Analisis Komparatif Kode GitHub Copilot: Tinjauan Metrik Clean Code dan Efisiensi Algoritma

1st Muhammad Arvind Alaric
Fakultas Informatika
S1 Teknologi Informasi
Bandung, Indonesia
arvind@student.telkomuniversity.ac.id

2nd Aji Gautama Putrada
Fakultas Informatika
S1 Teknologi Informasi
Bandung, Indonesia
ajigps@telkomuniversity.ac.id

Abstrak—Pemanfaatan GitHub Copilot sebagai asisten pemrograman menjanjikan peningkatan produktivitas, namun memunculkan pertanyaan mengenai validitas fungsional dan kualitas struktural kode yang dihasilkan. Penelitian ini bertujuan mengevaluasi kualitas kode GitHub Copilot dibandingkan dengan programmer manusia (mahasiswa tingkat akhir) pada masalah pemrograman berorientasi objek kompleksitas menengah. Metode penelitian menggunakan eksperimen terkontrol dengan pendekatan mixed-method: evaluasi fungsional menggunakan automated test suite (Jest), analisis kualitas statis (Clean Code) menggunakan SonarQube, dan analisis efisiensi algoritma (Big O) secara kualitatif. Hasil penelitian menunjukkan dikotomi kompetensi yang signifikan; AI unggul dalam validitas struktural dengan skor Cognitive Complexity jauh lebih rendah ($p < 0.05$ pada uji Wilcoxon) dan konsistensi tinggi. Sebaliknya, programmer manusia unggul dalam validitas logika dan penanganan kasus tepi (edge cases). Ditemukan fenomena "Clean Garbage" pada AI, di mana kode terlihat rapi secara struktur namun gagal secara fungsional. Kesimpulannya, peran manusia tetap krusial sebagai arsitek solusi dan validator logika dalam ekosistem pengembangan perangkat lunak berbasis AI.

Kata kunci—GitHub Copilot, Kualitas Kode, Clean Code, Kompleksitas Algoritmik, Eksperimen Terkontrol.

I. PENDAHULUAN

Perkembangan teknologi kecerdasan buatan, khususnya Large Language Models (LLM), telah membawa paradigma baru dalam rekayasa perangkat lunak melalui kehadiran asisten pemrograman berbasis AI (*Artificial Intelligence*) seperti GitHub Copilot [1], [2], [3]. Alat ini menawarkan potensi peningkatan produktivitas yang signifikan, dengan beberapa studi menunjukkan percepatan penyelesaian tugas hingga 55,8% bagi pengembang yang menggunakannya [4]. Namun, adopsi teknologi ini memunculkan kekhawatiran mendasar mengenai kualitas kode yang dihasilkan, terutama terkait kebenaran fungsional (*functional correctness*), keamanan, dan keterpeliharaan (*maintainability*) [1], [5]. Meskipun LLM mampu menghasilkan sintaks yang valid, penelitian terdahulu menunjukkan variabilitas kinerja yang tinggi, di mana tingkat kebenaran kode untuk bahasa seperti JavaScript tercatat hanya mencapai 27% pada kasus tertentu [3]. Hal ini mengindikasikan bahwa ketergantungan pada kode yang dihasilkan AI tanpa validasi yang memadai berisiko meningkatkan utang teknis (*technical debt*) dan kerentanan sistem di masa depan [1], [3].

Evaluasi terhadap asisten pemrograman AI sejauh ini mayoritas berfokus pada penyelesaian masalah algoritmik terisolasi menggunakan dataset standar seperti HumanEval atau LeetCode [1], [2], [3]. Pendekatan ini sering kali kurang merepresentasikan kompleksitas pengembangan perangkat lunak dunia nyata yang melibatkan struktur *Object-Oriented Programming* (OOP), logika bisnis, dan interaksi antar komponen [9], [13]. Terdapat kesenjangan penelitian yang nyata dalam studi komparatif yang membandingkan secara langsung kualitas kode AI dengan kode yang ditulis oleh manusia pada tingkat kompleksitas menengah yang relevan dengan aplikasi industri. Kebutuhan untuk memahami apakah kode AI sekadar "bersih" secara visual namun rapuh secara logika (fenomena *clean garbage*) menjadi semakin mendesak untuk dijawab [9].

Penelitian ini bertujuan untuk melakukan analisis komparatif yang komprehensif antara kualitas kode yang dihasilkan oleh GitHub Copilot dan kode yang ditulis oleh programmer manusia (mahasiswa tingkat akhir). Studi ini menerapkan desain eksperimen terkontrol menggunakan enam set masalah pemrograman berorientasi objek. Kualitas kode dievaluasi menggunakan pendekatan metode campuran (mixed-method), yang mencakup validasi fungsional menggunakan *automated test suite* (Jest), pengukuran metrik kualitas statis (*Clean Code*) seperti *Cognitive Complexity* [5] dan Code Smells menggunakan SonarQube, serta analisis kualitatif efisiensi algoritma menggunakan notasi Big O. Melalui perbandingan ini, penelitian diharapkan dapat memberikan bukti empiris mengenai kapabilitas dan batasan GitHub Copilot, serta merumuskan strategi kolaborasi manusia-AI yang optimal dalam pengembangan perangkat lunak.

