Department of Digital Business
Journal of Artificial Intelligence and Digital Business (RIGGS)
Homepage: https://journal.ilmudata.co.id/index.php/RIGGS
Vol. 4 No. 4 (2026) pp: 10256-10265
P-ISSN: 2963-9298, e-ISSN: 2963-914X

Peran Machine Learning dalam Predictive Analytics untuk Software Engineering: Tinjauan Integratif

Ahmad Al Kaafi
Universitas Bina Sarana Informatika
ahmad_akf@bsi.ac.id

Abstrak

Industri pengembangan perangkat lunak menghadapi tantangan yang semakin kompleks dalam memperkirakan waktu pengembangan, mengidentifikasi bug sejak dini, serta mengelola kualitas kode secara konsisten. Machine learning telah muncul sebagai solusi transformatif dalam ranah predictive analytics untuk rekayasa perangkat lunak, karena mampu menghasilkan prediksi yang lebih akurat sekaligus mendukung pengambilan keputusan berbasis data. Tinjauan integratif ini menganalisis secara komprehensif peran machine learning dalam berbagai aspek predictive analytics untuk software engineering, meliputi estimasi effort pengembangan, prediksi defect, analisis code smell, serta forecasting kebutuhan maintenance.Metode tinjauan sistematis diterapkan terhadap 87 artikel ilmiah yang dipublikasikan pada periode 2020–2025 dan diperoleh dari basis data bereputasi seperti IEEE Xplore, ACM Digital Library, serta ScienceDirect. Hasil analisis menunjukkan bahwa algoritma ensemble methods seperti Random Forest dan Gradient Boosting mencapai akurasi tertinggi (85–92%) dalam prediksi defect. Sementara itu, model deep learning menunjukkan performa yang lebih unggul dalam estimasi effort proyek dengan nilai RMSE < 15%. Neural networks dan transformer-based models juga terbukti efektif dalam menganalisis kualitas kode, dengan precision mencapai 89%. Tantangan utama yang teridentifikasi meliputi ketersediaan dataset berkualitas, isu interpretabilitas model, serta kemampuan generalisasi lintas proyek. Temuan ini memberikan roadmap komprehensif bagi praktisi software engineering untuk mengintegrasikan machine learning dalam seluruh siklus pengembangan perangkat lunak, disertai rekomendasi spesifik terkait pemilihan algoritma yang selaras dengan konteks proyek dan karakteristik data yang tersedia.

Kata kunci: Machine Learning, Predictive Analytics, Software Engineering, Estimasi Effort, Prediksi Defect, Code Quality

1. Pendahuluan

1.1 Latar Belakang

Industri perangkat lunak global telah mengalami pertumbuhan eksponensial dalam dekade terakhir, dengan nilai pasar yang diproyeksikan mencapai 1,3 triliun dolar AS pada tahun 2025. Kompleksitas proyek perangkat lunak modern yang melibatkan jutaan baris kode, tim terdistribusi secara global, dan metodologi agile yang iteratif menciptakan tantangan signifikan dalam manajemen proyek dan quality assurance. Menurut laporan Standish Group Chaos Report 2023, hanya 31% proyek perangkat lunak yang selesai tepat waktu dan sesuai anggaran, sementara 19% mengalami kegagalan total.

Kesenjangan antara estimasi awal dan realisasi aktual proyek perangkat lunak telah menjadi permasalahan kronis dalam industri. Penelitian oleh Software Engineering Institute menunjukkan bahwa rata-rata overrun waktu dalam proyek perangkat lunak mencapai 33%, dengan biaya aktual melampaui estimasi sebesar 43%. Ketidakakuratan prediksi ini berdampak pada alokasi resource yang tidak efisien, keterlambatan time-to-market, dan penurunan kepuasan klien.

Machine learning, sebagai subset dari artificial intelligence yang memungkinkan sistem belajar dari data tanpa pemrograman eksplisit, menawarkan paradigma baru dalam predictive analytics untuk software engineering. Berbeda dengan metode tradisional yang bergantung pada expert judgement dan rumus matematis sederhana seperti COCOMO (Constructive Cost Model), pendekatan machine learning dapat mengidentifikasi pola kompleks dan non-linear dalam data historis proyek, karakteristik kode, dan metrik pengembangan.

Sebagaimana dilaporkan oleh *TechCrunch* pada Januari 2024, perusahaan teknologi terkemuka seperti Google, Microsoft, dan Meta telah mengintegrasikan machine learning ke dalam pipeline CI/CD (Continuous Integration/Continuous Deployment) mereka, menghasilkan reduksi bug production sebesar 40% dan peningkatan developer productivity sebesar 25%.

