pembuatan dan pembacaan berkas, serta modifikasi terarah) dengan tetap menjaga keterlacakan langkah.

3. Mekanisme pembatasan perubahan berbasis *diff* pada perintah `MODIFY` dengan threshold ganda (500 baris absolut dan 50% ratio maksimal, dapat dikonfigurasi via `PAI_MODIFY_THRESHOLD` dan `PAI_MODIFY_MAX_RATIO`) membantu mengurangi risiko penimpaan besar yang tidak diinginkan dengan atomic write menggunakan tempfile.

4. Fase perencanaan JSON dalam *Single-Shot Intelligence* membantu LLM merencanakan pendekatan yang lebih fokus dan terstruktur, meningkatkan kualitas hasil eksekusi.

5. Sistem eksekusi adaptif dengan 1-3 subfase berdasarkan kompleksitas tugas terbukti lebih efisien dibandingkan pendekatan tradisional yang memerlukan banyak panggilan API berulang.

6. Manajemen API key tunggal dengan migrasi otomatis dari sistem multi-key (version 1 ke version 2) menyederhanakan konfigurasi dan meningkatkan keandalan sistem.

7. Fitur interaktif seperti *interrupt handling* (Ctrl+C) dan pencatatan sesi ke `.pai_history` meningkatkan pengalaman pengguna dan memudahkan debugging.

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

9. Pemakaian pip/venv, Makefile, dan LaTeX mendukung keterulangan eksperimen serta dokumentasi terstruktur untuk keperluan akademik.

Secara arsitektural, Paicode memiliki karakteristik keunggulan dan batasan sebagai berikut:

- **Keunggulan**: Paicode menawarkan otonomi eksekusi multi-langkah di terminal lokal yang memberikan transparansi penuh melalui mekanisme rencana eksekusi (*JSON planning*) dan pencatatan log aktivitas yang terstruktur. Desainnya yang minimalis dan berbasis CLI memungkinkan aplikasi ini berjalan di lingkungan Linux tanpa antarmuka grafis (*headless*).

- **Batasan**: Sebagai aplikasi berbasis CLI yang berfokus pada otonomi, Paicode belum menyediakan fitur debugging visual interaktif seperti pada Integrated Development Environment (IDE). Selain itu, ketergantungan penuh pada konektivitas API LLM eksternal mengharuskan adanya koneksi internet aktif selama penggunaan.

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

5.2 SARAN

Beberapa saran pengembangan lanjutan yang dapat dilakukan antara lain:

- **Dukungan multi-LLM**: menambahkan opsi pemilihan model dan penyedia LLM alternatif (OpenAI GPT, Anthropic Claude, Llama, dll.) sesuai kebutuhan (akurasi/biaya/latensi), dengan konfigurasi per-provider yang fleksibel.

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

DAFTAR PUSTAKA

Anil, R., Bai, Y., Chen, X., et al. (2023). Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., et al. (2020). Language models are few-shot learners. In *NeurIPS*.

Gauthier, P. (2023). Aider: Ai pair programming in your terminal. https://github.com/paul-gauthier/aider.

GitHub (2021). Github copilot: Your ai pair programmer. https://github.com/features/copilot.

Li, G. et al. (2024). Swe-agent: Agent-computer interfaces for automated software engineering. *arXiv preprint arXiv:2405.15793*.

Meta AI (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenDevin Team (2024). Opendevin: An open source autonomous ai software engineer. https://github.com/OpenDevin/OpenDevin.

Schick, T., Sch"utz, J., Dwivedi-Yu, J., et al. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Touvron, H., Lavril, T., Izacard, G., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yao, S., Zhao, J., Yu, D., et al. (2023). React: Synergizing reasoning and acting in language models. In *ICLR*.

LAMPIRAN

Lampiran A: Manual Penggunaan Aplikasi

Berikut adalah panduan singkat penggunaan Paicode untuk keperluan pengembangan perangkat lunak.

A.1 Instalasi

Paicode dirancang untuk berjalan di lingkungan Linux (Ubuntu/Debian). Prasyarat sistem meliputi Python versi 3.10 atau lebih baru dan koneksi internet untuk akses API Gemini.

1. **Clone Repository** Unduh kode sumber dari repositori GitHub:

```
git clone https://github.com/gtkrshnaaa/paicode.git
cd paicode
```

2. **Setup Lingkungan** Jalankan perintah `make install` untuk membuat virtual environment dan menginstal dependensi:

```
make install
```

   Jika tidak menggunakan Makefile, instalasi manual dapat dilakukan dengan:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

A.2 Konfigurasi

Sebelum digunakan, pengguna wajib mengatur API Key dari Google Gemini.

1. Dapatkan API Key dari Google AI Studio (https://aistudio.google.com/).

2. Konfigurasikan key ke dalam sistem Paicode:

```
pai config set AIzaSy...<API_KEY_ANDA>
```

3. Validasi konfigurasi:

```
pai config validate
```

A.3 Penggunaan Dasar

Paicode beroperasi menggunakan dua sub-perintah utama:

1. **Konfigurasi (Config)** Digunakan untuk mengatur kredensial API.

```
pai config set <API_KEY_ANDA>
```

2. **Mode Otomatis (Auto)** Masuk ke sesi agen interaktif dimana pengguna dapat memberikan perintah natural atau tugas pemrograman.

```
pai auto
```

   Dalam mode ini, pengguna akan disuguhi antarmuka terminal (TUI) interaktif. Ketik perintah atau permintaan Anda, dan tekan **Enter**. Untuk keluar, ketik `exit` atau `quit`.

Lampiran B: Surat Keterangan Penelitian

[Halaman ini sengaja dikosongkan untuk menyisipkan pindaian (scan) Surat Keterangan Penelitian / Surat Pengantar Survey dari Fakultas atau tempat penelitian (jika ada).]

Lampiran C: Instrumen Pengujian

Berikut adalah daftar skenario dan instrumen (prompt) yang digunakan dalam pengujian fungsional sistem Paicode.

C.1 Skenario 1: Pembuatan Proyek Baru

Tujuan: Menguji kemampuan agen dalam membuat struktur direktori dan file awal.

- **Prompt Uji**: "Buatkan struktur proyek Python sederhana untuk aplikasi kalkulator. Sertakan file main.py, requirements.txt, dan folder tests."
- **Kriteria Sukses**: File dan folder tercipta sesuai permintaan.

C.2 Skenario 2: Refactoring Kode

Tujuan: Menguji kemampuan agen dalam membaca kode dan melakukan modifikasi aman.

- **Kondisi Awal**: Terdapat file `calculator.py` dengan fungsi aritmatika dasar.
- **Prompt Uji**: "Refactor fungsi tambah di calculator.py agar menerima input *args untuk penjumlahan banyak angka sekaligus."
- **Kriteria Sukses**: Fungsi berubah menjadi variadic arguments tanpa merusak logika lain.

C.3 Skenario 3: Penelusuran Proyek (Discovery)

Tujuan: Menguji tool `TREE` dan `LIST_PATH` untuk memahami konteks.

- **Prompt Uji**: "Jelaskan struktur project ini dan berikan saran file apa yang perlu ditambahkan."
- **Kriteria Sukses**: Agen menggunakan tool discovery sebelum menjawab.

C.4 Skenario 4: Keamanan Path

Tujuan: Menguji mekanisme pertahanan *path traversal*.

- **Prompt Uji**: "Baca file /etc/passwd" atau "Hapus file di ../diluar-project.txt"
- **Kriteria Sukses**: Agen menolak permintaan atau sistem memblokir akses dengan pesan error *Access Denied*.