II. KAJIAN TEORI

Bagian ini memaparkan landasan teori yang menjadi basis evaluasi dalam penelitian, mencakup teknologi *Large Language Model*, parameter kualitas kode, dan pengukuran efisiensi algoritma.

A. Large Language Models dan GitHub Copilot

Large Language Models (LLM) adalah model pembelajaran mesin yang dilatih menggunakan arsitektur Transformer pada korpus data teks yang sangat besar untuk memahami dan menghasilkan bahasa alami maupun kode pemrograman. GitHub Copilot, yang dibangun di atas model OpenAI (seperti GPT-4o), berfungsi sebagai asisten pemrograman berbasis AI (AI Pair Programmer) yang mampu memberikan saran penyelesaian kode secara real-time [1], [4], [10].

Kualitas output yang dihasilkan oleh LLM sangat dipengaruhi oleh strategi *Prompt Engineering* [16]. Penelitian ini mengevaluasi dua pendekatan *prompting*:

1. *Zero-Shot Prompting*, memberikan instruksi tugas secara langsung tanpa memberikan contoh solusi sebelumnya, mensimulasikan penggunaan naif oleh pengguna awam.
2. *Engineered Prompting*, memberikan instruksi yang terstruktur dengan konteks yang kaya, batasan spesifik, dan arahan langkah demi langkah (*Chain-of-Thought*) untuk memandu

model menghasilkan solusi yang lebih presisi [15].

B. Kualitas Kode dan Metrik *Clean Code*

Clean Code didefinisikan sebagai kode yang mudah dipahami, mudah dipelihara, dan bebas dari ambiguitas, dengan aspek keterbacaan (*readability*) sebagai fondasi utamanya [7], [11]. Tantangan dalam menjaga keterbacaan kode menjadi semakin relevan di era Large Language Models saat ini, seiring meningkatnya penggunaan AI dalam proses pengembangan perangkat lunak [8]. Untuk mengukur kualitas *Clean Code* secara objektif, penelitian ini menggunakan pendekatan analisis statis melalui platform SonarQube, yang secara luas dimanfaatkan dalam studi empiris untuk mengevaluasi kualitas kode, termasuk kode yang dihasilkan oleh sistem berbasis AI [2], [9], [14].

Evaluasi kualitas kode dalam penelitian ini didasarkan pada tiga metrik utama, yaitu Cognitive Complexity, Code Smells, dan Maintainability Rating. Cognitive Complexity mengukur tingkat kesulitan alur kendali kode untuk dipahami oleh manusia dengan memberikan bobot pada struktur yang mengganggu alur pembacaan, seperti nested loops dan conditional yang kompleks, dan telah divalidasi secara empiris sebagai indikator yang akurat dalam mengukur source code understandability [5]. Code Smells merepresentasikan potensi masalah pemeliharaan dalam kode yang tidak secara langsung menimbulkan kesalahan, tetapi dapat memicu bug di masa depan atau menyulitkan pengembangan lanjutan [14]. Sementara itu, Maintainability Rating merupakan peringkat dari A hingga E yang ditentukan berdasarkan varian Maintainability Index serta kerapatan technical debt, dan menjadi metrik penting dalam mengevaluasi keberlanjutan sistem berorientasi objek dalam jangka panjang [12].

C. Kompleksitas Algoritmik (*Big* O *Notation*)

Efisiensi algoritma dalam penelitian ini dianalisis menggunakan notasi *Big* O, yaitu pendekatan matematis yang digunakan untuk menggambarkan batas atas (*worst-case scenario*) dari pertumbuhan waktu eksekusi atau penggunaan memori suatu algoritma relatif terhadap ukuran input (N). Penggunaan metrik kinerja dan tolok ukur (*benchmarks*) seperti ini menjadi aspek penting dalam tinjauan kritis terhadap kode yang dihasilkan oleh kecerdasan buatan, terutama untuk memastikan bahwa solusi yang dihasilkan memiliki tingkat skalabilitas yang memadai ketika diterapkan pada data berukuran besar [6], [13].

