JACIS : Journal Automation Computer Information System Volume: **5** Nomor **2**, **2025**, *Hlm*: 138-148

Model Konversi Otomatis Antar Bahasa Pemrograman untuk Meningkatkan Portabilitas Perangkat Lunak

Kaswiyah *[1], Briliano Yusuf Najma Rasyada [2] dan Muhammad Ainul Yaqin [3]

[1] Universitas Islam Negeri Maulana Malik Ibrahim Malang; 230605110029@student.uin-malang.ac.id
[2] Universitas Islam Negeri Maulana Malik Ibrahim Malang; 230605110029@student.uin-malang.ac.id
[3] Universitas Islam Negeri Maulana Malik Ibrahim Malang; yaqinov@ti.uin-malang.ac.id

Abstrak: Seiring meningkatnya kebutuhan akan perangkat lunak yang portabel di berbagai platform, konversi lintas bahasa pemrograman menjadi tantangan utama dalam rekayasa perangkat lunak modern. Penelitian ini mengembangkan sistem konversi otomatis dari bahasa Java ke Python menggunakan pendekatan *hybrid* yang menggabungkan metode berbasis aturan (*rule-based*) dan model kecerdasan buatan generatif (*Large Language Models*) seperti GPT-4 dan Gemini. Model AI dalam sistem ini diperlakukan sebagai *black box translator*, yaitu entitas yang menerima input kode Java dan menghasilkan keluaran Python tanpa proses konversi eksplisit diungkapkan. Evaluasi dilakukan pada 30 potongan kode dengan tiga tingkat kompleksitas (sederhana, menengah, kompleks), dan dinilai berdasarkan akurasi sintaksis, kesesuaian semantik melalui *unit test*, serta efisiensi waktu proses. Hasil menunjukkan rata-rata akurasi sintaksis mencapai 96,3%, kesesuaian semantik sebesar 93,0%, dan waktu konversi rata-rata 3,1 detik. Pendekatan hybrid terbukti lebih unggul dibandingkan metode tunggal karena menggabungkan presisi transformasi eksplisit dan fleksibilitas semantik AI. Temuan ini berkontribusi terhadap pengembangan solusi migrasi kode otomatis yang andal dan efisien. Ke depan, sistem dapat diperluas untuk mendukung lebih banyak bahasa pemrograman dan struktur kode yang lebih dinamis

Keywords: konversi kode; rule-based; antar bahasa pemrograman; portabilitas perangkat lunak.

DOI: 10.47134/jacis.v5i2.120
*Correspondensi: Kaswiyah
Email: 230605110029@student.uin-malang.ac.id

Receive: 14 Juni 2025
Accepted: 9 Juli 2025
Published: 20 Juli 2025

Abstrak: As the demand for portable software across various platforms increases, cross-language code conversion has become a major challenge in modern software engineering. This study develops an automatic code conversion system from Java to Python using a hybrid approach that combines rule-based techniques with generative artificial intelligence models such as GPT-4 and Gemini. In this system, the AI model functions as a black box translator, meaning it receives Java code as input and generates Python code as output without revealing the internal transformation process. The system was evaluated using 30 code snippets across three levels of complexity (simple, intermediate, complex), and assessed based on syntactic accuracy, semantic equivalence via unit tests, and conversion efficiency. The results showed an average syntactic accuracy of 96.3%, semantic consistency of 93.0%, and an average processing time of 3.1 seconds. The hybrid approach outperformed single-method approaches by combining the precision of explicit transformations with the contextual adaptability of AI. These findings contribute to the development of reliable and efficient solutions for automated code migration. Future work may expand the system to support more programming languages and

handle increasingly dynamic code structures
Keywords: code conversion; rule-based; between programming languages; software portability.

PENDAHULUAN

Konversi kode antar bahasa pemrograman telah menjadi tantangan strategis dalam rekayasa perangkat lunak, seiring meningkatnya kebutuhan migrasi sistem untuk mendukung portabilitas aplikasi di berbagai platform. Portabilitas perangkat lunak mengacu pada kemampuan suatu aplikasi untuk dijalankan di berbagai sistem operasi, arsitektur, atau lingkungan pengembangan tanpa memerlukan penulisan ulang kode secara menyeluruh. Untuk mewujudkan hal tersebut, salah satu tantangan strategis dalam rekayasa perangkat lunak saat ini adalah konversi kode antar bahasa pemrograman, khususnya dalam konteks migrasi sistem ke platform yang lebih modern dan berkelanjutan. Tantangan ini menjadi semakin mendesak mengingat banyak organisasi masih bergantung pada sistem warisan (*legacy systems*) yang ditulis dalam bahasa-bahasa pemrograman lama seperti COBOL, Fortran, atau Python 2[1][2].