Tabel 1 menyajikan perbandingan pendekatan tradisional dan machine learning dalam berbagai aspek predictive analytics untuk software engineering, yang mengindikasikan keunggulan signifikan pendekatan machine learning dalam hal akurasi dan adaptabilitas.

Tabel 1. Perbandingan Pendekatan Tradisional vs Machine Learning dalam Predictive Analytics Software Engineering

Aspek	Pendekatan Tradisional	Machine Learning	Peningkatan (%)
Akurasi Estimasi Effort	± 35-50%	± 12-18%	60-70%
Deteksi Bug/Defect	60-70%	85-92%	25-32%
Prediksi Maintenance	Reaktif	Proaktif 80%+	N/A
Waktu Analisis	Manual (hari)	Otomatis (menit)	95%+
Adaptasi ke Proyek Baru	Rendah	Tinggi	N/A

Sumber: Kompilasi dari studi Chen dkk. (2023), Kumar & Singh (2024), dan Zhang dkk. (2024)

1.2 Rumusan Masalah

Meskipun potensi machine learning dalam software engineering telah diakui secara luas, implementasi praktis masih menghadapi berbagai tantangan. Berdasarkan analisis preliminer terhadap literatur terkini, penelitian ini merumuskan pertanyaan-pertanyaan berikut:

- Bagaimana performa berbagai algoritma machine learning dalam konteks predictive analytics untuk software engineering, khususnya dalam estimasi effort, prediksi defect, dan analisis code quality?
- Apa saja tantangan utama dalam implementasi machine learning untuk predictive analytics dalam software engineering, dan bagaimana strategi untuk mengatasinya?
- Bagaimana tren dan arah pengembangan masa depan machine learning dalam domain software engineering?
- Apa rekomendasi praktis bagi praktisi software engineering dalam mengadopsi machine learning untuk predictive analytics?

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk:

- Menganalisis secara komprehensif aplikasi machine learning dalam berbagai domain predictive analytics untuk software engineering
- Mengevaluasi performa berbagai algoritma machine learning berdasarkan metrik akurasi, presisi, recall, dan F1-score dalam konteks software engineering
- Mengidentifikasi best practices dan lessons learned dari implementasi machine learning dalam industri perangkat lunak
- Menyusun framework rekomendasi untuk pemilihan dan implementasi algoritma machine learning berdasarkan karakteristik proyek dan ketersediaan data

1.4 Manfaat Penelitian

Tinjauan integratif ini memberikan kontribusi teoretis dalam pengembangan body of knowledge tentang aplikasi machine learning dalam software engineering. Secara praktis, hasil penelitian ini dapat digunakan oleh praktisi software engineering, project managers, dan quality assurance teams sebagai panduan dalam mengimplementasikan predictive analytics berbasis machine learning untuk meningkatkan efisiensi pengembangan, kualitas produk, dan keberhasilan proyek perangkat lunak.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

2. Tinjauan Pustaka

2.1 Machine Learning: Konsep Fundamental

Machine learning merupakan cabang artificial intelligence yang memungkinkan komputer belajar dari data dan pengalaman tanpa diprogram secara eksplisit. Dalam konteks software engineering, machine learning menggunakan data historis proyek—seperti metrik kode, bug reports, commit histories, dan project documentation—untuk membangun model prediktif yang dapat mengidentifikasi pola, membuat prediksi, dan memberikan insights actionable.

Terdapat tiga paradigma utama dalam machine learning yang relevan untuk software engineering: supervised learning, unsupervised learning, dan reinforcement learning. Supervised learning, yang paling umum digunakan dalam predictive analytics, melatih model menggunakan data berlabel untuk memprediksi outcome spesifik seperti effort estimation atau bug classification. Unsupervised learning digunakan untuk menemukan pola tersembunyi dalam kode dan mengidentifikasi anomali tanpa label eksplisit. Reinforcement learning, meskipun masih dalam tahap eksplorasi, menunjukkan potensi dalam automated testing dan continuous optimization.

2.2 Predictive Analytics dalam Software Engineering

Predictive analytics dalam software engineering merujuk pada penggunaan data historis, statistical algorithms, dan machine learning techniques untuk mengidentifikasi kemungkinan outcome masa depan dalam siklus pengembangan perangkat lunak. Berbeda dengan descriptive analytics yang menjelaskan apa yang terjadi dan diagnostic analytics yang menjelaskan mengapa sesuatu terjadi, predictive analytics berfokus pada apa yang akan terjadi dan bagaimana meresponsnya secara proaktif.

Penelitian oleh Malhotra & Khanna (2020) mengidentifikasi empat domain utama penerapan predictive analytics dalam software engineering: effort estimation, defect prediction, software maintenance prediction, dan developer productivity forecasting. Setiap domain memiliki karakteristik data yang berbeda dan memerlukan pendekatan machine learning yang disesuaikan.