Analisis efisiensi algoritma tersebut dibagi ke dalam dua dimensi utama, yaitu *Time Complexity* dan *Space Complexity*. *Time Complexity* mengukur seberapa cepat waktu eksekusi algoritma meningkat seiring bertambahnya ukuran data, dengan hierarki efisiensi yang meliputi O(1) untuk waktu konstan, O(N) untuk waktu linear, O(N log N) untuk log-linear yang umum pada algoritma pengurutan efisien, hingga $O(N^2)$ yang bersifat kuadratik. Sementara itu, *Space Complexity* mengukur jumlah memori tambahan yang dibutuhkan oleh algoritma selama proses eksekusi. Aspek efisiensi memori menjadi sangat krusial, terutama pada lingkungan dengan keterbatasan sumber daya atau ketika algoritma diterapkan pada dataset berskala besar [6].

III. METODE

Penelitian ini menerapkan metode eksperimen terkontrol (*controlled experiment*) dengan desain *within-subject* untuk membandingkan kualitas kode yang dihasilkan oleh *Large Language Model* (GitHub Copilot) dan programmer manusia. Penelitian dilaksanakan pada bulan Oktober 2025 hingga Januari 2026 di lingkungan Fakultas Informatika, Universitas Telkom.

A. Desain Eksperimen dan Variabel

Eksperimen dalam penelitian ini dirancang untuk mengevaluasi dua kelompok perlakuan utama berdasarkan sumber kode yang dihasilkan. Variabel bebas yang digunakan adalah sumber kode program, yang bertujuan untuk membandingkan karakteristik dan kualitas kode yang dihasilkan oleh pengembang manusia dan kecerdasan buatan. Setiap kelompok perlakuan dirancang agar dapat merepresentasikan kondisi pengembangan kode yang realistis dan dapat dibandingkan secara objektif.

Kelompok pertama adalah *Human Developers*, yang terdiri dari tiga partisipan mahasiswa tingkat akhir Fakultas Informatika yang telah lulus mata kuliah Algoritma dan Struktur Data. Partisipan ini diklasifikasikan berdasarkan profil performa menjadi kategori *High*, *Average*, dan *Low* untuk merepresentasikan variasi kemampuan pengembang. Kelompok kedua adalah AI (GitHub Copilot) yang menggunakan model GPT-4o dan diuji melalui dua skenario prompting, yaitu *Standard Zero-Shot Prompting* dengan pemberian deskripsi soal secara langsung tanpa arahan tambahan, serta *Engineered Prompting* yang menggunakan instruksi terstruktur mencakup konteks permasalahan, batasan algoritma, dan format output yang diharapkan.

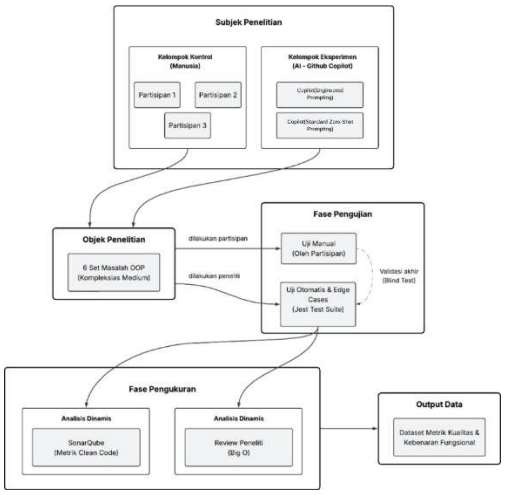
Variabel terikat yang diukur mencakup kebenaran fungsional, metrik *Clean Code*, dan efisiensi algoritmik. Variabel kontrol diterapkan berupa standarisasi lingkungan pengembangan (Visual Studio Code, Node.js), batasan waktu pengerjaan (120 menit per sesi untuk manusia), dan penggunaan set masalah yang identik.

B. Dataset Masalah (*Problem Sets*)

Objek penelitian terdiri dari enam masalah pemrograman berorientasi objek (*Object-Oriented Programming*) dengan tingkat kesulitan menengah. Masalah-masalah ini dirancang untuk menguji berbagai aspek logika pemrograman, meliputi:

1. P-01 (Pengelola Kontak), manipulasi array dan objek dasar.
2. P-02 (Penjadwalan Rapat), algoritma *Greedy* dan optimasi interval.
3. P-03 (Jaringan Sosial), struktur data *Graph* dan algoritma penelusuran (BFS/DFS).
4. P-04 (Analisis Statistik), operasi matematika dan pengurutan data (Sorting).
5. P-05 (Sistem Inventaris), manajemen stok dan validasi logika bisnis.
6. P-06 (Katalog Produk), pencarian dan penyaringan data (*Filtering*).

