Paicode: Agentic AI berbasis CLI untuk membantu proses coding secara interaktif ditenagai LLM eksternal via API

I PUTU GEDE GILANG TEJA KRISHNA
225410001

INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA
Angkatan 2022

2025

Lembar Pengesahan

Halaman ini berisi pengesahan skripsi oleh dosen pembimbing dan penguji sesuai format kampus. Silakan sesuaikan isi, penandatangan, tanggal, dan stempel sesuai ketentuan.

Pernyataan Keaslian

Saya yang bertanda tangan di bawah ini, menyatakan bahwa skripsi ini adalah hasil karya sendiri dan tidak memuat karya orang lain yang pernah diajukan untuk memperoleh gelar akademik di perguruan tinggi manapun, kecuali bagian-bagian tertentu yang dirujuk sebagaimana tercantum dalam daftar pustaka.

Yogyakarta, ...

I PUTU GEDE GILANG TEJA KRISHNA

NIM: 225410001

Kata Pengantar

Puji syukur ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan baik. Ucapan terima kasih disampaikan kepada semua pihak yang telah membantu dalam penyusunan skripsi ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan untuk perbaikan di masa mendatang.

Yogyakarta, ...

I PUTU GEDE GILANG TEJA KRISHNA

Ucapan Terima Kasih

Penulis menyampaikan terima kasih kepada orang tua, keluarga, dosen pembimbing, penguji, rekan-rekan, dan semua pihak yang telah memberikan dukungan moral maupun material sehingga skripsi ini dapat diselesaikan.

Yogyakarta, ...

I PUTU GEDE GILANG TEJA KRISHNA

Abstrak

Penelitian ini mengusulkan **Paicode**, sebuah agen AI berbasis Command Line Interface (CLI) untuk membantu proses pengembangan perangkat lunak secara interaktif. Sistem menerapkan prinsip *local-first* dengan kebijakan keamanan jalur, serta memanfaatkan layanan LLM (Gemini) hanya untuk kebutuhan inferensi. Himpunan perintah yang disediakan (mis. `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`) memungkinkan agen mengobservasi proyek, memanipulasi berkas, dan memodifikasi kode secara terarah.

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

Kata kunci: agentic AI, CLI, local-first, LLM, pengembangan perangkat lunak.

Abstract

This thesis presents **Paicode**, an agentic AI for Command Line Interface (CLI) that assists software development through interactive, stateful workflows. The system follows a *local-first* design with path-security policies, and leverages an external LLM (Gemini) solely for inference. A compact set of commands (e.g., `READ`, `WRITE`, `MODIFY`, `TREE`, `LIST_PATH`) enables the agent to observe the project, manipulate files, and apply targeted code modifications.

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

Keywords: agentic AI, CLI, local-first, LLM, software engineering.

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 tanpa mengorbankan prinsip *local-first*. Pendekatan *local-first* memusatkan kendali pada mesin pengguna sehingga interaksi, konteks, dan perubahan berkas terjadi secara lokal, sementara panggilan LLM eksternal hanya dilakukan sebatas kebutuhan inferensi [2, 1].

Penelitian ini menghadirkan **Paicode**, sebuah agen AI berbasis CLI yang dirancang untuk membantu proses pengembangan perangkat lunak secara interaktif. Paicode mampu: (i) mengobservasi struktur proyek (`TREE`, `LIST_PATH`); (ii) membaca dan menulis berkas (`READ`, `WRITE`); (iii) memodifikasi kode secara terarah dengan batasan perubahan berbasis diff (`MODIFY`); serta (iv) menegakkan kebijakan keamanan sistem berkas (memblokir akses ke direktori sensitif seperti `.git`, `venv`, dan `.env`). Sistem diimplementasikan pada lingkungan Ubuntu dengan bahasa pemrograman Python, pengelolaan dependensi melalui Poetry, dan menggunakan API Gemini sebagai LLM.

1.2 Rumusan Masalah

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

1. Bagaimana merancang arsitektur agen AI berbasis CLI yang *stateful*, *proactive*, dan menjunjung prinsip *local-first* untuk mendukung proses pengembangan perangkat lunak?

2. Bagaimana mengintegrasikan kemampuan observasi proyek, manipulasi berkas, serta modifikasi kode terarah berbasis deskripsi pengguna dengan pengamanan terhadap jalur dan direktori sensitif?

3. Bagaimana mengevaluasi efektivitas Paicode dalam membantu tugas-tugas pemrograman dibandingkan proses manual atau alat pembanding yang relevan?