2.3 Algoritma Machine Learning untuk Software Engineering

2.3.1 Supervised Learning Algorithms

Algoritma supervised learning yang paling sering diterapkan dalam software engineering mencakup:

Linear Regression dan Multiple Regression: Digunakan untuk estimasi effort dan prediksi waktu pengembangan. Meskipun sederhana, algoritma ini memberikan baseline yang berguna dan interpretabilitas tinggi. Penelitian oleh Azzeh dkk. (2021) menunjukkan bahwa multiple regression dengan feature engineering yang tepat dapat mencapai akurasi 75-80% dalam estimasi effort untuk proyek skala kecil hingga menengah.

Decision Trees dan Random Forest: Efektif untuk klasifikasi bug severity dan prediksi defect-prone modules. Random Forest, sebagai ensemble method yang mengkombinasikan multiple decision trees, menunjukkan robustness tinggi terhadap overfitting. Kumar & Singh (2024) melaporkan akurasi 88% dalam prediksi defect menggunakan Random Forest pada dataset NASA MDP.

Support Vector Machines (SVM): Powerful untuk klasifikasi dalam high-dimensional space, sering digunakan untuk analisis code smell dan identification of design patterns. Penelitian oleh Chen dkk. (2023) mendemonstrasikan bahwa SVM dengan kernel RBF mencapai precision 86% dalam mengidentifikasi code smells pada proyek open-source.

Gradient Boosting Methods (XGBoost, LightGBM): State-of-the-art untuk berbagai task dalam software engineering. Zhang dkk. (2024) melaporkan bahwa XGBoost mengungguli algoritma lain dalam prediksi build failure dengan AUC-ROC 0,93, sementara LightGBM menunjukkan training time yang lebih cepat untuk dataset besar.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

2.3.2 Deep Learning Approaches

Deep learning, khususnya neural networks dengan multiple hidden layers, telah menunjukkan hasil promising dalam software engineering tasks yang melibatkan data kompleks dan high-dimensional:

Recurrent Neural Networks (RNN) dan Long Short-Term Memory (LSTM): Efektif untuk sequential data seperti commit histories dan code evolution. Penelitian oleh Li dkk. (2023) menggunakan LSTM untuk memprediksi bug introduction berdasarkan sequence of code changes, mencapai F1-score 0,82.

Convolutional Neural Networks (CNN): Meskipun originally designed untuk image processing, CNN terbukti efektif untuk code analysis dengan memperlakukan source code sebagai 2D matrix. Wang dkk. (2022) mendemonstrasikan penggunaan CNN untuk vulnerability detection dengan accuracy 91%.

Transformer Models: Models seperti BERT dan GPT yang pre-trained pada large code corpora menunjukkan kemampuan luar biasa dalam code understanding dan generation. CodeBERT yang dikembangkan oleh Microsoft mencapai state-of-the-art performance dalam berbagai code-related tasks. Ahmed & Rahman (2024) melaporkan bahwa fine-tuned CodeBERT mencapai precision 89% dalam code review automation.

2.4 Aplikasi Spesifik dalam Software Engineering

2.4.1 Effort Estimation

Estimasi effort merupakan salah satu aplikasi paling kritis dari machine learning dalam software engineering. Metode tradisional seperti COCOMO, Function Point Analysis, dan Use Case Points memiliki keterbatasan dalam menangkap kompleksitas proyek modern yang melibatkan agile methodologies, distributed teams, dan emerging technologies.

Penelitian terkini menunjukkan bahwa ensemble methods menghasilkan estimasi paling akurat. Studi oleh Jørgensen & Grimstad (2023) membandingkan 15 algoritma machine learning pada dataset ISBSG (International Software Benchmarking Standards Group) yang mencakup 5.000+ proyek, menemukan bahwa Random Forest dan Gradient Boosting mencapai MMRE (Magnitude of Relative Error) terendah sebesar 0,24 dan 0,22 respectively, jauh lebih baik dibandingkan COCOMO II yang mencapai MMRE 0,51.

2.4.2 Defect Prediction

Defect prediction bertujuan mengidentifikasi modules atau components yang cenderung mengandung bugs sebelum testing atau deployment. Pendekatan ini memungkinkan allocation of testing resources yang lebih efisien dan targeted code review.

Meta-analisis oleh Kumar & Singh (2024) terhadap 87 studi defect prediction mengungkapkan bahwa ensemble methods konsisten mengungguli single classifiers. Random Forest mencapai median AUC 0,87, diikuti oleh XGBoost dengan AUC 0,85, sementara logistic regression hanya mencapai AUC 0,72. Penelitian juga menemukan bahwa feature engineering memainkan peran krusial—penambahan process metrics (seperti number of developers, commit frequency) ke code metrics meningkatkan AUC rata-rata sebesar 0,08.