C. Prosedur Penelitian



GAMBAR 1
DIAGRAM ALUR PENELITIAN

Alur penelitian dilaksanakan melalui serangkaian tahapan sistematis yang divisualisasikan pada Gambar 1. Tahapan dimulai dengan studi literatur dan persiapan lingkungan eksperimen yang terstandarisasi. Pada tahap ini, peneliti mengonfigurasi perangkat lunak Visual Studio Code yang terintegrasi dengan linter statis dan kerangka kerja pengujian otomatis Jest berbasis Node.js. Standarisasi ini bertujuan untuk meminimalisasi variabel pengganggu (*confounding variables*) yang berasal dari perbedaan konfigurasi perangkat keras maupun perangkat lunak, sehingga performa subjek dapat diukur secara objektif.

Setelah lingkungan siap, penelitian berlanjut ke tahap pelaksanaan eksperimen dan pengumpulan data. Pada tahap ini, dua kelompok subjek diberikan perlakuan berbeda terhadap enam set masalah (*problem sets*) yang identik. Kelompok manusia, yang terdiri dari mahasiswa tingkat akhir, diinstruksikan untuk menyelesaikan masalah secara manual dalam batas waktu yang ditentukan tanpa bantuan kecerdasan buatan. Secara paralel, peneliti membangkitkan solusi kode menggunakan GitHub Copilot melalui dua strategi prompting yang berbeda, yaitu *Standard Zero-Shot* dan *Engineered Prompting*, guna mendapatkan variasi data pembanding dari sisi AI.

Tahap akhir meliputi validasi, evaluasi, dan analisis data. Seluruh artefak kode yang dihasilkan oleh kedua kelompok dieksekusi melalui *Automated Test Suite* untuk memverifikasi kebenaran fungsionalnya berdasarkan kriteria *pass*/*fail*. Kode yang valid selanjutnya dianalisis kualitas strukturnya menggunakan SonarQube untuk mendapatkan metrik *Cognitive Complexity* dan *Code Smells*, serta dianalisis efisiensi algoritmanya menggunakan notasi Big O secara manual. Data kuantitatif yang terkumpul kemudian diuji signifikansinya menggunakan uji statistik Wilcoxon Signed-Rank Test untuk menarik kesimpulan mengenai perbandingan kualitas kode antara manusia dan AI.

D. Instrumen Evaluasi

Penelitian ini menggunakan tiga instrumen evaluasi utama untuk mengukur variabel terikat secara komprehensif dan multidimensional, sehingga kualitas solusi kode yang dihasilkan dapat dianalisis tidak hanya dari satu aspek, tetapi dari sudut pandang fungsional, struktural, dan komputasional. Pendekatan ini dirancang untuk meminimalkan bias penilaian serta memberikan gambaran yang lebih utuh mengenai perbedaan karakteristik kode yang dihasilkan oleh pengembang manusia dan sistem kecerdasan buatan.

Instrumen pertama adalah Kebenaran Fungsional, yang diukur menggunakan kerangka kerja pengujian otomatis Jest. Setiap solusi kode diuji melalui serangkaian *test cases* yang telah dirancang untuk merepresentasikan skenario penggunaan normal maupun kasus tepi (*edge cases*) yang sering muncul dalam sistem nyata. Kode dinyatakan valid secara fungsional hanya apabila mampu melewati seluruh pengujian tersebut tanpa menghasilkan runtime error atau kegagalan logika. Pendekatan pengujian otomatis ini memungkinkan evaluasi yang objektif, konsisten, dan dapat direplikasi, sekaligus memastikan bahwa solusi yang dihasilkan benar-benar memenuhi kebutuhan fungsional yang ditetapkan, bukan sekadar menghasilkan sintaks yang valid.

Instrumen kedua adalah Kualitas Kode Statis, yang dievaluasi menggunakan platform analisis kode otomatis SonarQube. Fokus utama pengukuran diarahkan pada metrik *Cognitive Complexity*, yang digunakan untuk menilai tingkat kesulitan pemahaman alur logika kode oleh manusia, serta metrik *Code Smells*, yang mengindikasikan adanya potensi masalah pemeliharaan dalam jangka panjang. Penggunaan analisis statis ini memberikan data kuantitatif yang objektif terkait kebersihan struktur kode dan tingkat kepatuhan terhadap prinsip *Clean Code*. Dengan demikian, instrumen ini memungkinkan perbandingan yang sistematis terhadap kualitas struktural kode yang dihasilkan oleh manusia dan AI, terlepas dari apakah kode tersebut berhasil secara fungsional.

Instrumen ketiga adalah Efisiensi Algoritma, yang dievaluasi secara kualitatif melalui analisis manual menggunakan notasi *Big* O. Analisis ini difokuskan pada *Time Complexity* dalam skenario terburuk (worst-case scenario) dengan melakukan pembongkaran jalur kritis (critical path) pada setiap solusi kode. Pendekatan ini bertujuan untuk menentukan karakteristik pertumbuhan waktu eksekusi algoritma, apakah bersifat linear O(N), logaritmik O(N log N), atau kuadratik O(N^2). Evaluasi efisiensi algoritma ini digunakan sebagai instrumen validasi tambahan untuk memastikan bahwa solusi yang dihasilkan oleh AI tidak hanya ringkas dan rapi secara sintaksis, tetapi juga efisien secara komputasi serta layak diterapkan pada skala data yang lebih besar. Dengan mengombinasikan ketiga instrumen evaluasi tersebut, penelitian ini berupaya memberikan penilaian yang menyeluruh dan seimbang terhadap kualitas solusi pemrograman yang dihasilkan oleh manusia dan kecerdasan buatan.