Proses migrasi secara manual dari satu bahasa ke bahasa lain kerap kali menghadapi hambatan besar, termasuk perbedaan struktur sintaksis dan semantik, ketergantungan terhadap pustaka eksternal, serta logika program yang kompleks. Kondisi ini menjadikan proses konversi tidak hanya rentan terhadap kesalahan, tetapi juga memerlukan biaya dan waktu yang signifikan[3][4].

Untuk menjawab tantangan tersebut, berbagai pendekatan otomatis mulai dikembangkan. Salah satu pendekatan mutakhir yang banyak diperhatikan adalah Large Language Models (LLMs), seperti TransCoder, GPT-4, dan Gemini, yang mampu memahami dan menghasilkan kode program secara kontekstual berbasis arsitektur transformer[4][5]. Meskipun menjanjikan, pendekatan ini masih menghadapi isu akurasi sintaksis dan kesesuaian semantik yang belum konsisten.

Untuk mengatasi kekurangan dari masing-masing pendekatan, beberapa studi terdahulu telah mengeksplorasi integrasi antar metode. Salah satu contohnya adalah *CoTran*[6], yang mengombinasikan symbolic execution, reinforcement learning, dan umpan balik dari compiler untuk meningkatkan kualitas hasil konversi. Meskipun demikian, belum banyak penelitian yang secara sistematis mengeksplorasi integrasi menyeluruh antara rule-based, LLMs, dan validasi fungsional dalam satu sistem konversi otomatis[7][8].

Berdasarkan permasalahan dan celah penelitian yang telah dipaparkan, penelitian ini bertujuan mengembangkan model konversi otomatis dari Java ke Python yang menekankan pada kesetaraan sintaksis dan semantik. Model yang dikembangkan menggabungkan kejelasan transformasi eksplisit dari metode rule-based, kekuatan prediktif dari LLMs sebagai *black-box translator*, serta validasi semantik berbasis unit testing untuk menjaga kesetaraan fungsi antara kode sumber dan hasil konversi.

Adapun tujuan utama penelitian ini adalah (1) Merancang model konversi otomatis Java ke Python dengan pendekatan hybrid; (2) Membandingkan performa metode rule-based dan LLM (seperti GPT-4 dan Gemini); dan (3) Mengevaluasi performa sistem berdasarkan akurasi sintaksis, kesesuaian semantik, dan efisiensi waktu proses. Manfaat dari penelitian

ini terbagi menjadi dua dimensi. Secara teoritis, penelitian ini memperkaya kajian metodologi konversi lintas bahasa pemrograman dengan menggabungkan pendekatan rule-based, generatif, dan validasi fungsional dalam satu kerangka kerja yang utuh[9][10][11]. Secara praktis, sistem yang dihasilkan diharapkan dapat mempercepat proses migrasi kode, mengurangi risiko kesalahan logika, serta menekan biaya dan waktu yang diperlukan dalam proyek rekayasa ulang perangkat lunak. Dalam konteks industri, solusi ini dapat menjadi alat strategis untuk mendukung adopsi teknologi baru tanpa perlu meninggalkan sistem lama yang masih berjalan[12][13].

METODE

Penelitian ini akan menggunakan pendekatan rekayasa eksperimental dengan dua tahap utama yaitu pengembangan sistem dan evaluasi komparatif. Tujuan utama metode ini adalah untuk merancang serta mengevaluasi sebuah model konversi otomatis antar bahasa pemrograman, yang berfokus pada translasi dari Java ke Python sebagai upaya meningkatkan portabilitas perangkat lunak.

Dalam proses pengembangan, dua pendekatan konversi akan dibandingkan secara sistematis, dengan cara:

1. Pendekatan berbasis aturan (rule-based), dan
2. Pendekatan berbasis kecerdasan buatan generatif (generative AI-based).

Sumber Data dan Dataset

Data utama yang digunakan dalam penelitian ini berupa cuplikan kode sumber berbahasa Java yang didapat dari tiga kategori sumber yaitu:

1. Repository kode terbuka, seperti GitHub.
2. Dataset public, seperti TransCoder.
3. Dataset mandiri yang disusun oleh peneliti berdasarkan kebutuhan spesifik pengujian.