2.4.3 Code Quality Analysis

Machine learning dapat mengautomasi deteksi code smells, anti-patterns, dan potential vulnerabilities. Pendekatan ini mengatasi keterbatasan static analysis tools yang berbasis rules dan sering menghasilkan false positives tinggi.

Penelitian oleh Chen dkk. (2023) mengembangkan deep learning model untuk mengklasifikasikan 23 types of code smells menggunakan Abstract Syntax Tree (AST) representations. Model CNN-LSTM mereka mencapai precision 0,86 dan recall 0,83, dengan false positive rate 30% lebih rendah dibandingkan traditional static analysis tools seperti PMD dan SonarQube.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

2.4.4 Software Maintenance Prediction

Prediksi maintenance effort dan identification of change-prone components membantu tim development dalam planning refactoring activities dan technical debt management. Machine learning models dapat memprediksi which parts of codebase yang akan require the most maintenance effort berdasarkan historical change patterns.

Studi longitudinal oleh Ahmed & Rahman (2024) yang melacak 50 open-source projects selama 5 tahun menunjukkan bahwa Random Forest model yang trained pada code and process metrics dapat memprediksi high-maintenance modules dengan precision 0,78 dan recall 0,81. Model ini mengidentifikasi bahwa code complexity metrics (cyclomatic complexity, nesting depth) dan change metrics (number of modifications, number of bug *fixes) merupakan strongest predictors.*

2.5 Tantangan dan Keterbatasan

Meskipun machine learning menunjukkan hasil promising, beberapa tantangan signifikan perlu diatasi:

Data Quality dan Availability: Machine learning models require large amounts of high-quality labeled data. Banyak organisasi tidak memiliki historical data yang cukup atau data quality yang poor (missing values, inconsistent labeling). Studi oleh Malhotra & Khanna (2020) menemukan bahwa 68% organizations struggle dengan data collection dan cleaning untuk machine learning initiatives.

Class Imbalance: Dalam defect prediction, bug-containing modules typically represent minority class (10-20% of modules). Standard machine learning algorithms tend to bias towards majority class. Teknik seperti SMOTE (Synthetic Minority Oversampling Technique) dan cost-sensitive learning diperlukan untuk addressing imbalance.

Model Interpretability: Complex models seperti deep neural networks often act as black boxes. Developers need to understand why a model makes certain predictions untuk trust dan act on them. Explainable AI (XAI) techniques seperti LIME (Local Interpretable Model-agnostic Explanations) dan SHAP (SHapley Additive exPlanations) gaining traction untuk addressing interpretability.

Cross-Project Generalization: Models trained on one project often perform poorly on different projects due to variations dalam programming languages, development practices, dan team characteristics. Transfer learning approaches sedang explored untuk improving cross-project prediction.

3. Metode Penelitian

3.1 Desain Penelitian

Penelitian ini menggunakan pendekatan tinjauan integratif (integrative review) yang menggabungkan hasil dari multiple research methodologies untuk menghasilkan comprehensive understanding tentang peran machine learning dalam predictive analytics untuk software engineering. Berbeda dengan systematic literature review yang strictly quantitative, integrative review memungkinkan sintesis dari diverse sources termasuk empirical studies, case studies, dan theoretical papers.

3.2 Strategi Pencarian Literatur

Pencarian literatur dilakukan pada tiga database akademik utama: IEEE Xplore Digital Library, ACM Digital Library, dan ScienceDirect. Search string yang digunakan: ("machine learning" OR "deep learning" OR "artificial intelligence") AND ("software engineering" OR "software development") AND ("prediction" OR "estimation" OR "forecasting" OR "analytics") AND ("defect" OR "bug" OR "effort" OR "maintenance" OR "quality").

Kriteria inklusi: (1) Artikel dipublikasikan dalam periode 2020-2025, (2) Peer-reviewed journals atau conference proceedings, (3) Fokus pada aplikasi machine learning untuk predictive analytics dalam software engineering, (4) Menyajikan hasil empiris atau case studies. Kriteria eksklusi: (1) Non-English articles, (2) Gray literature, (3) Duplicate publications.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

Pencarian awal menghasilkan 342 artikel. Setelah screening berdasarkan title dan abstract, 156 artikel dipilih untuk full-text review. Quality assessment menggunakan adapted MMAT (Mixed Methods Appraisal Tool) menghasilkan 87 artikel yang memenuhi threshold quality score ≥ 75% untuk dianalisis secara mendalam.

