GEAR: General Engineering, Applied Science and Research Journal, Vol.1, Issue-2, 2026
Website: https://publications.corejurnal.com/index.php/gear/
E-ISSN: 3124-4092

Kajian Pustaka tentang Peran Algoritma *Machine Learning* dalam Optimalisasi Proses Pemrograman

Anggie Dwi Putra[1*], Micheal Gunawan[2]
[1*]*Universitas Mercu Buana, Indonesia, anggidputra567@gmail.com*
[2]*Universitas Mercu Buana, Indonesia, michealgunawan05@gmail.com*

Abstrak
Penelitian ini bertujuan untuk mengkaji secara komprehensif peran algoritma machine learning dalam optimalisasi proses pemrograman, meliputi aspek code generation, deteksi dan perbaikan bug, pengujian perangkat lunak, serta optimasi kode. Metode yang digunakan adalah penelitian kualitatif dengan pendekatan deskriptif melalui studi pustaka, di mana data dikumpulkan melalui penelusuran literatur dari artikel ilmiah, prosiding konferensi, dan dokumen akademik yang relevan. Analisis data dilakukan secara sistematis melalui tahapan identifikasi tema, reduksi data, kategorisasi konsep, dan penarikan kesimpulan secara induktif untuk memperoleh pemahaman yang mendalam. Hasil kajian menunjukkan bahwa machine learning mampu meningkatkan efisiensi, akurasi, dan produktivitas dalam pengembangan perangkat lunak melalui otomatisasi proses pemrograman, peningkatan kualitas pengujian, serta optimalisasi performa kode. Selain itu, pendekatan berbasis deep learning, reinforcement learning, dan model hybrid terbukti memberikan kontribusi signifikan dalam mengatasi kompleksitas sistem perangkat lunak modern. Implikasi penelitian ini menunjukkan bahwa integrasi machine learning dalam rekayasa perangkat lunak dapat mentransformasi praktik pemrograman menjadi lebih adaptif dan berbasis data. Dengan demikian, penelitian ini berkontribusi dalam memperkaya pemahaman teoretis dan memberikan dasar praktis bagi pengembangan teknologi pemrograman berbasis kecerdasan buatan di masa depan.

Kata kunci: machine learning, pemrograman, optimasi kode, rekayasa perangkat lunak, kecerdasan buatan

A Literature Review on the Role of Machine Learning Algorithms in Optimizing the Programming Process

Abstract
This study aims to comprehensively examine the role of machine learning algorithms in optimizing the programming process, including code generation, bug detection and correction, software testing, and code optimization. The method used is qualitative research with a descriptive approach through a literature review, in which data was collected by reviewing scientific articles, conference proceedings, and relevant academic documents. Data analysis was conducted systematically through the stages of theme identification, data reduction, concept categorization, and inductive conclusion drawing to gain a deep understanding. The results of the study indicate that machine learning can enhance efficiency, accuracy, and productivity in software development through the automation of programming processes, improved testing quality, and code performance optimization. Furthermore, approaches based on deep learning, reinforcement learning, and hybrid models have proven to make significant contributions to addressing the complexity of modern software systems. The implications of this research suggest that the integration of machine learning into software engineering can transform programming practices into more adaptive and data-driven ones. Thus, this research contributes to enriching theoretical understanding and provides a practical foundation for the future development of artificial intelligence-based programming technologies.

Keywords: machine learning, programming, code optimization, software engineering, artificial intelligence

PENDAHULUAN

Perkembangan teknologi informasi dalam beberapa dekade terakhir telah mendorong transformasi signifikan dalam berbagai bidang, termasuk dalam proses pengembangan perangkat lunak. Salah satu inovasi paling berpengaruh adalah penerapan machine learning sebagai bagian dari kecerdasan buatan yang mampu mengotomatisasi proses analisis, prediksi, dan pengambilan keputusan dalam sistem komputasi. Machine learning tidak hanya berfungsi sebagai alat analisis data, tetapi juga telah berkembang menjadi komponen penting dalam meningkatkan efisiensi dan kualitas proses pemrograman (Tawar, 2024).

Dalam konteks rekayasa perangkat lunak, machine learning telah digunakan secara luas untuk mendukung berbagai tahapan dalam Software Development Life Cycle. Mulai dari analisis kebutuhan, desain sistem, hingga pengujian dan pemeliharaan, algoritma machine learning mampu memberikan kontribusi yang signifikan dalam meningkatkan produktivitas pengembang serta kualitas perangkat lunak yang dihasilkan (Chen et al., 2024). Hal ini menunjukkan bahwa integrasi machine learning dalam proses pemrograman bukan lagi sekadar tren, melainkan kebutuhan yang semakin mendesak.

Tren terkini menunjukkan bahwa penggunaan machine learning dalam pengembangan perangkat lunak terus meningkat seiring dengan bertambahnya kompleksitas sistem yang dikembangkan. Sistem perangkat lunak modern cenderung memiliki struktur yang kompleks dan dinamis, sehingga metode konvensional sering kali tidak mampu mengatasi tantangan tersebut secara efektif. Oleh karena itu, pendekatan berbasis machine learning menjadi solusi yang menjanjikan untuk mengoptimalkan berbagai proses dalam pemrograman (Hoffman & Brooks, 2025).

Salah satu aspek penting dalam pemrograman yang mengalami transformasi signifikan adalah proses deteksi dan perbaikan kesalahan kode. Metode tradisional yang bergantung pada debugging manual cenderung memakan waktu dan rentan terhadap kesalahan manusia. Sebaliknya, machine learning memungkinkan deteksi bug secara otomatis dengan tingkat akurasi yang tinggi, serta mampu memberikan rekomendasi perbaikan secara real-time (Imran et al., 2025).