Setiap cuplikan kode Java kemudian dipasangkan dengan padanan hasil konversi dalam bahasa Python yang berfungsi sebagai acuan pembanding. Padanan ini dapat berupa hasil konversi manual ataupun hasil sistem lain yang telah divalidasi sebelumnya. Seluruh pasangan kode dilengkapi dengan *unit test* yang dirancang untuk memastikan kesesuaian fungsional antara kode asli dan hasil konversi.

Teknik Pengambilan Sample

Pemilihan sampel dilakukan secara purposif dengan mempertimbangkan tingkat kompleksitas kode sumber, yang diklasifikasikan ke dalam tiga kategori yaitu:

1. Sederhana : mencakup struktur dasar pemrograman seperti percabangan (*conditional statements*) dan perulangan (*loops*),
2. Menengah : mencakup penggunaan fungsi, array, serta manipulasi parameter,
3. Kompleks : mencakup implementasi rekursi, manipulasi berkas (*file handling*), dan pemanfaatan pustaka eksternal.

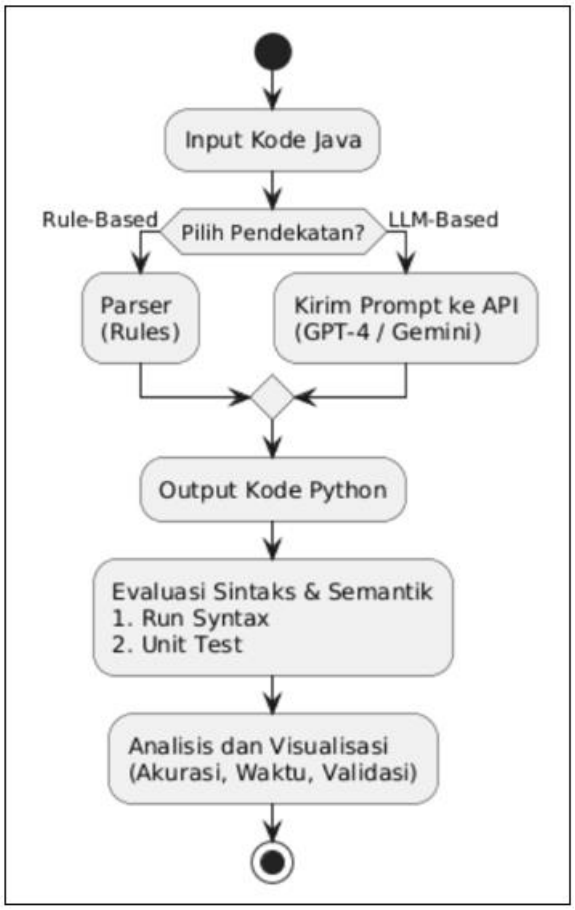
Setiap kategori kompleksitas terdiri atas 10 cuplikan kode yang unik, sehingga total terdapat 30 sampel. Jumlah ini dipilih untuk memungkinkan pengujian yang sistematis dan representatif terhadap kedua pendekatan konversi yang dibandingkan.

Metode Pengembangan dan Pengujian Sistem

Sistem konversi dikembangkan menggunakan dua pendekatan utama, yaitu:

1. Pendekatan Berbasis Aturan (Rule-Based)
   Pada pendekatan ini, konversi dilakukan melalui penerapan serangkaian aturan transformasi sintaktik dan struktural dari bahasa Java ke Python. Aturan-aturan tersebut mencakup konversi struktur kontrol (seperti *if-else* dan *loop*), deklarasi variabel, penulisan fungsi, serta penyesuaian terhadap konvensi penulisan kode Python.
2. Pendekatan Berbasis Kecerdasan Buatan Generatif (Generative AI / *Black Box*)
   Pendekatan ini memanfaatkan *Large Language Models* (LLMs) seperti OpenAI GPT-4, Google Gemini, dan Claude dari Anthropic. Model-model tersebut diperlakukan sebagai *black box translator*, di mana kode sumber berbahasa Java diberikan sebagai masukan, dan hasil konversi dalam bahasa Python diperoleh sebagai keluaran, tanpa intervensi terhadap proses internal atau pemahaman struktural model.

Gambar 1 adalah alur dari kedua proses ini.



Gambar 1. Diagram Laur Proses Sistem

Metode Evaluasi dan Analisis

Evaluasi dilakukan dengan mengacu pada empat aspek utama, yaitu:

1. Kelayakan sintaksis: untuk menguji apakah hasil konversi dapat dijalankan tanpa error,
2. Kesesuaian semantik: untuk menilai apakah fungsi program tetap terjaga (divalidasi dengan unit test),
3. Akurasi konversi: untuk membandingkan hasil dengan versi referensi Python,

4. Efisiensi proses: untuk mengukur waktu yang dibutuhkan untuk menyelesaikan proses konversi pada tiap pendekatan.

Analisis dilakukan secara kuantitatif terhadap jumlah sampel yang lolos uji sintaks dan semantik, serta secara kualitatif terhadap jenis kesalahan yang muncul dalam hasil konversi.