3.3 Analisis Data

Data extraction dilakukan menggunakan structured form yang mencakup: study characteristics (author, year, publication venue), research methodology, machine learning algorithms used, application domain, dataset characteristics, performance metrics, dan key findings. Thematic analysis digunakan untuk mengidentifikasi patterns dan themes across studies.

Sintesis hasil dilakukan melalui narrative synthesis yang mengorganisir findings berdasarkan application domains (effort estimation, defect prediction, code quality, maintenance). Perbandingan algorithm performance dilakukan dengan menormalisasi reported metrics dan calculating weighted averages based on sample sizes.

4. Hasil dan Pembahasan

4.1 Karakteristik Studi yang Dianalisis

Dari 87 artikel yang dianalisis, 42% merupakan empirical studies menggunakan public datasets, 31% case studies di organisasi spesifik, 18% comparative studies antar algoritma, dan 9% theoretical atau conceptual papers. Distribusi berdasarkan application domain: defect prediction (38%), effort estimation (27%), code quality analysis (21%), dan maintenance prediction (14%).

Dataset yang paling sering digunakan termasuk NASA MDP (32 studies), PROMISE Repository (28 studies), dan GitHub repositories (41 studies). Terdapat trend increasing penggunaan real-world industrial datasets, dengan 23% studies melaporkan collaboration dengan companies untuk accessing proprietary data.

4.2 Performa Algoritma Machine Learning

4.2.1 Effort Estimation

Analisis terhadap 23 studi effort estimation mengungkapkan bahwa ensemble methods secara konsisten mengungguli individual algorithms. Tabel 2 menyajikan performance comparison dari major algorithms:

Tabel 2. Performa Algoritma dalam Effort Estimation

Algoritma	MMRE (rata-rata)	PRED(25) %	Jumlah Studi	Dataset Umum
Random Forest	0,22	68%	12	ISBSG, COCOMO
Gradient Boosting	0,21	71%	8	ISBSG, Proprietary
Neural Networks	0,26	64%	7	ISBSG, GitHub
SVM	0,31	58%	6	COCOMO, PROMISE
Linear Regression	0,45	42%	9	ISBSG, Historical

MMRE = Magnitude of Mean Relative Error; PRED(25) = Percentage of predictions within 25% of actual
Sumber: Sintesis dari 23 studi effort estimation

Random Forest dan Gradient Boosting menunjukkan performance superior dengan MMRE di bawah 0,25, yang dianggap acceptable dalam industry standards. Deep learning approaches, meskipun menunjukkan promise untuk extremely large projects dengan complex dependencies, memerlukan substantially more training data dan computational resources.

Feature importance analysis across studies mengidentifikasi bahwa size metrics (lines of code, function points), complexity metrics (cyclomatic complexity), dan team experience merupakan strongest predictors untuk effort estimation. Penggunaan ensemble feature selection methods meningkatkan prediction accuracy rata-rata sebesar 12%.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

4.2.2 Defect Prediction

Defect prediction merupakan application domain dengan most extensive research coverage. Meta-analisis dari 33 studi menghasilkan insights berikut:

Ensemble methods—khususnya Random Forest dan XGBoost—consistently achieve highest performance dengan median AUC 0,87 dan 0,85 respectively. Deep learning models menunjukkan competitive performance (AUC 0,84) namun dengan substantially higher computational cost. Cost-sensitive learning techniques meningkatkan recall untuk minority class (bug-containing modules) rata-rata sebesar 15% dengan trade-off minimal dalam precision.

Analysis of false positives dan false negatives mengungkapkan bahwa imbalanced learning techniques seperti SMOTE dan ADASYN significantly reduce false positives (rata-rata reduction 28%) without substantially increasing false negatives. Hybrid approaches yang mengkombinasikan oversampling dengan ensemble learning menunjukkan best overall performance.

Process metrics (developer experience, commit frequency, file age) ketika dikombinasikan dengan code metrics meningkatkan prediction accuracy rata-rata sebesar 9%. Temporal validation—testing model pada future releases—menunjukkan performance degradation rata-rata 7% dibandingkan cross-validation, highlighting importance of model retraining.

4.2.3 Code Quality Analysis

Machine learning untuk code quality analysis mencakup detection of code smells, identification of design patterns, dan vulnerability detection. Analisis dari 18 studi mengungkapkan:

Deep learning approaches, khususnya CNN dan LSTM models yang operate pada AST (Abstract Syntax Tree) atau token representations, achieve highest precision (0,86-0,89) dalam code smell detection. Traditional machine learning algorithms seperti SVM dan Random Forest menunjukkan competitive performance (precision 0,82-0,85) dengan advantage of better interpretability dan lower computational requirements.

Transformer-based models seperti CodeBERT demonstrate exceptional performance dalam code understanding tasks, achieving F1-score 0,88 dalam automated code review suggestions. Fine-tuning pre-trained models pada domain-specific codebases improves performance rata-rata sebesar 11% dibandingkan training from scratch.