Selain itu, penerapan machine learning juga memungkinkan optimalisasi dalam proses pengujian perangkat lunak. Dengan memanfaatkan teknik seperti predictive testing dan adaptive testing, algoritma machine learning dapat meningkatkan efisiensi pengujian serta mengurangi waktu debugging hingga 37 persen, sekaligus menekan biaya pemeliharaan hingga 50 persen (Jawalkar, 2021). Hal ini menunjukkan potensi besar machine learning dalam meningkatkan kualitas dan efisiensi proses pengembangan perangkat lunak.

Di sisi lain, kemajuan dalam bidang deep learning telah membuka peluang baru dalam otomatisasi pemrograman, khususnya dalam code generation. Model berbasis transformer seperti AlphaCode menunjukkan kemampuan untuk menghasilkan kode secara otomatis dengan performa yang mendekati kemampuan manusia dalam kompetisi pemrograman (Li et al., 2022). Inovasi ini menandai pergeseran paradigma dari pemrograman manual menuju pemrograman berbasis kecerdasan buatan.

Namun demikian, meskipun machine learning menawarkan berbagai keunggulan, terdapat sejumlah tantangan yang masih perlu diatasi. Salah satunya adalah kompleksitas

dalam proses optimasi model machine learning itu sendiri, yang memerlukan teknik khusus untuk mencapai performa yang optimal. Berbagai metode optimasi seperti gradient descent dan Bayesian optimization terus dikembangkan untuk meningkatkan efisiensi pelatihan model (Karthick, 2024).

Selain itu, algoritma optimasi berbasis evolusi juga menjadi salah satu pendekatan yang banyak digunakan dalam machine learning. Algoritma seperti genetic algorithm dan particle swarm optimization terbukti efektif dalam menangani permasalahan optimasi yang kompleks dan berdimensi tinggi. Namun, tantangan seperti konvergensi lokal dan kompleksitas komputasi masih menjadi kendala dalam implementasinya (Farhadi, 2022).

Dalam konteks pemrograman, optimalisasi tidak hanya terbatas pada model machine learning, tetapi juga mencakup optimalisasi kode program itu sendiri. Pendekatan berbasis reinforcement learning telah digunakan untuk mengotomatisasi proses optimasi kode, sehingga mampu meningkatkan performa eksekusi program secara signifikan (Lamouri et al., 2025). Hal ini menunjukkan bahwa machine learning memiliki peran strategis dalam meningkatkan efisiensi sistem secara keseluruhan.

Lebih lanjut, penggunaan machine learning dalam compiler modern juga menunjukkan perkembangan yang pesat. Teknologi seperti auto-tuning dan deep learning compiler memungkinkan optimalisasi kode secara otomatis dengan mempertimbangkan berbagai faktor seperti penggunaan memori dan efisiensi komputasi (Zhao et al., 2025). Inovasi ini semakin memperkuat peran machine learning dalam proses pemrograman.

Meskipun berbagai penelitian telah menunjukkan keberhasilan penerapan machine learning dalam pemrograman, masih terdapat kesenjangan antara teori dan praktik. Studi terbaru menunjukkan bahwa lebih dari 50 persen permasalahan debugging dalam sistem machine learning belum sepenuhnya teratasi oleh penelitian yang ada (Nguyen et al., 2025). Hal ini menunjukkan adanya kebutuhan untuk penelitian lebih lanjut dalam bidang ini.

Selain itu, keterbatasan dalam interpretabilitas model machine learning juga menjadi tantangan yang signifikan. Banyak model yang bersifat black-box, sehingga sulit untuk dipahami dan diinterpretasikan oleh pengembang. Hal ini dapat menghambat proses debugging dan pengambilan keputusan dalam pengembangan perangkat lunak (Chen, 2025).

Permasalahan lain yang tidak kalah penting adalah kebutuhan akan data dalam jumlah besar untuk melatih model machine learning. Ketersediaan data yang terbatas dapat mempengaruhi performa model, sehingga diperlukan pendekatan baru untuk mengatasi masalah ini, seperti data augmentation dan transfer learning (More & Bradbury, 2025).

Berdasarkan uraian tersebut, dapat disimpulkan bahwa meskipun machine learning memiliki potensi besar dalam mengoptimalkan proses pemrograman, masih terdapat berbagai tantangan dan kesenjangan yang perlu diatasi. Oleh karena itu, diperlukan kajian pustaka yang komprehensif untuk memahami peran algoritma machine learning dalam konteks ini.

Artikel ini bertujuan untuk mengkaji secara sistematis peran algoritma machine learning dalam optimalisasi proses pemrograman, termasuk dalam aspek code generation, bug detection, testing, dan code optimization. Selain itu, artikel ini juga bertujuan untuk mengidentifikasi tantangan dan peluang yang ada dalam penerapan machine learning di bidang rekayasa perangkat lunak.

Secara teoretis, kajian ini diharapkan dapat memberikan kontribusi dalam pengembangan konsep dan pemahaman mengenai integrasi machine learning dalam pemrograman. Secara praktis, hasil kajian ini diharapkan dapat menjadi referensi bagi pengembang perangkat lunak dan peneliti dalam mengimplementasikan algoritma machine learning secara efektif untuk meningkatkan kualitas dan efisiensi proses pemrograman.

METODE

Penelitian dalam artikel ini menggunakan pendekatan kualitatif dengan desain deskriptif melalui metode studi pustaka. Pendekatan kualitatif dipilih karena mampu memberikan pemahaman mendalam terhadap fenomena yang kompleks, khususnya terkait peran algoritma machine learning dalam optimalisasi proses pemrograman. Metode ini menekankan pada eksplorasi makna, interpretasi konsep, serta analisis kontekstual terhadap berbagai sumber ilmiah yang relevan. Dalam perkembangan terbaru, penelitian kualitatif menuntut adanya transparansi, sistematika, dan ketelitian dalam proses analisis agar hasil penelitian memiliki tingkat kredibilitas yang tinggi (Bingham, 2023; Pratt, 2025).