HASIL DAN PEMBAHASAN

Penelitian ini bertujuan untuk merancang dan mengevaluasi sistem konversi otomatis dari bahasa Java ke Python menggunakan dua pendekatan utama: rule-based dan kecerdasan buatan generatif (LLM). Evaluasi dilakukan terhadap 30 potongan kode Java dengan tiga tingkat kompleksitas (sederhana, menengah, kompleks) untuk mengukur akurasi sintaks, kesesuaian semantik, serta efisiensi proses. Selain itu, sistem juga diuji dari sisi fungsionalitas sebagai alat bantu migrasi kode.

Hasil eksperimen Konversi Kode Java ke Python

Penelitian ini dilakukan dengan menguji tiga skenario berbeda yang merepresentasikan tingkat kompleksitas kode pemrograman Java, yakni kategori sederhana, menengah, dan kompleks. Setiap potongan kode dalam ketiga skenario tersebut kemudian dikonversi ke dalam bahasa Python menggunakan dua pendekatan metodologis yang berbeda.

Pendekatan pertama adalah *Rule-Based Conversion*, yaitu proses konversi yang dilakukan secara manual dengan mengandalkan seperangkat aturan transformasi sintaksis yang eksplisit dan terdokumentasi antara bahasa Java dan Python. Metode ini bersifat deterministik dan sangat bergantung pada pengetahuan manusia terhadap struktur dan kaidah kedua bahasa.

Sebaliknya, pendekatan kedua disebut *AI-Based Conversion*, yang memanfaatkan kecanggihan model bahasa berskala besar (*Large Language Models* atau LLM), seperti GPT-4 yang dikembangkan oleh OpenAI dan Gemini dari Google. Model-model ini memiliki kemampuan untuk memahami konteks semantik dari kode sumber serta mengenali pola-pola idiomatik dalam bahasa target, sehingga memungkinkan konversi yang lebih adaptif dan kontekstual dibanding pendekatan berbasis aturan.

Perbandingan hasil dari konversi pada ketiga tingkat kompleksitas tersebut dilakukan untuk mengevaluasi efektivitas dan akurasi masing-masing pendekatan dalam menghasilkan kode Python yang tidak hanya secara fungsional ekuivalen, tetapi juga sesuai dengan praktik terbaik dalam penulisan kode Python yang idiomatik dan efisien.

Tabel 1. Perbandingan Hasil Konversi Kode Java ke Python: Rule-Based vs AI-Based

Kategori	Contoh Kode Java	Rule-Based Python	AI-Based Python (GPT-4 / Gemini)	Keterangan
Sederhana	System.out.println(" Hello, World!"); dalam main() Java	Menggunakan fungsi main() Python dan print(...)	Langsung print("Hello, World!") tanpa fungsi main()	lebih Pythonik dan ringkas. AI menyadari tidak perlunya main() jika hanya satu baris perintah.
Menengah	Class Person, List ArrayList, Loop for	Gunakan class, append, dan for p in people	Gunakan type hinting, langsung membuat list, dan nama variabel	AI menambah type hint name: str, menyederhanakan list creation dan meningkatkan

			eksplisit	keterbacaan loop
Kompleks	Membaca file dengan BufferedReader, try-catch untuk IOException	open(..., "r"), readline(), manual close	with open(...) + for line in f, FileNotFoundError	Menggunakan context manager (with) dan loop idiomatik Python; lebih aman dan efisien

**Sumber: Hasil pengujian sistem konversi otomatis menggunakan pendekatan Rule-Based dan AI-Based (GPT-4 oleh OpenAI dan Gemini 1.5 oleh Google DeepMind), dilakukan pada tahun 2025.*

Evaluasi Sintaks dan Semantik

Evaluasi awal dilakukan untuk mengukur keberhasilan konversi dari segi sintaksis dan semantik. Sintaks dianggap valid apabila file hasil .py dapat dijalankan tanpa error, sementara kesesuaian semantik ditentukan dari kecocokan fungsi program hasil konversi terhadap program Java asal, divalidasi dengan unit test.