Integration of static analysis tools dengan machine learning reduces false positive rates rata-rata sebesar 35%. Hybrid approaches yang menggunakan rule-based systems untuk filtering followed by machine learning classification menunjukkan best trade-off antara precision dan recall.

4.2.4 Maintenance Prediction

Prediction of maintenance effort dan identification of change-prone components assist dalam technical debt management dan refactoring prioritization. Analysis of 12 studi maintenance prediction mengidentifikasi:

Random Forest models trained pada combination of code metrics, change metrics, dan developer metrics achieve best performance dengan precision 0,78 dan recall 0,81 dalam predicting high-maintenance modules. Temporal features—capturing evolution patterns over time—improve prediction accuracy rata-rata sebesar 14%.

Survival analysis approaches combined dengan machine learning show promise untuk predicting time-to-next-bug dan time-to-refactoring. Cox proportional hazards models with machine learning-based feature selection outperform traditional approaches dengan concordance index meningkat dari 0,68 menjadi 0,76.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

4.3 Best Practices dan Lessons Learned

4.3.1 Data Preparation dan Feature Engineering

Quality of training data merupakan determining factor untuk model performance. Best practices yang teridentifikasi across studies:

- Comprehensive data cleaning: Removal of duplicates, handling of missing values (median/mode imputation untuk numerical/categorical features), dan outlier detection using IQR atau isolation forests
- Feature engineering: Creation of interaction terms, polynomial features, dan domain-specific metrics. Studies yang invested dalam thoughtful feature engineering achieved 15-20% better performance
- Feature selection: Application of multiple selection methods (correlation analysis, mutual information, recursive feature elimination) dengan consensus approach
- Data balancing: Use of appropriate techniques (SMOTE, ADASYN, cost-sensitive learning) untuk addressing class imbalance, with careful validation untuk avoiding overfitting

4.3.2 Model Selection dan Validation

Selection of appropriate algorithm depends on context, data characteristics, dan requirements untuk interpretability vs accuracy:

- Untuk effort estimation dengan limited data: Random Forest atau Gradient Boosting with cross-validation
- Untuk defect prediction dengan imbalanced data: Ensemble methods dengan cost-sensitive learning atau oversampling
- Untuk code quality analysis requiring interpretability: SVM atau Random Forest dengan feature importance analysis
- Untuk complex patterns dengan large datasets: Deep learning approaches dengan appropriate regularization

Validation strategies yang robust termasuk temporal validation (testing pada future data), cross-project validation (testing pada different projects), dan stratified k-fold cross-validation dengan multiple random seeds untuk ensuring stability.

4.3.3 Implementation Considerations

Successful deployment of machine learning dalam software engineering requires addressing practical considerations:

- Integration dengan existing tools: Machine learning models perlu seamlessly integrate dengan CI/CD pipelines, issue tracking systems, dan code review platforms
- Model monitoring dan retraining: Performance degradation over time necessitates regular monitoring dan retraining schedules. Studies recommend quarterly retraining untuk defect prediction models
- Explainability untuk adoption: Developers are more likely to trust dan act on predictions yang can be explained. Implementation of SHAP values atau LIME improves model adoption
- Incremental implementation: Pilot studies dalam specific domains sebelum organization-wide rollout reduces risks dan enables learning

4.4 Tantangan dan Solusi

4.4.1 Data Availability dan Quality

Tantangan: 68% of organizations lack sufficient historical data atau struggle dengan data quality issues. Proprietary nature of software projects limits access to diverse datasets.

Solusi yang emerging: Transfer learning approaches allowing models trained pada public datasets to be fine-tuned dengan limited private data. Synthetic data generation using GANs (Generative Adversarial Networks) showing promise untuk augmenting limited datasets. Federated learning enabling collaborative model training without sharing sensitive data.

DOI: https://doi.org/10.31004/riggs.v4i4.4997

4.4.2 Model Interpretability

Tantangan: Black-box nature of complex models hampers developer trust dan adoption. Regulatory requirements dalam certain domains demand explainability.

Solusi: Integration of XAI techniques (SHAP, LIME) dalam deployment pipelines. Development of inherently interpretable models using decision rules atau linear models dengan feature engineering. Hybrid approaches combining interpretable base models dengan ensemble methods.

4.4.3 Cross-Project Generalization

Tantangan: Models trained pada one project often perform poorly pada different projects due to contextual differences. Average performance degradation of 15-25% observed dalam cross-project scenarios.

Solusi: Transfer learning approaches showing 40% reduction dalam performance gap. Domain adaptation techniques adjusting models untuk target project characteristics. Ensemble of models from multiple source projects improving robustness.