Pendekatan deskriptif digunakan untuk menggambarkan secara sistematis berbagai konsep, teori, dan temuan penelitian terkait machine learning dalam pemrograman tanpa melakukan manipulasi variabel. Pendekatan ini memungkinkan peneliti untuk menyajikan fenomena secara komprehensif berdasarkan data yang tersedia dalam literatur. Selain itu, pendekatan deskriptif juga memberikan fleksibilitas dalam mengintegrasikan berbagai perspektif penelitian sehingga dapat menghasilkan pemahaman yang utuh terhadap topik yang dikaji (Doyle et al., 2019; Abraham & P, 2024).

Metode studi pustaka dalam penelitian ini dilakukan dengan mengumpulkan dan menganalisis berbagai sumber data sekunder yang relevan, seperti artikel jurnal ilmiah, prosiding konferensi, laporan penelitian, serta dokumen akademik lainnya. Sumber-sumber tersebut dipilih berdasarkan relevansi dengan topik machine learning dalam rekayasa perangkat lunak, khususnya yang membahas code generation, bug detection, dan code optimization. Studi pustaka menjadi metode yang efektif untuk mengkaji perkembangan teori dan praktik terkini, serta memetakan kontribusi penelitian dalam bidang yang sedang berkembang pesat (Togia & Malliari, 2017; Jimenez et al., 2024).

Strategi pencarian literatur dilakukan secara sistematis melalui beberapa basis data ilmiah dan repositori akademik, seperti Google Scholar, ScienceDirect, IEEE Xplore, SpringerLink, ACM Digital Library, dan arXiv. Pencarian literatur difokuskan pada publikasi yang relevan dengan topik machine learning dalam pemrograman dan rekayasa perangkat lunak. Kata kunci yang digunakan dalam proses pencarian meliputi “machine learning in programming”, “machine learning for software engineering”, “code generation using machine learning”, “bug detection machine learning”, “code optimization machine learning”, “AI-assisted programming”, dan “automated programming tools”. Untuk memperluas cakupan pencarian, kata kunci tersebut juga dikombinasikan menggunakan operator Boolean, seperti AND dan OR, misalnya “machine learning AND code generation”, “bug detection AND machine learning”, serta “software engineering AND artificial intelligence”.

Seleksi literatur dilakukan melalui beberapa tahapan. Tahap pertama adalah identifikasi awal terhadap artikel berdasarkan judul, abstrak, dan kata kunci. Tahap kedua adalah penyaringan literatur berdasarkan kesesuaian topik, tahun publikasi, jenis sumber, dan ketersediaan teks lengkap. Tahap ketiga adalah penilaian kelayakan isi literatur dengan membaca bagian pendahuluan, metode, hasil, dan kesimpulan untuk memastikan bahwa artikel memiliki kontribusi langsung terhadap fokus penelitian. Literatur yang memenuhi kriteria kemudian dianalisis lebih lanjut untuk memperoleh data konseptual dan temuan utama yang relevan dengan tujuan penelitian.

Kriteria inklusi dalam penelitian ini mencakup: 1) literatur yang membahas penerapan machine learning dalam pemrograman atau rekayasa perangkat lunak; 2) literatur yang secara khusus berkaitan dengan code generation, bug detection, atau code optimization; 3) artikel ilmiah, prosiding konferensi, laporan penelitian, atau dokumen akademik yang dapat dipertanggungjawabkan; 4) literatur yang tersedia dalam teks lengkap; dan 5) publikasi yang diterbitkan dalam kurun waktu terbaru agar sesuai dengan perkembangan teknologi. Sementara itu, kriteria eksklusi meliputi: 1) literatur yang tidak memiliki relevansi langsung dengan topik penelitian; 2) artikel populer, opini, atau sumber non-akademik; 3) literatur yang tidak menjelaskan metode atau temuan secara jelas; 4) dokumen duplikat; dan 5) publikasi yang hanya membahas machine learning secara umum tanpa keterkaitan dengan pemrograman atau rekayasa perangkat lunak.

Teknik pengumpulan data dilakukan melalui penelusuran literatur, dokumentasi, dan pencatatan informasi penting dari setiap sumber yang terpilih. Informasi yang dikumpulkan meliputi identitas publikasi, tujuan penelitian, metode yang digunakan, fokus kajian, temuan utama, serta relevansinya dengan optimalisasi proses pemrograman. Data yang diperoleh kemudian disusun dalam kategori tematik untuk memudahkan proses analisis. Kajian teoritis juga digunakan untuk mengintegrasikan berbagai konsep yang diperoleh dari literatur sehingga menghasilkan kerangka pemikiran yang terstruktur (Granikov et al., 2020; Bandaranayake, 2024).

Prosedur analisis data dilakukan melalui beberapa tahapan, yaitu identifikasi tema, reduksi data, kategorisasi konsep, sintesis temuan, dan penarikan kesimpulan secara induktif. Tahap identifikasi tema bertujuan untuk menemukan pola dan topik utama dalam literatur. Reduksi data dilakukan dengan menyaring informasi yang relevan dengan tujuan penelitian. Selanjutnya, kategorisasi konsep dilakukan dengan mengelompokkan temuan berdasarkan tema tertentu, seperti optimalisasi algoritma, otomatisasi pemrograman, code generation, bug detection, dan code optimization. Setelah itu, sintesis temuan dilakukan dengan membandingkan persamaan, perbedaan, kelebihan, dan keterbatasan dari masing-masing literatur. Tahap akhir adalah penarikan kesimpulan secara induktif untuk memperoleh pemahaman yang komprehensif terhadap fenomena yang dikaji (Belotto, 2018; Fife & Gossner, 2024; Vila-Henninger et al., 2022).

