PREDIKSI CERDAS SOFTWARE DEFECTS PADA PROYEK PYTHON MENGGUNAKAN STATIC ANALYSIS DAN MACHINE LEARNING

Bagas Firmansyah[1)], Muhammad Rigi Yuda Syahrial[2)], Siti Nur Kamila[3)], Rina Kurniawati[4)], Bradley Yaman[5)], dan Gina Purnama Insany[6)]
[1), 2), 3), 4), 5), 6)]Fakultas Teknik Komputer dan Desain, Universitas Nusa Putra, Sukabumi, Indonesia
e-mail: bagas.firmansyah_ti23@nusaputra.ac.id[1)], rigi.yuda_ti23@nusaputra.ac.id[2)], siti.nur_ti23@nusaputra.ac.id[3)], rina.kurniawati_ti23@nusaputra.ac.id[4)],bradley.yaman_ti23@nusaputra.ac.id[5)], gina.purnama@nusaputra.ac.id[)]

* Korespondensi: e-mail: bagas.firmansyah_ti23@nusaputra.ac.id

ABSTRAK

Keandalan perangkat lunak merupakan aspek krusial dalam rekayasa perangkat lunak modern, seiring meningkatnya penggunaan bahasa Python dalam pengembangan sistem open-source maupun enterprise. Deteksi dini cacat perangkat lunak sebelum proses eksekusi menjadi penting untuk menjaga kualitas sistem serta menekan biaya pemeliharaan. Namun, pendekatan tradisional seperti tinjauan kode manual dan analisis statis berbasis aturan sering mengalami keterbatasan dalam menangkap cacat yang bersifat kompleks dan kontekstual, khususnya pada proyek berskala besar, serta cenderung menghasilkan tingkat false positive yang tinggi. Penelitian ini mengusulkan sebuah intelligent framework yang mengintegrasikan static code analysis dengan pendekatan machine learning untuk memprediksi kecenderungan perbaikan bug pada proyek Python. Dataset disusun dari 10.757 snapshot file Python yang diekstraksi dari riwayat commit dua repositori berskala besar, yaitu Django dan scikit-learn. Enam fitur numerik diekstraksi menggunakan Radon, Pylint, dan Bandit untuk merepresentasikan kompleksitas kode, struktur program, pelanggaran gaya penulisan, serta potensi isu keamanan. Penelitian ini menggunakan tiga algoritma pembelajaran mesin, yaitu Random Forest, Support Vector Machine (SVM), dan Regresi Logistik, untuk tugas klasifikasi. Pemilihan algoritma tersebut didasarkan pada keunggulan masing-masing: Random Forest dipilih karena merupakan metode ensemble yang tahan terhadap overfitting, SVM dipilih karena efektif menangani data yang bersifat non-linear, dan Regresi Logistik digunakan sebagai baseline sederhana dengan proses komputasi yang cepat. Berdasarkan hasil evaluasi, model Random Forest memberikan kinerja terbaik pada metrik akurasi, presisi, recall, dan skor F1. Oleh karena itu, Random Forest dijadikan model utama dalam penelitian ini, sedangkan SVM dan Regresi Logistik hanya digunakan sebagai pembanding. Ketidakseimbangan kelas ditangani menggunakan Synthetic Minority Over-sampling Technique (SMOTE). Hasil evaluasi menunjukkan bahwa model Random Forest memberikan performa terbaik dengan nilai akurasi sebesar 76,91%, precision 89,93%, recall 83,29%, dan F1-score 86,48% pada skema evaluasi stratified cross-validation. Analisis feature importance mengungkapkan bahwa kompleksitas kode dan jumlah pelanggaran gaya penulisan merupakan faktor paling berpengaruh terhadap prediksi bug.