4.5 Tren dan Arah Masa Depan

4.5.1 Large Language Models untuk Code Understanding

Emergence of large pre-trained models seperti GPT-4 Code, CodeBERT, dan GraphCodeBERT revolutionizing code understanding. These models capture semantic meaning dan contextual relationships yang difficult untuk traditional approaches. Future research exploring application dalam automated code review, bug localization, dan intelligent code completion.

4.5.2 Automated Machine Learning (AutoML)

AutoML platforms demokratizing machine learning dengan automating algorithm selection, hyperparameter tuning, dan feature engineering. Tools seperti AutoGluon dan H2O.ai enabling software engineers without deep ML expertise untuk building effective models. Future integration dengan software development IDEs predicted.

4.5.3 Continuous Learning dan Adaptive Models

Current models typically static setelah deployment. Emerging approaches enabling continuous learning dari new data dengan online learning algorithms. Adaptive models yang automatically adjust to changing codebases dan development practices showing 20-30% better long-term performance.

4.5.4 Multimodal Learning

Integration of multiple data sources—code, documentation, commit messages, issue discussions—through multimodal learning approaches. Early studies showing 15-25% improvement dibandingkan code-only approaches. Graph neural networks capturing structural relationships dalam codebases showing particular promise.

5. Kesimpulan

Tinjauan integratif ini menganalisis 87 studi tentang aplikasi machine learning dalam predictive analytics untuk software engineering, menghasilkan insights komprehensif tentang state-of-the-art, best practices, dan future directions. Temuan utama meliputi: Pertama, ensemble methods—khususnya Random Forest dan Gradient Boosting—consistently mengungguli algoritma lain across multiple domains, achieving MMRE 0,21-0,22 dalam effort estimation dan AUC 0,85-0,87 dalam defect prediction. Deep learning approaches menunjukkan competitive atau superior performance untuk complex tasks dengan large datasets, namun memerlukan substantially more data dan computational resources. Kedua, feature engineering dan appropriate data preparation merupakan critical success factors, often contributing more untuk model performance dibandingkan algorithm selection alone. Studies yang invested dalam domain-specific feature engineering achieved 15-20% better performance. Ketiga, practical challenges termasuk data availability, model interpretability, dan cross-project generalization require thoughtful solutions. Emerging approaches seperti transfer learning, explainable AI techniques, dan federated learning showing promise dalam addressing these challenges. Keempat, integration

DOI: https://doi.org/10.31004/riggs.v4i4.4997

of machine learning dalam software engineering workflows requires careful consideration of practical factors including tool integration, model monitoring, explainability, dan incremental adoption strategies. Penelitian ini memberikan kontribusi teoretis dengan menyajikan comprehensive synthesis dari diverse research pada machine learning dalam software engineering, identifying patterns dan trends yang tidak apparent dari individual studies. Framework untuk algorithm selection berdasarkan project characteristics dan data availability provides theoretical foundation untuk decision-making. Secara praktis, penelitian ini menyediakan actionable guidance untuk practitioners dalam selecting, implementing, dan deploying machine learning solutions. Best practices dan lessons learned dari successful implementations reduce barriers untuk adoption dan accelerate time-to-value. Penelitian ini memiliki beberapa keterbatasan. Pertama, bias publikasi mungkin ada karena studies dengan negative results cenderung under-represented dalam literature. Kedua, focus pada English-language publications mungkin exclude relevant research dalam bahasa lain. Ketiga, rapid evolution dalam field means beberapa findings mungkin become outdated quickly.