1.3 Batasan Masalah

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

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

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah membangun dan mengevaluasi sebuah agen AI berbasis CLI yang dapat membantu pengembang dalam proses pemrograman secara interaktif. Secara khusus, penelitian menargetkan:

1. Merancang arsitektur Paicode yang mencakup modul agen, jembatan LLM, antarmuka CLI, lapisan keamanan sistem berkas, serta komponen tampilan terminal.

2. Mengimplementasikan kemampuan observasi proyek, manipulasi berkas, dan modifikasi kode terarah dengan mekanisme *patch/diff* untuk membatasi ruang perubahan.

3. Menyusun prosedur evaluasi dengan skenario tugas pemrograman yang representatif dan mengukur peningkatan produktivitas atau kualitas hasil.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini meliputi:

- **Akademis:** menyediakan studi kasus dan arsitektur rujukan untuk pengembangan agen AI *local-first* di lingkungan CLI, 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 sistem berkas; 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), paradigma *local-first*, serta perangkat bantu yang digunakan seperti Poetry untuk manajemen dependensi dan `rich` untuk antarmuka terminal.

2.1.1 Command Line Interface (CLI)

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

2.1.2 AI Agent

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

2.1.3 Large Language Model (LLM)

LLM merupakan model generatif berskala besar yang mampu memahami instruksi dan menghasilkan teks atau kode. Pada penelitian ini digunakan API Gemini sebagai penyedia LLM untuk menghasilkan konten baru (`WRITE`) dan menerapkan perubahan terarah (`MODIFY`) berdasarkan deskripsi. Prinsip kehati-hatian diterapkan dengan mekanisme

pembatasan perubahan berbasis *diff* sehingga modifikasi tidak berskala besar tanpa kontrol [2, 1].

2.1.4 Local-First Software

Paradigma *local-first* menempatkan mesin pengguna sebagai pusat kendali: file, struktur proyek, dan logik eksekusi utama berada secara lokal. Panggilan ke layanan eksternal (LLM) dilakukan seminimal mungkin dan tidak menyimpan konteks proyek di luar mesin pengguna. Pendekatan ini relevan untuk kebutuhan privasi, kepemilikan data, dan ketahanan terhadap jaringan.

2.1.5 Manajemen Dependensi dengan Poetry

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

2.1.6 Antarmuka Terminal dengan `rich`

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

2.2 Penelitian Terkait

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

Sebaliknya, pendekatan *local-first* pada Paicode menempatkan agen di lingkungan CLI dan beroperasi langsung pada sistem berkas. Perintah agen disederhanakan ke dalam himpunan tindakan yang eksplisit (`MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`) dengan *policy* keamanan jalur. 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:

- **Local-first CLI**: agen berjalan di terminal, tindakan langsung tercermin di sistem berkas, dan tidak bergantung pada editor tertentu.
- **Keamanan berkas**: kebijakan pelarangan akses jalur sensitif dan validasi jalur mencegah *path traversal* dan operasi berisiko.
- **Modifikasi terarah**: perintah `MODIFY` memanfaatkan *diff* untuk membatasi ruang perubahan, mendukung prinsip perubahan minimal.
- **Keterulangan eksperimen**: penggunaan Poetry dan Makefile memudahkan replikasi lingkungan dan dokumentasi langkah.

BAB 3

Metodologi Penelitian

3.1 Metode Pengembangan

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

3.2 Arsitektur Sistem

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

- **Antarmuka CLI (`cli.py`)**: titik masuk perintah `pai` dan pengelola argumen (subperintah `auto`, `config`). Secara default, CLI memanggil sesi interaktif agen.
- **Agen (`agent.py`)**: menyusun prompt, mengelola memori percakapan, dan mengeksekusi rencana aksi hasil LLM. Menyediakan perintah: `MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`.
- **Jembatan LLM (`llm.py`)**: menangani konfigurasi API Gemini dan penyederhanaan hasil keluaran.
- **Gerbang Sistem Berkas (`fs.py`)**: menyediakan operasi berkas dengan kebijakan keamanan (validasi jalur, blokir direktori sensitif), serta mekanisme *diff*-aware untuk `MODIFY`.
- **Tampilan Terminal (`ui.py`)**: penyajian hasil eksekusi menggunakan `rich` (panel, warna, penomoran baris).

Alur data tipikal: masukan pengguna (CLI) → konstruksi prompt (Agen) → panggilan LLM → rencana aksi → eksekusi tindakan (FS/UI) → pelaporan dan pencatatan konteks sebagai memori percakapan.