***Kata Kunci**: Python, Static Code Analysis, Machine Learning, Software Defect Prediction, Intelligent Framework.*

ABSTRACT

Software reliability is a crucial aspect of modern software engineering, with the increasing use of Python in the development of open-source and enterprise systems. Early detection of software defects before execution is crucial for maintaining system quality and reducing maintenance costs. However, traditional approaches such as manual code review and rule-based static analysis often suffer from limitations in capturing complex and contextual defects, especially in large-scale projects, and tend to produce high false positive rates. This study proposes an intelligent framework that integrates static code analysis with machine learning approaches to predict bug fix propensity in Python projects. The dataset is composed of 10,757 Python file snapshots extracted from the commit histories of two large-scale repositories, namely Django and scikit-learn. Six numerical features are extracted using Radon,

Pylint, and Bandit to represent code complexity, program structure, style violations, and potential security issues. This study uses three machine learning algorithms, namely Random Forest, Support Vector Machine (SVM), and Logistic Regression, for the classification task. The algorithms were selected based on their respective advantages: Random Forest was chosen because it is an ensemble method that is resistant to overfitting, SVM was chosen because it is effective in handling non-linear data, and Logistic Regression was used as a simple baseline with a fast computational process. Based on the evaluation results, the Random Forest model provided the best performance in accuracy, precision, recall, and F1-score metrics. Therefore, Random Forest was used as the main model in this study, while SVM and Logistic Regression were only used for comparison. Class imbalance was handled using the Synthetic Minority Over-sampling Technique (SMOTE). The evaluation results showed that the Random Forest model provided the best performance with an accuracy value of 76.91%, precision of 89.93%, recall of 83.29%, and F1-score of 86.48% in the stratified cross-validation evaluation scheme. Feature importance analysis revealed that code complexity and the number of writing style violations were the most influential factors in bug prediction.

Keywords: *Python, Static Code Analysis, Machine Learning, Software Defect Prediction, Intelligent Framework.*

I. PENDAHULUAN

Keandalan perangkat lunak (*software reliability*) merupakan aspek fundamental dalam rekayasa perangkat lunak modern, terutama dengan meningkatnya adopsi bahasa pemrograman Python dalam berbagai bidang seperti *data science*, *machine learning*, dan pengembangan aplikasi berbasis web. Kompleksitas kode yang semakin tinggi seiring dengan bertambahnya fitur dan ukuran proyek menyebabkan risiko munculnya *software defects* meningkat secara signifikan. Cacat perangkat lunak yang tidak terdeteksi pada tahap awal pengembangan dapat menyebabkan penurunan kualitas sistem, biaya pemeliharaan yang besar, hingga potensi kerugian ekonomi bagi organisasi [1]. Pendekatan tradisional seperti pengujian manual dan *code review* memiliki keterbatasan karena sangat bergantung pada waktu dan keahlian manusia, serta sulit menjangkau skala kode yang besar. Sejumlah penelitian menunjukkan bahwa metrik kode statis dapat dimanfaatkan secara efektif sebagai prediktor cacat perangkat lunak ketika dikombinasikan dengan teknik pembelajaran mesin[2].

Oleh karena itu, penggunaan pendekatan otomatis berbasis *machine learning* dan *static code analysis* (SCA) menjadi semakin penting. Masalah utama yang dihadapi saat ini adalah alat analisis statis konvensional sering kali menghasilkan laporan yang sangat panjang dengan tingkat *false positive* yang tinggi. Selain itu, belum banyak kerangka kerja yang secara spesifik mengintegrasikan metrik statis dengan pembelajaran mesin untuk proyek Python, mengingat karakteristik unik bahasa ini seperti dynamic typing. Tantangan lainnya adalah ketidakseimbangan kelas (*imbalanced data*) pada dataset cacat perangkat lunak, di mana jumlah modul bersih jauh lebih dominan dibandingkan modul cacat, yang dapat menurunkan kinerja model prediksi.

Sejumlah penelitian terdahulu telah dilakukan untuk mengatasi masalah prediksi cacat perangkat lunak. Albattah dan Alzahrani menguji delapan algoritma *Machine Learning* dan *Deep Learning*, menyimpulkan bahwa model berbasis LSTM memberikan akurasi tertinggi pada dataset yang memadai[3]. Sementara itu, Aimin menerapkan algoritma seperti SVM dan Random Forest dengan optimisasi partikel untuk meningkatkan performa klasifikasi cacat[4]. Penelitian lain oleh Ferenc et al. menunjukkan bahwa penggabungan metrik kode statis dengan *Deep Learning* mampu menandai segmen kode yang mencurigakan, meskipun pendekatan statis murni masih rentan terhadap *false positive*[5]. Mahbub et al. juga berkontribusi dengan menyediakan dataset *Defectors* berskala besar untuk Python, yang membuka peluang riset lebih lanjut pada bahasa ini[6].