Referensi

1. Ahmed, T., & Rahman, A. (2024). Deep learning approaches for software maintenance prediction: A comprehensive study. IEEE Transactions on Software Engineering, 50(3), 412-429. https://doi.org/10.1109/TSE.2024.3156789
2. Azzeh, M., Elsheikh, Y. M., & Banitaan, S. (2021). Multi-criteria decision making for software effort estimation. Information and Software Technology, 138, 106589. https://doi.org/10.1016/j.infsof.2021.106589
3. Chen, X., Zhang, Y., Wang, H., & Liu, M. (2023). CNN-LSTM hybrid model for code smell detection using abstract syntax trees. Journal of Systems and Software, 195, 111534. https://doi.org/10.1016/j.jss.2022.111534
4. Goel, L., Sharma, M., Khatri, S. K., & Damodaran, D. (2024). Explainable AI for software defect prediction: A SHAP-based approach. Expert Systems with Applications, 238, 121842. https://doi.org/10.1016/j.eswa.2023.121842
5. Hassan, A. E., Xie, T., & Mockus, A. (2023). Predictive analytics in software engineering: A systematic mapping study. ACM Computing Surveys, 55(8), 1-38. https://doi.org/10.1145/3571788
6. Jiarpakdee, J., Tantithamthavorn, C., & Treude, C. (2021). The impact of automated feature engineering on machine learning-based software defect prediction. Empirical Software Engineering, 26(4), 78. https://doi.org/10.1007/s10664-021-09969-w
7. Jørgensen, M., & Grimstad, S. (2023). Ensemble methods for software development effort estimation: A comparative study. Information and Software Technology, 162, 107279. https://doi.org/10.1016/j.infsof.2023.107279
8. Kumar, R., & Singh, A. K. (2024). Random forest and XGBoost for software defect prediction: A meta-analysis of 87 empirical studies. IEEE Access, 12, 15234-15251. https://doi.org/10.1109/ACCESS.2024.3358912
9. Li, Z., Zhao, Y., Liu, Q., & Chen, L. (2023). LSTM-based bug introduction prediction through code change sequences. Automated Software Engineering, 30(1), 12. https://doi.org/10.1007/s10515-023-00378-4
10. Liu, J., Zhou, Y., Yang, Y., Lu, H., & Xu, B. (2022). Code vulnerability detection using graph neural networks. ACM Transactions on Software Engineering and Methodology, 31(4), 1-31. https://doi.org/10.1145/3505247
11. Malhotra, R., & Khanna, M. (2020). An empirical study for software change prediction using machine learning techniques. Soft Computing, 24(21), 16611-16644. https://doi.org/10.1007/s00500-020-04968-z
12. Majd, A., Vahidi-Asl, M., Khalilian, A., Pourreza, H., & Haghighi, H. (2024). Transfer learning for cross-project software defect prediction: A systematic review. Journal of Systems and Software, 207, 111871. https://doi.org/10.1016/j.jss.2023.111871
13. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for software effort estimation: A systematic review. Artificial Intelligence Review, 54(8), 5613-5668. https://doi.org/10.1007/s10462-021-10026-1
14. Rajbahadur, G. K., Wang, S., Kamei, Y., & Hassan, A. E. (2023). The impact of using regression models to build defect prediction models. IEEE Transactions on Software Engineering, 49(4), 2094-2113. https://doi.org/10.1109/TSE.2022.3201162
15. Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., & Grundy, J. (2021). Neural network-based detection of self-admitted technical debt: From performance to explainability. ACM Transactions on Software Engineering and Methodology, 30(3), 1-45. https://doi.org/10.1145/3429444
16. Soltani, M., Hermans, F., & Bäck, T. (2024). Automated machine learning for software engineering: Opportunities and challenges. IEEE Software, 41(2), 58-66. https://doi.org/10.1109/MS.2023.3321456
17. Tian, Y., Wijedasa, D., Lo, D., & Le Goues, C. (2023). Learning to predict build failures in continuous integration: An empirical study. Journal of Systems and Software, 199, 111632. https://doi.org/10.1016/j.jss.2023.111632
18. Wang, S., Liu, T., Tan, L., Qiu, J., & Tan, H. (2022). Automatically learning semantic features for vulnerability prediction. IEEE Transactions on Software Engineering, 48(11), 4441-4456. https://doi.org/10.1109/TSE.2021.3118682
19. Xu, Z., Li, S., Tang, J., Luo, X., Zhang, T., Liu, Z., & Yang, Y. (2024). Cross-project defect prediction via feature selection and transfer learning. Information and Software Technology, 165, 107334. https://doi.org/10.1016/j.infsof.2023.107334
20. Yang, X., Lo, D., Xia, X., Zhang, Y., & Sun, J. (2023). Deep learning for just-in-time defect prediction. IEEE Transactions on Software Engineering, 49(5), 2712-2730. https://doi.org/10.1109/TSE.2022.3191168
21. Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y. (2024). Towards building a universal defect prediction model with deep learning. Empirical Software Engineering, 29(2), 45. https://doi.org/10.1007/s10664-023-10421-w
22. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K. J., Ajagbe, M. A., Chioasca, E. V., & Batista-Navarro, R. T. (2021). Natural language processing for requirements engineering: A systematic mapping study. ACM Computing Surveys, 54(3), 1-41. https://doi.org/10.1145/3444689
23. Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., & Xu, B. (2023). How far we have progressed in the journey? An examination of cross-project defect prediction. ACM Transactions on Software Engineering and Methodology, 32(4), 1-51. https://doi.org/10.1145/3576039
24. Zhu, K., Zhang, N., Ying, S., & Wang, X. (2024). CodeBERT fine-tuning for automated code review: An industrial case study. IEEE Software, 41(3), 71-80. https://doi.org/10.1109/MS.2024.3367891

DOI: https://doi.org/10.31004/riggs.v4i4.4997