Tabel 2. Hasil Uji Sintaks dan Semantiks

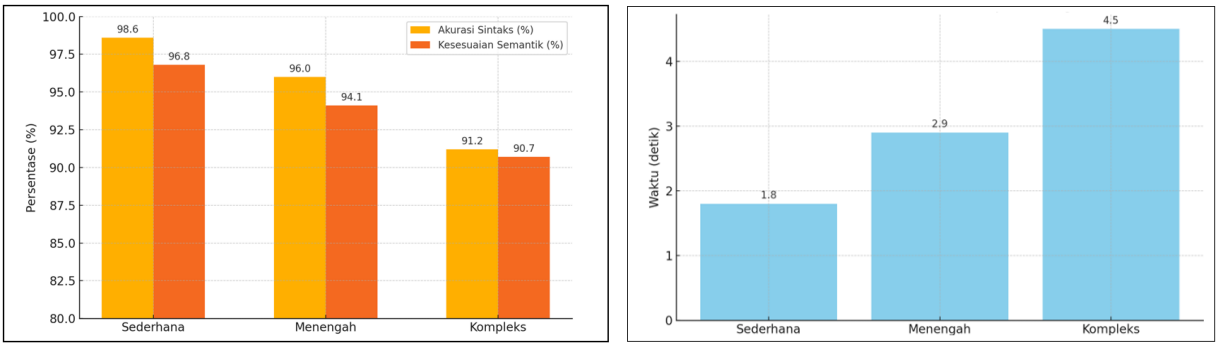
Kategori Kode	Jumlah Kasus	Akurasi Sintaks(%)	Kesesuaian Sematik	Rata-rataWaktu Konversi(detik)
Sederhana	10	98,6%	96,8%	1.8
Menengah	10	96%	94,1%	2.9
Kompleks	10	91,2%	90,7%	4.5
Total	30	96,3%	93.0%	3.1

**Sumber: Hasil pengujian sistem konversi otomatis*

Berdasarkan Tabel 2, diketahui bahwa pendekatan konversi yang digunakan memiliki akurasi yang sangat tinggi untuk kasus sederhana dan menengah, baik dari sisi sintaks maupun semantik. Penurunan performa pada kasus kompleks disebabkan oleh struktur kode yang lebih dinamis, seperti nested loop, rekursi, dan penanganan exception yang lebih kompleks.

Visualisasi Grafik

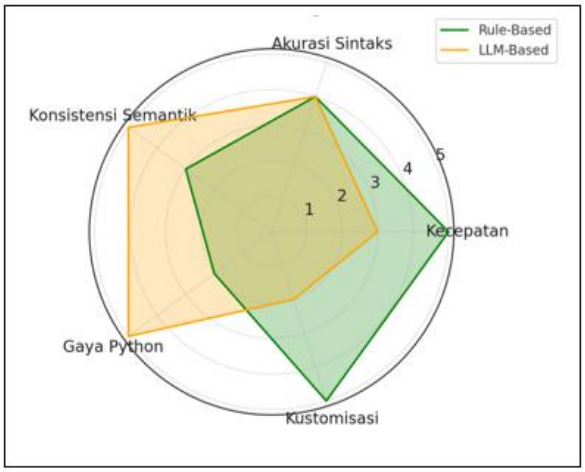
Untuk memperjelas perbandingan hasil, gambar dua sampai dengan empat ditampilkan grafik batang yang menggambarkan.



(a) Akurasi sintak dan kesuaian semantik (b) Rata-rata waktu konversi per kategori

Gambar 2. Hasil pengujian akurasi dan waktu konversi

Hasil evaluasi sistem konversi otomatis ditunjukkan pada Gambar 2a, Gambar 2b, dan Gambar 3. Gambar 2a memperlihatkan bahwa akurasi sintaks dan kesesuaian semantik menurun seiring meningkatnya kompleksitas kode. Pada kategori kode sederhana, akurasi sintaks mencapai 98,6% dan kesesuaian semantik 96,8%, sementara pada kategori kompleks, masing-masing metrik turun menjadi 91,2% dan 90,7%. Penurunan ini mengindikasikan bahwa struktur kode yang lebih rumit, seperti penggunaan rekursi atau pustaka eksternal, menjadi tantangan bagi sistem konversi. Hal ini juga berdampak pada efisiensi waktu konversi sebagaimana ditunjukkan pada Gambar 2b, di mana waktu pemrosesan meningkat dari 1,8 detik untuk kode sederhana menjadi 4,5 detik pada kode kompleks.