3.3 Alat dan Lingkungan

Lingkungan dan alat yang digunakan:

- Sistem operasi: Ubuntu (Linux).
- Bahasa pemrograman: Python (`>= 3.9`).
- Manajer dependensi/kemasan: Poetry; titik masuk CLI didefinisikan pada `pyproject.toml`.
- LLM: Google Gemini melalui paket `google-generativeai`.
- TUI: `rich` untuk panel, warna, dan penyorotan sintaks.
- LaTeX: TeX Live (`texlive-latex-recommended`, `texlive-latex-extra`, dsb.) dan Makefile untuk kompilasi naskah.
- Kendali versi: Git dan GitHub.

3.4 Prosedur Penelitian

Prosedur penelitian dan evaluasi dirancang sebagai berikut:

1. **Perancangan**: mendefinisikan skenario penggunaan, himpunan perintah agen, dan kebijakan keamanan jalur.
2. **Implementasi**: membangun modul-modul inti (CLI, Agen, LLM, FS, 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 jalur sensitif), dan (iv) pengalaman pengguna (keterbacaan output).

6. **Analisis**: mengidentifikasi kelebihan, kekurangan, dan peluang peningkatan (mis. dukungan multi-LLM, integrasi editor, perluasan kebijakan keamanan).

BAB 4

Implementasi dan Hasil

4.1 Implementasi Paicode

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

4.1.1 Instalasi

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

Listing 4.1: Instalasi dependensi dengan Poetry

```
poetry install
```

4.1.2 Konfigurasi API Key

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

Listing 4.2: Set dan verifikasi API key Gemini

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

4.1.3 Menjalankan Agen

Sesi interaktif dapat dimulai langsung:

Listing 4.3: Menjalankan sesi agen interaktif

```
poetry run pai
```

4.2 Alur Interaksi

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

4.3 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 sumber kode
# Agen mengeksekusi: READ
```

4.4 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 jalur (kegagalan akses jalur sensitif).

Hasil awal menunjukkan bahwa pendekatan agen *stateful* dengan batasan perubahan berbasis *diff* memudahkan penulisan dan pengembangan bertahap sambil menekan risiko penimpaan berkas yang tidak diinginkan. Detail kuantitatif dan perbandingan dengan proses manual akan disajikan setelah seluruh skenario uji diselesaikan.

BAB 5

Kesimpulan dan Saran

5.1 Kesimpulan

Penelitian ini menghasilkan prototipe **Paicode**, sebuah agen AI berbasis CLI yang mendukung proses pengembangan perangkat lunak secara interaktif dengan memanfaatkan LLM eksternal. Sistem dirancang dengan prinsip *local-first* dan dilengkapi kebijakan keamanan jalur untuk mencegah akses ke direktori sensitif. Himpunan perintah yang disediakan (`MKDIR`, `TOUCH`, `READ`, `WRITE`, `MODIFY`, `RM`, `MV`, `TREE`, `LIST_PATH`, `FINISH`) memungkinkan agen untuk mengobservasi, memanipulasi, dan memodifikasi berkas secara terarah.

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

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

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

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

4. Pemakaian Poetry, Makefile, dan LaTeX mendukung keterulangan eksperimen serta dokumentasi terstruktur untuk keperluan akademik.

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

5.2 Saran

Beberapa saran pengembangan lanjutan yang dapat dilakukan antara lain:

- **Dukungan multi-LLM**: menambahkan opsi pemilihan model dan penyedia LLM alternatif sesuai kebutuhan (akurasi/biaya/latensi).
- **Integrasi editor**: menyediakan jembatan ringan ke IDE (mis. VS Code) tanpa mengorbankan sifat *local-first*, misalnya melalui ekstensi yang memanggil agen CLI.
- **Peningkatan keamanan**: memperluas kebijakan *allow/deny list* jalur, menambah konfirmasi eksplisit untuk operasi berisiko, dan memperketat validasi konten sebelum penulisan berkas.
- **Memori jangka panjang**: menambahkan ringkasan sesi dan penyimpanan konteks terkurasi agar agen dapat mempelajari preferensi proyek pengguna secara berkelanjutan.
- **Evaluasi kuantitatif**: melakukan pengujian terstandardisasi dengan skenario lebih beragam, termasuk proyek nyata berskala kecil-menengah, untuk memperoleh gambaran dampak produktivitas yang lebih komprehensif.

Bibliografi

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] Ashish Vaswani et al. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.