Berbeda dengan penelitian sebelumnya yang sebagian besar berfokus pada bahasa statis atau model *Deep Learning* yang berat secara komputasi, penelitian ini mengusulkan kerangka kerja *hybrid* yang efisien menggunakan Random Forest dan teknik sampling SMOTE. Tujuannya adalah mengembangkan

model prediksi cerdas untuk proyek Python yang mampu menyeimbangkan *precision* dan *recall* dalam mendeteksi bug, dengan memanfaatkan fitur numerik dari analisis statis (kompleksitas, gaya penulisan, dan keamanan).

II. LANDASAN TEORI

A. *Static Code Analysis*

Analisis kode statis adalah proses evaluasi perangkat lunak yang dilakukan tanpa mengeksekusi program. Teknik ini menganalisis kode sumber untuk menemukan potensi kerentanan keamanan, kesalahan logika, pelanggaran standar penulisan, atau kompleksitas yang berlebihan[6]. Dalam penelitian ini, digunakan tiga alat analisis utama:

1. Pylint: Alat untuk memeriksa kesalahan kode, memaksakan standar pengkodean (PEP 8), dan mencari *code smells*.
2. Radon: Alat untuk menghitung berbagai metrik kode, termasuk *Cyclomatic Complexity* dan metrik Halstead, yang berguna untuk mengukur tingkat kerumitan alur logika program.
3. Bandit: Alat yang dirancang khusus untuk menemukan isu keamanan umum dalam kode Python.

B. *Machine Learning*

Dalam konteks proyek ini, Machine Learning digunakan untuk membangun model klasifikasi biner yang memprediksi apakah sebuah file Python dari repositori Django dan Scikit-learn mengandung potensi bug (*bug-prone*) atau tidak (*clean*). Tiga algoritma yang diterapkan adalah Random Forest, Support Vector Machine (SVM), dan Logistic Regression. Ketiganya dipilih karena representatif terhadap pendekatan *ensemble*, *kernel-based*, dan *linear modeling*. Untuk keperluan klasifikasi, setiap file direpresentasikan dalam bentuk vektor fitur numerik hasil ekstraksi dari alat analisis statis (Radon, Pylint, Bandit). Model dilatih pada data yang telah diseimbangkan menggunakan SMOTE agar model tidak bias terhadap kelas mayoritas. Dari ketiga model, Random Forest dipilih sebagai model utama karena menunjukkan performa terbaik, sedangkan SVM dan Logistic Regression digunakan sebagai pembanding untuk menilai keunggulan model utama secara objektif.