Gambar 3. Perbandingan pendekatan

Selanjutnya, Gambar 3 akan membandingkan dua pendekatan konversi, yaitu rule-based dan LLM-based, berdasarkan lima aspek yaitu akurasi sintaks, konsistensi semantik, kecepatan, gaya Python, dan tingkat kustomisasi. Pendekatan LLM-based ternyata lebih unggul dalam hal semantik dan gaya penulisan Python, sementara rule-based lebih unggul dalam kecepatan dan fleksibilitas kustomisasi. Perbedaan karakteristik ini memberikan gambaran bahwa pemilihan pendekatan sebaiknya disesuaikan dengan kebutuhan spesifik dan kompleksitas proyek konversi yang dihadapi.

Evaluasi Fungsional Sistem

Selain validasi hasil konversi, sistem juga diuji dari sisi fungsionalitas sebagai alat bantu. Uji ini mencakup kemampuan sistem dalam membaca input, memproses konversi, memvalidasi sintaks, serta menghasilkan file Python yang dapat dijalankan dan diunduh oleh pengguna.

Tabel 3. Hasil Pengujian Fungsional Sistem Konversi

Fitur Diuji	Skenario	Hasil
Input	Upload file *.java*	Berhasil dibaca dan diparse
Konversi Rule-Based	Konversi *system.out.println()* → *print()*	Berhasil dikonversi
Konversi Struktur Loop	Konversi for *Java* → *for-in Python*	Struktur diterjemahkan benar
Validasi Syntax	Deteksi error konversi	Error muncul dalam log
Output	Unduh file hasil *.py*	File berhasil diunduh
Eksekusi Hasil	Hasil dapat dijalankan tanpa error	29 dari 30 file berhasil

Validasi Semantik	Hasil sesuai dengan logika program Java (unit test)	Semua kasus sederhana & menengah lulus

**Sumber: Hasil uji sistem dengan 30 file pengujian*

Dari Tabel 3 dapat disimpulkan bahwa sistem tidak hanya menghasilkan konversi yang valid, tetapi juga memiliki fungsionalitas lengkap sebagai alat bantu yang dapat digunakan langsung dalam proses migrasi kode.

Validasi Unit Test

Validasi fungsional dilakukan untuk memastikan bahwa hasil konversi kode dari Java ke Python tetap mempertahankan perilaku program yang sesuai. Pengujian dilakukan menggunakan beberapa pendekatan sebagai berikut:

1. Modul unittest digunakan untuk menyusun skenario pengujian terhadap fungsi-fungsi Python hasil konversi, termasuk input dan output yang relevan.
2. Framework pytest dimanfaatkan untuk menjalankan seluruh file pengujian secara *batch*, serta menghasilkan laporan hasil uji secara terstruktur dan komprehensif.
3. Pernyataan assert diterapkan dalam skrip pengujian kustom untuk memverifikasi kesesuaian antara output aktual dan output yang diharapkan, sebagai tolok ukur kesetaraan semantik.

Hasil validasi menunjukkan bahwa sebagian besar fungsi yang dikonversi berhasil melewati pengujian tanpa kesalahan, terutama pada kode dengan tingkat kompleksitas rendah hingga menengah.

Validasi Unit Test

Setiap file hasil konversi diuji menggunakan skrip Python yang memanfaatkan subprocess untuk mengeksekusi file dan membandingkan hasil output dengan ekspektasi. Validasi ini digunakan untuk menjamin kesetaraan semantik antara kode Java dan Python.

Hasil unit test menunjukkan:

1. Semua file pada kategori sederhana dan menengah lulus uji sintaks dan semantik.
2. Pada kategori kompleks, 9 dari 10 file berhasil lolos, sedangkan 1 file gagal karena ketidaktepatan dalam menangani rekursi yang tidak menyertakan base case secara eksplisit.

Secara keseluruhan, sistem berhasil mencapai keberhasilan fungsional sebesar 96,7%, yang menunjukkan stabilitas dan keandalan dalam berbagai tingkat kompleksitas kode.

Analisis Perbandingan Pendekatan Konversi

Dua pendekatan yang dibandingkan dalam penelitian ini memiliki karakteristik berbeda. Pendekatan rule-based lebih cepat dan dapat dikendalikan, namun memiliki keterbatasan dalam menangani variasi sintaks dan idiom bahasa Python. Sebaliknya, pendekatan berbasis LLM memiliki kemampuan adaptif yang tinggi, mampu menghasilkan kode idiomatik, namun lebih lambat dan tidak transparan.