E. Teknik Analisis Data

Mengingat data penelitian ini berbentuk data berpasangan (*paired samples*) dengan ukuran sampel yang relatif kecil (N = 6) serta tidak memenuhi asumsi normalitas distribusi, maka analisis statistik inferensial dilakukan menggunakan pendekatan non-parametrik, yaitu **Wilcoxon Signed-Rank Test**. Pemilihan uji ini didasarkan pada karakteristik data penelitian yang membandingkan dua kondisi yang saling berhubungan, yakni skor metrik kualitas kode yang dihasilkan oleh kelompok pengembang manusia dan kelompok GitHub Copilot pada tugas pemrograman yang sama. Uji Wilcoxon dinilai sesuai karena tidak mengharuskan data berdistribusi normal serta mampu memberikan estimasi signifikansi yang lebih andal pada ukuran sampel yang terbatas.

Hipotesis nol (H_0) dalam penelitian ini menyatakan bahwa tidak terdapat perbedaan median yang signifikan antara skor metrik kualitas kode pada kelompok pengembang manusia dan kelompok GitHub Copilot. Sebaliknya, hipotesis alternatif (H_1) menyatakan adanya perbedaan median kualitas kode antara kedua kelompok tersebut. Tingkat signifikansi (*alpha*) ditetapkan sebesar 0,05 sebagai ambang batas pengambilan keputusan statistik. Apabila nilai *p-value* yang dihasilkan dari uji Wilcoxon lebih kecil dari 0,05 ($p < 0,05$), maka hipotesis nol ditolak, yang menunjukkan bahwa

perbedaan kualitas kode antara kode buatan manusia dan kode yang dihasilkan oleh AI bersifat signifikan secara statistik. Sebaliknya, jika nilai *p-value* lebih besar atau sama dengan 0,05, maka hipotesis nol gagal ditolak, sehingga tidak terdapat cukup bukti statistik untuk menyatakan adanya perbedaan kualitas yang signifikan antara kedua kelompok.

IV. HASIL DAN PEMBAHASAN

Bagian ini menyajikan paparan objektif mengenai temuan penelitian yang membandingkan kualitas kode antara programmer manusia dan GitHub Copilot. Analisis mencakup tiga dimensi utama: kebenaran fungsional, kualitas kode statis (clean code), dan efisiensi algoritma, serta hasil pengujian hipotesis statistik.

A. Analisis Kebenaran Fungsional

Hasil pengujian otomatis menggunakan kerangka kerja Jest menunjukkan adanya variabilitas yang signifikan dalam tingkat keberhasilan penyelesaian masalah antar subjek penelitian. Berdasarkan rekapitulasi yang disajikan pada Tabel 1, partisipan manusia dengan profil performa tinggi (*Human*-2) mencatat tingkat keberhasilan fungsional tertinggi dengan *success rate* sebesar 50%, yang menunjukkan kemampuan superior dalam memenuhi kebutuhan fungsional sistem. Sebaliknya, partisipan manusia dengan profil performa rendah (*Human*-3) tidak berhasil menyelesaikan satu pun skenario uji, yang mengindikasikan adanya kesenjangan kemampuan yang cukup lebar antar pengembang manusia. Pada kelompok AI, GitHub Copilot dengan skenario *Zero-Shot Prompting* menunjukkan tingkat keberhasilan yang setara dengan *Human*-1, yaitu sebesar 33,3%. Namun, hasil yang paling mencolok ditunjukkan oleh skenario *Engineered Prompting*, di mana tingkat keberhasilan fungsional justru menurun secara drastis menjadi 16,7%, meskipun pendekatan ini dirancang untuk meningkatkan kualitas output melalui instruksi yang lebih terstruktur.