HASIL DAN PEMBAHASAN

Hasil menunjukkan bahwa algoritma machine learning memiliki peran yang sangat signifikan dalam mengoptimalkan berbagai aspek proses pemrograman. Berdasarkan analisis terhadap

literatur yang digunakan, ditemukan bahwa penerapan machine learning tidak hanya terbatas pada satu tahap pengembangan perangkat lunak, tetapi mencakup seluruh siklus pengembangan mulai dari penulisan kode, pengujian, hingga optimasi performa sistem. Temuan ini menegaskan bahwa machine learning berfungsi sebagai teknologi kunci dalam meningkatkan efisiensi dan kualitas perangkat lunak secara menyeluruh (Tawar, 2024).

Salah satu temuan utama adalah kemampuan machine learning dalam mengotomatisasi proses code generation. Model berbasis deep learning seperti transformer telah menunjukkan performa yang sangat kompetitif dalam menghasilkan kode program. Sistem seperti AlphaCode mampu mencapai tingkat performa mendekati manusia dalam kompetisi pemrograman dengan memanfaatkan teknik generasi kode dalam skala besar dan proses seleksi hasil yang optimal (Li et al., 2022). Selain itu, pendekatan multi-agent seperti Blueprint2Code mampu meningkatkan akurasi dan reliabilitas kode yang dihasilkan dengan pendekatan iteratif yang menyerupai alur kerja manusia (Mao et al., 2025).

Temuan berikutnya menunjukkan bahwa machine learning berperan penting dalam deteksi dan perbaikan bug secara otomatis. Pendekatan berbasis pembelajaran seperti DeepBugs mampu mencapai tingkat akurasi antara 89 persen hingga 95 persen dalam mendeteksi kesalahan kode dengan waktu analisis yang sangat cepat (Pradel & Sen, 2018). Selain itu, model berbasis Random Forest dan Neural Network menunjukkan performa yang tinggi dalam mendeteksi bug dengan akurasi hingga 86.67 persen dan AUC sebesar 0.97, meskipun terdapat perbedaan dalam efisiensi dan waktu pelatihan (Imran et al., 2025). Hal ini menunjukkan bahwa pemilihan algoritma sangat bergantung pada kebutuhan dan konteks penggunaan.

Dalam konteks pengujian perangkat lunak, integrasi machine learning terbukti mampu meningkatkan efisiensi secara signifikan. Model predictive testing dan adaptive testing mampu mengurangi waktu debugging hingga 37 persen serta menekan biaya pemeliharaan hingga 50 persen. Selain itu, penggunaan reinforcement learning dalam pemilihan test case dapat mengurangi eksekusi pengujian yang redundan hingga 43 persen (Jawalkar, 2021). Temuan ini menunjukkan bahwa machine learning tidak hanya meningkatkan akurasi pengujian, tetapi juga efisiensi operasional.

Selanjutnya, hasil kajian menunjukkan bahwa algoritma optimasi memainkan peran penting dalam meningkatkan performa model machine learning dan kode program. Teknik optimasi seperti gradient descent, adaptive learning rate, dan Bayesian optimization digunakan untuk meningkatkan efisiensi pelatihan model serta mengatasi masalah overfitting dan kompleksitas komputasi (Karthick, 2024). Selain itu, algoritma evolusioner seperti genetic algorithm dan particle swarm optimization terbukti efektif dalam menangani permasalahan optimasi yang kompleks, meskipun masih menghadapi tantangan dalam konvergensi dan efisiensi komputasi (Farhadi, 2022).

Dalam aspek optimasi kode program, pendekatan berbasis reinforcement learning menunjukkan hasil yang sangat menjanjikan. Framework seperti Pearl mampu meningkatkan performa eksekusi program hingga 2.02 kali lebih cepat dibandingkan metode konvensional, serta hingga 3.36 kali dibandingkan pendekatan baseline lainnya (Lamouri et al., 2025). Selain itu, penggunaan deep learning compiler seperti Sifter mampu mengurangi waktu optimasi

hingga 41 persen dan mengeliminasi 52 persen konfigurasi yang tidak efisien tanpa menurunkan kualitas hasil optimasi (Zhao et al., 2025).

Tabel 1. Temuan Utama Peran Machine Learning dalam Optimalisasi Proses Pemrograman

Aspek	Metode/Model	Hasil Utama	Sumber
Code generation	AlphaCode	Mampu menghasilkan kode program dengan performa kompetitif dan mencapai performa 54,3% dalam kompetisi pemrograman.	Li et al. (2022)
Code generation	Blueprint2Code	Meningkatkan akurasi kode hingga 96,3% pada HumanEval melalui pendekatan ***multi-agent*** dan proses iteratif.	Mao et al. (2025)
Bug detection	DeepBugs	Mampu mendeteksi kesalahan kode dengan tingkat akurasi 89%–95%.	Pradel & Sen (2018)
Bug detection	*Random Forest* dan *Neural Network*	Menghasilkan akurasi deteksi *bug* sebesar 86,67% dan AUC 0,97.	Imran et al. (2025)
Software testing	ML-based QA, ***predictive testing***, dan ***adaptive testing***	Mengurangi waktu ***debugging*** hingga 37%, menekan biaya pemeliharaan hingga 50%, dan mengurangi pengujian redundan hingga 43%.	Jawalkar (2021)
Optimasi model ML	*Gradient descent*, *adaptive learning rate*, dan *Bayesian optimization*	Meningkatkan efisiensi pelatihan model serta membantu mengatasi *overfitting* dan kompleksitas komputasi.	Karthick (2024)
Optimasi algoritmik	*Genetic algorithm* dan *particle swarm optimization*	Efektif untuk permasalahan optimasi kompleks, tetapi masih menghadapi tantangan konvergensi dan efisiensi komputasi.	Farhadi (2022)
Code optimization	Pearl berbasis ***reinforcement learning***	Meningkatkan performa eksekusi program hingga 2,02 kali dibandingkan metode konvensional dan hingga 3,36 kali dibandingkan ***baseline***.	Lamouri et al. (2025)
Compiler optimization	Sifter berbasis ***deep learning compiler***	Mengurangi waktu optimasi hingga 41% dan mengeliminasi 52% konfigurasi yang tidak efisien tanpa menurunkan kualitas hasil optimasi.	Zhao et al. (2025)
Analisis dan deteksi kode	*Neuro-symbolic AI*	Meningkatkan akurasi deteksi ***bug*** hingga 15% dan kualitas analisis kode sebesar 22% dibandingkan metode konvensional.	Chen (2025)
Automated program repair	Pendekatan berbasis ***deep learning***	Mampu memperbaiki hingga 50% kesalahan kompilasi secara otomatis, dengan keberhasilan 86% dalam tiga rekomendasi teratas.	Mesbah et al. (2019)