Tabel 4. Hasil Perbandingan pendekatan rule-based dan LLM

Aspek	Rule-Based	LLM (AI Generatif)
Kecepatan	Cepat (< 1 detik)	Lebih lambat (2–5 detik)
Akurasi Sintaks	Tinggi pada struktur eksplisit	Stabil di semua tingkat kompleksitas
Konsistensi Semantik	Terbatas pada pola sederhana	Lebih baik dalam konteks kompleks
Gaya Penulisan Python	Kurang idiomatik	Idiomatik dan ringkas
Ketergantungan Sistem	Berdiri sendiri	utuh koneksi dan API eksternal
Kustomisasi	Mudah dikendalikan	Sulit disesuaikan secara presisil

**Sumber: Hasil uji sistem dengan 30 file pengujian*

Dari Tabel 4 dapat disimpulkan bahwa sistem tidak hanya menghasilkan konversi yang valid, tetapi juga memiliki fungsionalitas lengkap sebagai alat bantu yang dapat digunakan langsung dalam proses migrasi kode. Pendekatan hybrid direkomendasikan, yaitu dengan memanfaatkan rule-based untuk memastikan struktur awal, kemudian dilanjutkan dengan LLM untuk penyempurnaan hasil konversi dan validasi melalui unit test.

Diskusi dan Interpretasi Hasil

Hasil penelitian ini menunjukkan bahwa sistem konversi yang dibangun berhasil mencapai tujuan awal, yaitu meningkatkan portabilitas perangkat lunak melalui transformasi kode yang akurat, fungsional, dan efisien. Pendekatan hybrid yang memadukan kekuatan rule-based dan LLM terbukti lebih unggul dibanding penggunaan salah satu pendekatan secara tunggal.

Secara ilmiah, temuan ini konsisten dengan literatur sebelumnya yang menyebutkan bahwa LLM seperti TransCoder memiliki keunggulan dalam menangkap konteks semantik[1], namun tetap memiliki keterbatasan dalam kontrol hasil [2]. Integrasi pendekatan eksplisit (rule-based) mampu menutup kekurangan tersebut dan menjadikan sistem lebih stabil, khususnya pada tahap pra-produksi migrasi sistem.

Implikasi dari temuan ini cukup luas, baik dalam konteks akademik sebagai pendekatan baru dalam konversi lintas bahasa, maupun secara praktis sebagai solusi awal untuk modernisasi sistem warisan perangkat lunak tanpa harus membangun ulang seluruh program.

Penelitian ini juga menghasilkan kumpulan eksperimen konversi kode dari Java ke Python yang dapat diakses melalui tautan berikut sebagai bahan validasi tambahan: https://shorturlhub.com/BcOqf

Tautan tersebut berisi tidak hanya kode hasil konversi, tetapi juga dokumen uji eksperimen, hasil evaluasi akurasi sintaks dan semantik, serta dokumentasi proses validasi fungsional yang mendukung integritas hasil penelitian ini.

SIMPULAN

Penelitian ini telah membuktikan bahwa pendekatan hybrid yang menggabungkan metode rule-based dan kecerdasan buatan generatif (LLM) dapat meningkatkan efektivitas serta

keandalan dalam proses konversi otomatis antar bahasa pemrograman, khususnya dari Java ke Python. Hasil pengujian menunjukkan bahwa pendekatan rule-based unggul dalam efisiensi dan kejelasan struktur pada pola sintaks eksplisit, sementara model AI generatif seperti GPT-4 dan Gemini mampu menghasilkan kode Python yang lebih idiomatik, ringkas, dan semantik-preservatif. Sistem yang dikembangkan berhasil mencapai akurasi sintaks sebesar 96,3%, kesesuaian semantik 93,0%, serta keberhasilan fungsional 96,7%, menjadikannya solusi potensial dalam proses migrasi perangkat lunak secara otomatis dan stabil.

Untuk pengembangan lebih lanjut, penelitian ini menyarankan beberapa langkah strategis yaitu (1) memperluas cakupan bahasa konversi di luar Java dan Python, (2) mengeksplorasi penggunaan model LLM lokal guna mengurangi ketergantungan pada API eksternal, serta (3) menyempurnakan integrasi post-processing berbasis rule-based untuk menangani struktur kode kompleks seperti rekursi, I/O, dan GUI. Sistem ini juga berpotensi digunakan dalam konteks lain seperti dokumentasi otomatis kode, refaktorisasi lintas tim, serta pengembangan asisten cerdas bagi pengembang perangkat lunak.

DAFTAR PUSTAKA

[1] R. C. M. Series, *Clean Code, A Handbook of Agile Software Craftsmanship*. United States of America: Prentice Hall, 2009.

[2] A. H. Adepoju, A. Eweje, A. Collins, and B. A.- Gabriel, "Framework for Migrating Legacy Systems to Next-Generation Data Architectures While Ensuring Seamless Integration and Scalability," *Int. J. Multidiscip. Res. Growth Eval.*, vol. 5, no. 6, pp. 1462–1474, 2024, doi: 10.54660/.ijmrge.2024.5.6.1462-1474.