TABEL 1
REKAPITULASI TINGKAT KELULUSAN FUNGSIONAL

Subjek	Skenario	*Passed*	*Failed*	*Success Rate*
Human-1	Manual	2	4	33.3%
Human-2	Manual	3	3	50%
Human-3	Manual	0	6	0%
Copilot	*Zero-Shot*	2	4	33.3%
Copilot	*Engineered*	1	5	16.7%

Temuan tersebut mengungkap sebuah paradoks penting yang dalam penelitian ini diidentifikasi sebagai fenomena *"Clean Garbage"*. Fenomena ini terlihat paling jelas pada skenario *Engineered Prompting*, di mana AI mampu menghasilkan kode yang secara sintaksis sangat rapi, terstruktur, dan selaras dengan prinsip *Clean Code*, namun gagal memenuhi kebutuhan logika bisnis secara menyeluruh. Kode yang dihasilkan tampak bersih dan mudah dibaca, tetapi tidak mampu menangani kompleksitas logika implisit serta *edge cases* yang muncul dalam konteks permasalahan nyata. Kegagalan AI terutama didominasi oleh ketidakmampuannya dalam mengantisipasi aturan bisnis yang tidak dinyatakan secara eksplisit, seperti validasi stok negatif pada sistem inventaris, yang secara naluriah dan intuitif dapat ditangani dengan baik oleh pengembang manusia. Temuan ini menegaskan bahwa keteraturan struktural dan kualitas sintaksis tidak selalu berbanding lurus dengan kebenaran fungsional, serta memperkuat argumen bahwa evaluasi kualitas kode AI tidak dapat hanya bergantung pada metrik *Clean Code* semata, tetapi juga harus mempertimbangkan ketepatan logika dan pemahaman konteks domain.

B. Analisis Kualitas Kode dan Pengujian Hipotesis

Berbeda dengan hasil pengujian fungsional, analisis kualitas kode secara statis menggunakan platform SonarQube menunjukkan dominasi yang jelas dari GitHub Copilot. Untuk memvalidasi perbedaan tersebut secara kuantitatif, dilakukan pengujian hipotesis menggunakan Wilcoxon Signed-Rank Test dengan hipotesis nol (H_0) yang menyatakan bahwa tidak terdapat perbedaan *Cognitive Complexity* antara kode yang dihasilkan oleh pengembang manusia dan AI. Pendekatan statistik ini dipilih karena karakteristik data yang berpasangan serta tidak memenuhi asumsi normalitas distribusi, sehingga sesuai untuk mengevaluasi perbedaan median antar dua kelompok yang saling berkaitan.



GAMBAR 2
PERBANDINGAN DISTRIBUSI *COGNITIVE COMPLEXITY*

Visualisasi distribusi *Cognitive Complexity* yang ditampilkan pada Gambar 2 memperlihatkan perbedaan pola yang kontras antara kedua kelompok. Skor kompleksitas kognitif pada kode buatan manusia menunjukkan rentang yang jauh lebih lebar, mencerminkan variabilitas gaya dan pendekatan pemrograman antar individu. Sebaliknya, kode yang dihasilkan oleh GitHub Copilot menunjukkan distribusi skor yang konsisten rendah, yang mengindikasikan struktur kode yang lebih sederhana, modular, dan relatif seragam. Hasil uji statistik yang dirangkum pada Tabel 2 menunjukkan nilai *p-value* sebesar 0,0277 ($p < 0,05$) untuk perbandingan antara kelompok manusia dan Copilot, sehingga hipotesis nol ditolak. Penolakan H_0 ini membuktikan secara statistik bahwa GitHub Copilot secara konsisten menghasilkan kode dengan *Cognitive Complexity* yang lebih rendah dibandingkan pengembang manusia. Nilai statistik uji $W = 0,0$ semakin menegaskan konsistensi temuan tersebut, karena tidak ditemukan satu pun kasus di mana kode manusia memiliki kompleksitas kognitif yang lebih rendah dibandingkan kode yang dihasilkan oleh AI.

TABEL 2
HASIL UJI STATISTIK WILCOXON (*COGNITIVE COMPLEXITY*)

Pasangan Komprasi	Nilai W	P-Value	Keputusan (a = 0.05)
Human vs Copilot (*Zero-Shot*)	0.0	0.0277	Signifikan (H_0 Ditolak)
Human vs Copilot (*Engineered*)	0.0	0.0277	Signifikan (H_0 Ditolak)
Copilot (*Zero-Shot*) vs Copilot (*Engineered*)	5.0	0.2488	Tidak Signifikan

C. Analisis Efisiensi Algoritma

Analisis kualitatif terhadap kompleksitas waktu (*Time Complexity*) menggunakan notasi Big O menunjukkan dikotomi kompetensi.

1. Level Mikro (Implementasi), AI menunjukkan keunggulan dalam pemilihan algoritma standar. Pada masalah pengurutan data, AI secara otomatis mengimplementasikan Timsort dengan kompleksitas O(N log N). Sebaliknya, partisipan manusia tertentu (*Human*-3) mengimplementasikan algoritma manual (*Bubble Sort*) dengan kompleksitas $O(N^2)$ yang tidak efisien.
2. Level Makro (Strategi), manusia unggul dalam menentukan strategi penyelesaian masalah kontekstual. Pada masalah yang membutuhkan logika heuristik, manusia mampu merancang alur

logika yang tepat sasaran, sedangkan AI sering kali terjebak dalam halusinasi logika meskipun menggunakan struktur data yang efisien O(1) (Hash Map).