Selain itu, temuan lain menunjukkan bahwa pendekatan hybrid seperti neuro-symbolic AI mampu meningkatkan akurasi deteksi bug hingga 15 persen dan meningkatkan kualitas analisis kode sebesar 22 persen dibandingkan metode konvensional (Chen, 2025). Pendekatan ini menggabungkan kekuatan pembelajaran statistik dengan logika simbolik sehingga menghasilkan model yang lebih akurat dan interpretatif.

Penelitian juga menunjukkan adanya perkembangan signifikan dalam automated program repair. Teknik berbasis deep learning mampu memperbaiki hingga 50 persen kesalahan kompilasi secara otomatis, dengan tingkat keberhasilan mencapai 86 persen dalam tiga

rekomendasi teratas (Mesbah et al., 2019). Hal ini menunjukkan bahwa machine learning tidak hanya mampu mendeteksi kesalahan, tetapi juga memberikan solusi perbaikan yang efektif.

Jika dibandingkan dengan pendekatan tradisional, seluruh penelitian menunjukkan bahwa machine learning memberikan peningkatan signifikan dalam hal efisiensi, akurasi, dan skalabilitas. Metode tradisional seperti debugging manual dan rule-based testing cenderung lebih lambat, kurang adaptif, dan rentan terhadap kesalahan manusia. Sebaliknya, pendekatan machine learning mampu belajar dari data historis dan beradaptasi dengan perubahan sistem secara dinamis.

Namun demikian, hasil kajian juga mengungkap beberapa keterbatasan. Beberapa model masih menghadapi masalah interpretabilitas, kebutuhan data yang besar, serta kompleksitas komputasi yang tinggi. Selain itu, performa model sering kali menurun pada kasus yang lebih kompleks atau pada data yang tidak representatif (Liu et al., 2024). Hal ini menunjukkan bahwa meskipun machine learning memiliki potensi besar, masih diperlukan pengembangan lebih lanjut.

Secara keseluruhan, hasil penelitian menunjukkan bahwa algoritma machine learning memberikan kontribusi yang signifikan dalam optimalisasi proses pemrograman. Temuan-temuan ini secara langsung menjawab tujuan penelitian, yaitu mengidentifikasi peran dan efektivitas machine learning dalam meningkatkan kualitas dan efisiensi pengembangan perangkat lunak. Hasil ini juga menjadi dasar yang kuat untuk pembahasan lebih lanjut mengenai implikasi dan pengembangan di masa depan.

Pembahasan

Temuan kajian pustaka ini menunjukkan bahwa peran algoritma machine learning dalam optimalisasi proses pemrograman dapat dipahami melalui tiga kerangka utama, yaitu otomatisasi, prediksi, dan optimasi adaptif. Dalam kerangka otomatisasi, machine learning memperluas kemampuan sistem untuk menghasilkan, memperbaiki, dan mentransformasikan kode secara lebih mandiri. Dalam kerangka prediksi, machine learning membantu mengidentifikasi cacat perangkat lunak, potensi kegagalan, dan kebutuhan pengujian secara lebih dini. Sementara itu, dalam kerangka optimasi adaptif, algoritma pembelajaran digunakan untuk memilih strategi kompilasi, tuning, dan perbaikan kode yang paling efektif sesuai konteks program. Dengan demikian, hasil kajian ini menegaskan bahwa machine learning tidak lagi sekadar alat bantu tambahan, melainkan telah menjadi mekanisme inti dalam peningkatan kualitas dan efisiensi rekayasa perangkat lunak (Tawar, 2024; Chen et al., 2024).

Pada aspek code generation, hasil kajian memperlihatkan bahwa model berbasis deep learning dan large language model telah menggeser paradigma pemrograman dari aktivitas yang sepenuhnya manual menuju kolaborasi manusia dan mesin. Temuan dari AlphaCode menunjukkan bahwa model dapat mencapai performa kompetitif dalam problem solving pemrograman, yang mengindikasikan bahwa representasi kode berbasis transformer telah mampu menangkap pola sintaktik dan sebagian penalaran algoritmik (Li et al., 2022). Hasil ini diperkuat oleh CodeT5 yang menunjukkan bahwa pemahaman identifier dan relasi semantik dalam kode berkontribusi pada peningkatan performa dalam berbagai tugas pemahaman dan generasi kode (Wang et al., 2021). Selain itu, pendekatan multi agen seperti Blueprint2Code

memperlihatkan bahwa akurasi generasi kode meningkat ketika proses pemrograman dimodelkan sebagai rangkaian tahap memahami masalah, menyusun rencana, menulis kode, dan memperbaiki kesalahan (Mao et al., 2025). Secara teoretis, hal ini sejalan dengan pandangan bahwa kualitas keluaran sistem cerdas meningkat ketika pembelajaran statistik dipadukan dengan struktur proses yang menyerupai strategi pemecahan masalah manusia.