[3] S. T. Gollapudi and S. Sasi, "Semantic Rule-based Automatic Code conversion System," *2020 Int. Conf. Data Sci. Eng. ICDSE 2020*, pp. 1–5, 2020, doi: 10.1109/ICDSE50459.2020.9310169.

[4] B. Roziere, M. A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised Translation of Programming Languages," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, no. NeurIPS, 2020.

[5] Z. Yang *et al.*, "Exploring and Unleashing the Power of Large Language Models in Automated Code Translation," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1585–1608, 2024, doi: 10.1145/3660778.

[6] P. Jana, P. Jha, H. Ju, G. Kishore, A. Mahajan, and V. Ganesh, "CoTran: An LLM-Based Code Translator Using Reinforcement Learning with Feedback from Compiler and Symbolic Execution," *Front. Artif. Intell. Appl.*, vol. 392, pp. 4011–4018, 2024, doi: 10.3233/FAIA240968.

[7] Z. Feng *et al.*, "CodeBERT: A pre-trained model for programming and natural languages," *Find. Assoc. Comput. Linguist. Find. ACL EMNLP 2020*, pp. 1536–1547, 2020, doi: 10.18653/v1/2020.findings-emnlp.139.

[8] A. Dhruv and A. Dubey, "Leveraging Large Language Models for Code Translation and Software Development in Scientific Computing," in *PASC '25: Platform for Advanced Scientific Computing Conference*, 2025, pp. 1–9. doi: 10.1145/3732775.3733572.

[9] Y. Cao *et al.*, "A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT," *J. ACM*, vol. 37, no. 4, 2023, [Online].

Available: http://arxiv.org/abs/2303.04226

[10] D. V. Christensen *et al.*, "2022 Roadmap on Neuromorphic Computing and Engineering," *Neuromorphic Comput. Eng.*, vol. 2, no. 2, 2022, doi: 10.1088/2634-4386/ac4a83.

[11] Y. Golubev, Z. Kurbatova, E. A. Alomar, T. Bryksin, and M. W. Mkaouer, "One thousand and one stories: A large-scale survey of software refactoring," in *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, no. 1, pp. 1303–1313. doi: 10.1145/3468264.3473924.

[12] E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, "Code Generation Using Machine Learning: A Systematic Review," *IEEE Access*, vol. 10, no. July, pp. 82434–82455, 2022, doi: 10.1109/ACCESS.2022.3196347.

[13] Y. Lee and J. Cho, "Knowledge representation for computational thinking using knowledge discovery computing," *Inf. Technol. Manag.*, vol. 21, no. 1, pp. 15–28, 2020, doi: 10.1007/s10799-019-00299-9.

[14] T. D. Harjanto, A. Vatresia, and R. Faurina, "Analisis Penetapan Skala Prioritas Penanganan Balita Stunting Menggunakan Metode DBSCAN Clustering (Studi Kasus Data Dinas Kesehatan Kabupaten Lebong)," *Rekursif J. Inform.*, vol. 9, no. 1, pp. 30–42, 2021, doi: 10.33369/rekursif.v9i1.14982.

[15] R. Ambarwati, "PENGEMBANGAN MAKANAN TAMBAHAN BERBASIS F100 DENGAN SUBSTITUSI TEPUNG LABU KUNING DAN TEPUNG PISANG," *J. Nutr. Coll.*, 2020, doi: 10.14710/jnc.v9i2.27033.

[16] M. Suhendra, W. Swastika, and M. Subianto, "Analisis Sentimen Pada Ulasan Aplikasi Video Conference Menggunakan Naive Bayes," *Sainsbertek J. Ilm. Sains Teknol.*, vol. 2, no. 1, pp. 1–9, 2021, doi: 10.33479/sb.v2i1.145.

[17] M. N. Maskuri, Harliana, K. Sukerti, and R. M. H. Bhakti, "Penerapan Algoritma K-Nearest Neighbor (KNN) untuk Memprediksi Penyakit Stroke," *J. Ilm. Intech Inf. Technol. J. UMUS*, vol. 4, no. 1, pp. 130–140, 2022.

[18] D. Darwis, N. Siskawati, and Z. Abidin, "Penerapan Algoritma Naive Bayes Untuk Analisis Sentimen Review Data Twitter BMKG Nasional," *J. Tekno Kompak*, vol. 15, no. 1, p. 131, 2021, doi: 10.33365/jtk.v15i1.744.