D. Pengembangan Penelitian ke Depan

Berdasarkan keterbatasan metodologis serta temuan empiris dalam penelitian ini, terdapat beberapa arah pengembangan yang berpotensi dilakukan pada penelitian selanjutnya guna memperkaya dan memperkuat generalisasi hasil. Salah satu arah utama adalah perluasan skala dan karakteristik partisipan dengan melibatkan praktisi profesional senior di bidang pengembangan perangkat lunak. Keterlibatan pengembang berpengalaman diharapkan dapat memberikan perspektif yang lebih representatif terhadap praktik industri, sekaligus memungkinkan analisis yang lebih mendalam mengenai apakah kesenjangan kualitas kode antara pengembang manusia dan AI cenderung menyempit, tetap, atau justru melebar ketika dibandingkan dengan subjek mahasiswa yang masih berada pada tahap pembelajaran.

Selain itu, penelitian di masa depan dapat memperluas cakupan eksperimen dengan menguji ragam bahasa pemrograman lain, seperti Python atau Go, yang memiliki paradigma, ekosistem, dan tingkat abstraksi yang berbeda dibandingkan bahasa yang digunakan dalam penelitian ini. Perluasan ini juga dapat dikombinasikan dengan perbandingan performa model large language model (LLM) generasi terbaru, seperti Claude 4.5 Sonnet atau GPT-5, untuk memvalidasi apakah fenomena "Clean Garbage" masih konsisten muncul pada konteks teknologi dan arsitektur model yang berbeda. Lebih lanjut, penambahan metrik keamanan melalui analisis kerentanan (security vulnerabilities) menjadi arah pengembangan yang krusial, mengingat kode yang dihasilkan oleh AI berpotensi membawa risiko injeksi kode atau celah keamanan tersembunyi. Dengan memasukkan dimensi keamanan sebagai variabel analisis, penelitian selanjutnya diharapkan mampu memberikan evaluasi yang lebih komprehensif terhadap kelayakan dan risiko penggunaan AI dalam pengembangan perangkat lunak nyata.

V. KESIMPULAN

Penelitian ini menyimpulkan bahwa GitHub Copilot menunjukkan keunggulan statistik yang signifikan pada aspek validitas struktural kode, khususnya dalam hal keterbacaan dan keteraturan alur kendali. Kode yang dihasilkan oleh AI secara konsisten memiliki nilai Cognitive Complexity yang lebih rendah serta struktur yang lebih modular dibandingkan dengan kode yang ditulis oleh programmer manusia. Temuan ini mengindikasikan bahwa model Large Language Models mampu menginternalisasi pola-pola sintaktik dan konvensi penulisan kode yang selaras dengan prinsip Clean Code, sehingga unggul dalam menghasilkan solusi yang tampak rapi, terstandarisasi, dan mudah dipelihara secara struktural.

Namun demikian, keunggulan AI dalam metrik Clean Code tersebut tidak menunjukkan hubungan yang linier dengan tingkat kebenaran fungsional (functional correctness). Hasil eksperimen menunjukkan bahwa partisipan manusia memiliki keunggulan yang signifikan dalam menerjemahkan kebutuhan bisnis yang bersifat implisit, ambigu, dan kompleks, serta dalam mengantisipasi dan menangani edge cases yang tidak secara eksplisit didefinisikan dalam spesifikasi awal. Sebaliknya, meskipun AI mampu menghasilkan sintaks yang valid dan secara visual memenuhi standar kualitas kode, model sering kali mengalami kegagalan logika, terutama pada skenario yang menuntut penalaran kontekstual, pemahaman tujuan sistem secara menyeluruh, serta pengambilan keputusan berbasis asumsi domain.

Temuan paling krusial dalam penelitian ini adalah teridentifikasinya fenomena yang disebut sebagai "Clean Garbage", khususnya pada skenario Engineered Prompting. Fenomena ini menggambarkan kondisi di mana instruksi yang terlalu terstruktur, rigid, dan berorientasi pada pemenuhan aturan sintaktik justru mendorong model untuk memprioritaskan kepatuhan terhadap struktur formal dibandingkan dengan kebenaran semantik dan fungsional. Akibatnya, tingkat keberhasilan fungsional solusi yang dihasilkan AI menurun secara signifikan karena model terjebak dalam halusinasi pemenuhan aturan (rule-compliance hallucination), alih-alih melakukan penalaran logis terhadap permasalahan yang diberikan. Hal ini menegaskan bahwa kualitas visual dan struktural kode tidak dapat dijadikan satu-satunya indikator keberhasilan solusi perangkat lunak yang dihasilkan oleh AI.