Pada aspek deteksi bug dan perbaikan otomatis, hasil kajian menunjukkan bahwa machine learning memberikan kontribusi kuat pada peningkatan efisiensi debugging dan quality assurance. DeepBugs membuktikan bahwa unsur linguistik dalam kode seperti nama variabel dan fungsi mengandung informasi semantik yang bernilai untuk deteksi kesalahan, sehingga bug detection tidak harus hanya bergantung pada aturan statis tradisional (Pradel & Sen, 2018). Temuan ini diperkuat oleh studi berbasis AST, Random Forest, dan Neural Network yang menunjukkan performa tinggi dalam klasifikasi kode bermasalah, meskipun masing masing model memiliki trade off yang berbeda antara stabilitas, sensitivitas, dan waktu pelatihan (Imran et al., 2025). Di sisi lain, kajian automated program repair menegaskan bahwa pendekatan berbasis pembelajaran dapat melampaui deteksi dan masuk ke tahap rekomendasi atau bahkan perbaikan kode secara otomatis, sebagaimana terlihat pada DeepDelta dan survei tentang learning based automated program repair (Mesbah et al., 2019; Zhang et al., 2023). Secara konseptual, hasil ini memperlihatkan bahwa pembelajaran dari repositori kode historis memungkinkan sistem mengenali pola bug fixing yang berulang, sehingga pemeliharaan perangkat lunak menjadi lebih cepat dan lebih skalabel.

Dalam konteks pengujian perangkat lunak, hasil yang ditemukan memperkuat gagasan bahwa machine learning bekerja paling efektif ketika diintegrasikan ke dalam alur kerja pengembangan yang berkelanjutan. Studi tentang machine learning dalam quality assurance menunjukkan adanya penurunan waktu debugging, pengurangan biaya pemeliharaan, dan peningkatan cakupan pengujian melalui predictive testing, adaptive testing, dan seleksi test case berbasis reinforcement learning (Jawalkar, 2021). Temuan ini juga mendukung arah perkembangan deep learning based software engineering yang menempatkan testing, fault localization, dan maintenance sebagai area yang paling terdampak oleh kemajuan model pembelajaran mendalam (Chen et al., 2024). Implikasi praktisnya sangat besar karena organisasi pengembang dapat memanfaatkan machine learning untuk mempercepat siklus rilis, mengurangi beban kerja manual, dan meningkatkan keandalan sistem pada lingkungan agile maupun CI/CD. Dari sisi akademik, hasil ini memperkaya pemahaman bahwa kualitas perangkat lunak dapat dianalisis sebagai persoalan prediksi berbasis data, bukan semata persoalan verifikasi prosedural.

Pada aspek optimasi kode dan kompilasi, hasil kajian menunjukkan bahwa machine learning tidak hanya membantu menulis kode yang benar, tetapi juga mendorong kode menjadi lebih efisien. Pearl memperlihatkan bahwa reinforcement learning dapat digunakan untuk memilih urutan optimasi kompiler secara otomatis dan menghasilkan peningkatan performa eksekusi yang berarti (Lamouri et al., 2025). Sifter dan IntelliGen menunjukkan bahwa auto tuning berbasis pembelajaran dan eksplorasi ruang desain yang lebih cerdas mampu mengurangi konfigurasi yang tidak perlu serta mempercepat proses optimasi tanpa kehilangan kualitas hasil (Zhao et al., 2025; Ma et al., 2025). Temuan ini konsisten dengan TVM dan

oneDNN Graph Compiler yang menekankan pentingnya pendekatan compiler cerdas untuk mencapai portabilitas performa dan optimasi end to end pada beban kerja deep learning yang semakin kompleks (T. Chen et al., 2018; J. Li et al., 2023). Kontribusi ilmiah dari temuan ini adalah penegasan bahwa optimasi program kini bergeser dari domain heuristik manual ke domain pembelajaran berbasis pengalaman dan eksplorasi adaptif.

Walaupun hasil kajian sangat mendukung efektivitas machine learning, beberapa faktor turut memengaruhi capaian tersebut. Faktor pendukung meliputi ketersediaan data kode dalam skala besar, kemajuan arsitektur transformer, meningkatnya kapasitas komputasi, dan penggunaan representasi struktur program seperti AST, graph, dan intermediate representation. Faktor faktor ini memungkinkan model belajar lebih baik terhadap hubungan sintaksis, semantik, dan konteks eksekusi (Wang et al., 2021; J. Li et al., 2023). Namun, terdapat pula faktor yang dapat menahan performa, seperti kualitas data pelatihan yang tidak merata, bias evaluasi, data duplication pada benchmark, serta lemahnya generalisasi pada tugas yang lebih kompleks atau di luar distribusi pelatihan (Liu et al., 2024). Selain itu, beberapa pendekatan menunjukkan performa sangat baik pada skenario tertentu tetapi menurun pada kasus multi langkah, multi kelas, atau kebutuhan reasoning yang lebih dalam, sebagaimana terlihat pada evaluasi GPT 4 Vision untuk UML based code generation dan analisis reliabilitas model program generation (Antal et al., 2024; Liu et al., 2024). Ini berarti hasil yang tinggi pada benchmark tidak selalu identik dengan kesiapan penuh untuk penggunaan luas di dunia nyata.