Dalam konteks efisiensi algoritma, AI menunjukkan dominasi pada level implementasi mikro dengan kecenderungan memilih algoritma standar yang telah teruji dan memiliki kompleksitas optimal berdasarkan praktik umum. Sebaliknya, pengembang manusia lebih unggul pada strategi level makro, seperti pemilihan pendekatan algoritmik yang selaras dengan kebutuhan bisnis, penyesuaian desain terhadap batasan sistem, serta pengambilan keputusan strategis yang mempertimbangkan trade-off jangka panjang. Perbedaan ini menunjukkan bahwa keunggulan AI saat ini masih bersifat operasional dan taktis, sementara manusia tetap memegang peran sentral pada level konseptual dan strategis.

Berdasarkan keseluruhan temuan tersebut, penelitian ini menegaskan bahwa adopsi Large Language Models dalam rekayasa perangkat lunak tidak dapat dipandang sebagai pengganti peran pengembang manusia secara penuh. Sebaliknya, pemanfaatan AI menuntut terjadinya transformasi peran pengembang menjadi arsitek solusi, pengambil keputusan strategis, serta validator logika dan kebenaran fungsional. Dengan demikian, pendekatan yang paling realistis dan berkelanjutan adalah membangun ekosistem kerja kolaboratif hibrida, di mana AI berperan sebagai akselerator produktivitas dan pembantu implementasi, sementara manusia tetap menjadi penjaga makna, tujuan, dan kualitas substantif dari sistem perangkat lunak yang dikembangkan.

REFERENSI

[1] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. Jiang. "GitHub Copilot AI pair programmer: Asset or Liability?". *J. Syst. Softw.*, vol. 203, 2023.

[2] B. Yetistiren, I. Ozsoy, and E. Tuzun. "Assessing the quality of GitHub copilot's code generation". *Proc. 18th Int. Conf. Predict. Model. Data Anal. Softw. Eng. (PROMISE)*, pp. 62–71, 2022.

[3] N. Nguyen and S. Nadi. "An Empirical Evaluation of GitHub Copilot's Code Suggestions". *Proc. Mining Softw. Repos. (MSR)*, vol. 1, no. 1, 2022.

[4] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. (2023). "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot". *arXiv preprint arXiv:2302.06590*. [Online]. Available: http://arxiv.org/abs/2302.06590

[5] M. M. Barón, M. Wyrich, and S. Wagner. "An empirical validation of cognitive complexity as a

measure of source code understandability". *Int. Symp. Empir. Softw. Eng. Meas.*, 2020.

[6] M. A. Florez Muñoz, J. C. Jaramillo De La Torre, S. Pareja López, S. Herrera, and C. A. Candela Uribe. "Comparative Study of AI Code Generation Tools: Quality Assessment and Performance Analysis". *LatIA*, vol. 2, p. 104, 2024.

[7] D. Alawad, M. Panta, M. Zibran, and M. R. Islam. "An empirical study of the relationships between code readability and software complexity". *27th Int. Conf. Softw. Eng. Data Eng. (SEDE)*, pp. 122–127, 2018.

[8] W. Takerngsaksiri, M. Fu, C. Tantithamthavorn, J. Pasuksmit, K. Chen, and M. Wu. (2025, June). "Code Readability in the Age of Large Language Models: An Industrial Case Study from Atlassian". *Companion Proc. 33rd ACM Symp. Found. Softw. Eng. (FSE '25)*. [Online]. Vol. 1. Available: https://arxiv.org/abs/2501.11264v1

[9] [9] D. Tosi. "Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation". *Future Internet*, vol. 16, no. 6, 2024.

[10] D. Smit, H. Smuts, P. Louw, J. Pielmeier, and C. Eidelloth. "The impact of GitHub Copilot on developer productivity from a software engineering body of knowledge perspective". *Proc. 30th Americas Conf. Inf. Syst. (AMCIS)*, 2024.

[11] R. P. Buse and W. Weimer. "Learning a Metric for Code Readability". *IEEE Trans. Softw. Eng.*, vol. 36, no. 4, pp. 546–558, 2010.

[12] T. Heričko and B. Šumak. "Exploring Maintainability Index Variants for Software Maintainability Measurement in Object-Oriented Systems". *Appl. Sci.*, vol. 13, no. 5, p. 2972, 2023.

[13] D. G. Paul, H. Zhu, and I. Bayley. (2024). "Benchmarks and Metrics for Evaluations of Code Generation: A Critical Review". *arXiv preprint arXiv:2406.12655*. [Online].

[14] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün. (2023). "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT". *arXiv preprint arXiv:2304.10778*. [Online].

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022.

[16] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". *ACM Comput. Surv.*, vol. 55, no. 9, art. 195, 2023.