Temuan kajian ini juga mengungkap adanya ketegangan antara akurasi dan interpretabilitas. Model berbasis deep learning dan LLM sering unggul dari sisi performa, tetapi tidak selalu mudah dijelaskan. Hal ini menjadi masalah penting karena rekayasa perangkat lunak membutuhkan kepercayaan, auditabilitas, dan kemampuan penelusuran keputusan. Kajian tentang reliability and explainability menegaskan bahwa evaluasi yang terlalu optimistis dapat terjadi jika benchmark tidak dirancang secara ketat, sedangkan pendekatan neuro symbolic menawarkan jalan tengah dengan menggabungkan pembelajaran statistik dan penalaran simbolik untuk meningkatkan akurasi sekaligus keterjelasan proses inferensi (Liu et al., 2024; Chen, 2025). Dengan demikian, implikasi teoretis dari hasil ini adalah perlunya pergeseran dari sekadar mengejar skor performa menuju pengembangan sistem machine learning untuk pemrograman yang juga dapat dipercaya, dapat dijelaskan, dan dapat diverifikasi.

Dari sudut kontribusi bidang, kajian ini memberikan sumbangan pada literatur dengan memperlihatkan bahwa peran algoritma machine learning dalam pemrograman bersifat multidimensi. Kontribusi tersebut mencakup percepatan penulisan kode, peningkatan kualitas pengujian, otomasi debugging, optimasi kompilasi, hingga perbaikan perangkat lunak berbasis pembelajaran. Kajian ini juga menunjukkan bahwa batas antara software engineering dan machine learning semakin kabur karena teknik pembelajaran kini tidak hanya diterapkan pada produk perangkat lunak, tetapi juga pada proses pembuatannya sendiri (Tawar, 2024; Chen et al., 2024). Secara praktis, hasil ini dapat menjadi dasar bagi pengembang, peneliti, dan organisasi untuk mengadopsi strategi integrasi machine learning yang lebih sistematis dalam toolchain pemrograman.

Untuk penelitian selanjutnya, beberapa perbaikan dapat disarankan. Pertama, diperlukan evaluasi yang lebih ketat, transparan, dan bebas bias data duplikasi agar performa model benar benar mencerminkan kemampuan generalisasi. Kedua, pengembangan model perlu diarahkan pada integrasi explainable AI, formal verification, dan symbolic reasoning agar hasil yang diperoleh lebih dapat dipercaya untuk penggunaan kritis. Ketiga, riset masa depan perlu lebih banyak menguji model pada proyek nyata, multi bahasa pemrograman, serta tugas yang menuntut reasoning bertahap dan pemahaman konteks yang lebih dalam. Keempat, pendekatan hybrid dan multi agent tampak menjanjikan karena mampu menggabungkan kelebihan beberapa teknik sekaligus, baik dalam code generation, repair, maupun optimization (Mao et al., 2025; Chen, 2025; Charalambous et al., 2024). Dengan arah tersebut, pengembangan machine learning untuk optimalisasi proses pemrograman dapat bergerak dari tahap asistif menuju tahap yang lebih andal, adaptif, dan bertanggung jawab.

SIMPULAN

Berdasarkan hasil kajian pustaka, dapat disimpulkan bahwa algoritma machine learning berperan penting dalam optimalisasi proses pemrograman melalui otomatisasi code generation, deteksi bug, pengujian, dan optimasi kode, sehingga meningkatkan efisiensi, akurasi, dan produktivitas pengembangan perangkat lunak. Temuan ini menunjukkan bahwa pemrograman telah bertransformasi menjadi proses yang adaptif dan berbasis data, sekaligus memperkuat integrasi antara kecerdasan buatan dan rekayasa perangkat lunak dalam ranah akademik maupun praktik industri. Secara sosial, hal ini mendorong perubahan peran pengembang menuju pengelola sistem cerdas. Namun, keterbatasan studi pustaka mengindikasikan perlunya validasi empiris lebih lanjut. Oleh karena itu, disarankan agar praktisi mengadopsi machine learning secara sistematis, akademisi memperdalam penelitian terutama pada aspek interpretabilitas dan keandalan, serta penelitian selanjutnya menggunakan pendekatan triangulasi dan eksplorasi implementasi nyata agar hasilnya lebih komprehensif dan aplikatif.

DAFTAR PUSTAKA

Abraham, D., & P, P. (2024). A methodological framework for descriptive phenomenological research. *Western Journal of Nursing Research*. https://doi.org/10.1177/01939459241308071

Antal, G., Vozár, R., & Ferenc, R. (2024). Toward a new era of rapid development: Assessing GPT 4 Vision's capabilities in UML based code generation. *2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*. https://doi.org/10.1145/3643795.3648391

Bandaranayake, P. (2024). Application of grounded theory methodology in library and information science research: An overview. *Sri Lanka Library Review*. https://doi.org/10.4038/sllr.v38i2.70

Belotto, M. (2018). Data analysis methods for qualitative research: Managing the challenges of coding, interrater reliability, and thematic analysis. *The Qualitative Report*. https://doi.org/10.46743/2160-3715/2018.3492

Bingham, A. (2023). From data management to actionable findings: A five-phase process of qualitative data analysis. *International Journal of Qualitative Methods*. https://doi.org/10.1177/16094069231183620

Charalambous, Y., Manino, E., & Cordeiro, L. C. (2024). Automated repair of AI code with large language models and formal verification. *arXiv*. https://doi.org/10.48550/arXiv.2405.08848

Chen, L. (2025). Hybrid neuro symbolic AI model for automated code analysis in software engineering. *Darpan International Research Analysis*. https://doi.org/10.36676/dira.v13.i4.186

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Cowan, M., Shen, H., Wang, L., Hu, Y., Ceze, L., Guestrin, C., & Krishnamurthy, A. (2018). TVM: An automated end to end optimizing compiler for deep learning. *Operating Systems Design and Implementation*. https://doi.org/10.5555/3291168.3291211

Doyle, L., McCabe, C., Keogh, B., Brady, A., & McCann, M. (2019). An overview of the qualitative descriptive design within nursing research. *Journal of Research in Nursing*. https://doi.org/10.1177/1744987119880234

Farhadi, S. (2022). A review on the impact of evolutionary optimization algorithms in enhancing learning processes. *Journal of Study and Innovation in Education and Development*. https://doi.org/10.61838/jsied.2.2.2

Fife, S., & Gossner, J. (2024). Deductive qualitative analysis: Evaluating, expanding, and refining theory. *International Journal of Qualitative Methods*. https://doi.org/10.1177/16094069241244856

Granikov, V., Hong, Q., Crist, E., & Pluye, P. (2020). Mixed methods research in library and information science: A methodological review. *Library & Information Science Research*. https://doi.org/10.1016/j.lisr.2020.101003

Hoffman, I., & Brooks, N. (2025). Automated bug detection and correction in software development using machine learning. *International Journal on Advanced Computer Theory and Engineering*. https://doi.org/10.65521/ijacte.v12i1.108

Imran, B., Wahyudi, E., Riadi, S., Muahidin, Z., Erniwati, S., & Wahyuni, W. (2025). A comparative hybrid approach for Python bug detection using syntactic features, random forest, and neural network. https://doi.org/10.21512/commit.v19i2.13183

Jawalkar, S. K. (2021). Machine learning in QA: A vision for predictive and adaptive software testing. *International Journal of Scientific Research in Engineering and Management*. https://doi.org/10.55041/ijsrem9725

Jimenez, S., Berbegal-Mirabent, J., & De La Torre, R. (2024). How do university libraries contribute to the research process?. *The Journal of Academic Librarianship*. https://doi.org/10.1016/j.acalib.2024.102930

Karthick, K. (2024). Comprehensive overview of optimization techniques in machine learning training. *Control Systems and Optimization Letters*. https://doi.org/10.59247/csol.v2i1.69

Lamouri, D. R., Aouadj, I. N., Kourta, S., & Baghdadi, R. (2025). Pearl: Automatic code optimization using deep reinforcement learning. *International Conference on Supercomputing*. https://doi.org/10.1145/3721145.3725766

Li, J., Qin, Z., Mei, Y., Cui, J., Song, Y., Chen, C., Zhang, Y., Du, L., Cheng, X., Jin, B., Ye, J., Lin, E., & Lavery, D. M. (2023). oneDNN graph compiler: A hybrid approach for high performance deep learning compilation. *IEEE/ACM International Symposium on Code Generation and Optimization*. https://doi.org/10.1109/CGO57630.2024.10444871

Liu, Y., Tantithamthavorn, C., Liu, Y., & Li, L. (2024). On the reliability and explainability of language models for program generation. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.1145/3641540

Ma, Z., Wang, H., Xing, J., Huang, S., Zheng, L., Zhang, C., Cao, H., Huang, K., Zhai, M., Tang, S., Wang, P., & Zhai, J. (2025). IntelliGen: Instruction level auto tuning for tensor program with monotonic memory optimization. *IEEE/ACM International*

Symposium on Code Generation and Optimization. https://doi.org/10.1145/3696443.3708967

Mao, K., Hu, B., Lin, R., Li, Z., Lu, G., & Zhang, Z. (2025). Blueprint2Code: A multi agent pipeline for reliable code generation via blueprint planning and repair. *Frontiers in Artificial Intelligence*. https://doi.org/10.3389/frai.2025.1660912

Mesbah, A., Rice, A., Johnston, E., Glorioso, N., & Aftandilian, E. (2019). DeepDelta: Learning to repair compilation errors. https://doi.org/10.1145/3338906.3340455

More, R., & Bradbury, J. S. (2025). Assessing data augmentation-induced bias in training and testing of machine learning models. *IEEE SANER*. https://doi.org/10.1109/SANER-C66551.2025.00015

Nguyen, T. D., Tian, H., Le, B., Thongtanunam, P., & McIntosh, S. (2025). A systematic survey on debugging techniques for machine learning systems. *arXiv*. https://doi.org/10.48550/arXiv.2503.03158

Pradel, M., & Sen, K. (2018). DeepBugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages*. https://doi.org/10.1145/3276517

Pratt, M. (2025). On the evolution of qualitative methods in organizational research. *Annual Review of Organizational Psychology and Organizational Behavior*. https://doi.org/10.1146/annurev-orgpsych-111722-032953

Tawar, D. B. (2024). The role of machine learning in software development. *International Journal of Innovative Science and Research Technology*. https://doi.org/10.38124/ijisrt/ijisrt24may2519

Togia, A., & Malliari, A. (2017). Research methods in library and information science. https://doi.org/10.5772/intechopen.68749

Vila-Henninger, L., Dupuy, C., Van Ingelgom, V., Caprioli, M., Teuber, F., Pennetreau, D., Bussi, M., & Gall, C. (2022). Abductive coding: Theory building and qualitative (re)analysis. *Sociological Methods & Research*. https://doi.org/10.1177/00491241211067508

Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier aware unified pre trained encoder decoder models for code understanding and generation. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/2021.emnlp-main.685

Zhang, Q., Fang, C., Ma, Y., Sun, W., & Chen, Z. (2023). A survey of learning based automated program repair. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.48550/arXiv.2301.03270

Zhao, Q., Wang, R., Liu, Y., Yang, H., Luan, Z., & Qian, D. (2025). Sifter: An efficient operator auto-tuner with speculative design space exploration for deep learning compiler. *IEEE Transactions on Computers*. https://doi.org/10.1109/TC.2024.3441820