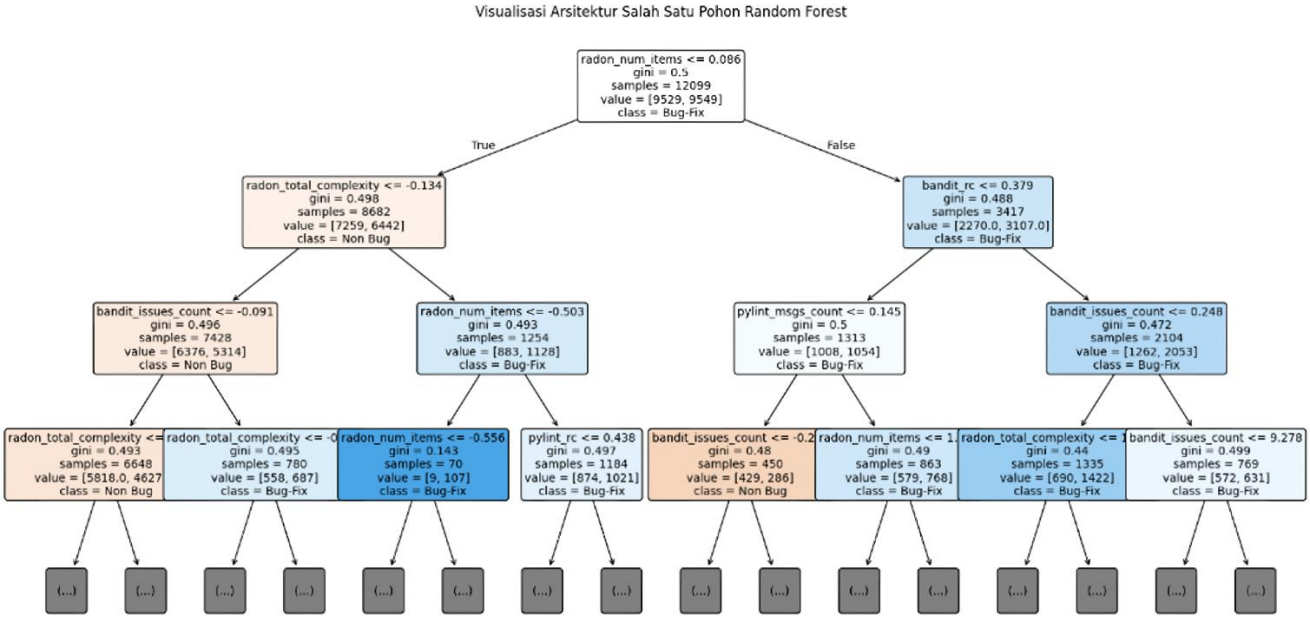
C. *Random Forest*

Random Forest (RF) adalah algoritma ensemble yang menggabungkan sejumlah pohon keputusan (*decision trees*) dan melakukan voting untuk hasil akhir. Algoritma *Random Forest* (RF) adalah metode *ensemble learning* yang bekerja dengan membangun banyak pohon keputusan (*decision trees*) pada saat pelatihan[7]. RF dipilih karena kemampuannya menangani data non-linear, tahan terhadap *overfitting* (dibandingkan pohon keputusan tunggal), dan memberikan estimasi pentingnya fitur (*feature importance*) yang transparan.

Rumus Voting Random Forest:

$$y\hat{} = mode(h1(x), h2(x), \ldots, hn(x)) \qquad (1)$$

di mana $h_i(x)$adalah prediksi dari pohon ke-i, dan $\hat{y}$adalah hasil prediksi akhir.



Gambar I. Visualisasi Arsitektur Random Forest Sumber: Olahan Peneliti (2025)

Visualisasi pada Gambar I menunjukkan salah satu pohon keputusan pada model Random Forest. Struktur hierarki dan pembelahan node berdasarkan nilai fitur menunjukkan bagaimana model ini mampu menangani kompleksitas dan interaksi antar fitur dengan baik. Setiap node menampilkan aturan pembagian, nilai gini, distribusi kelas, dan jumlah sampel pada node tersebut.

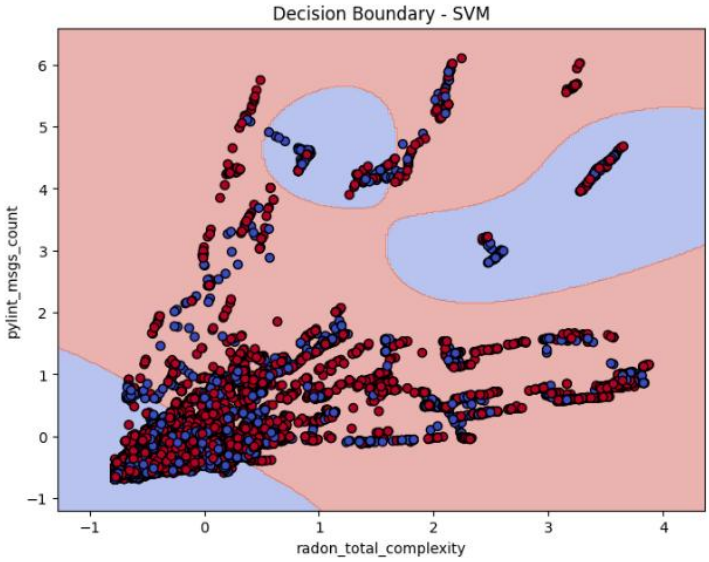
D. Support Vector Machine (SVM)

Support Vector Machine (SVM) adalah algoritma yang mencari hyperplane terbaik yang memisahkan dua kelas dengan margin maksimum. Dalam proyek ini, SVM digunakan sebagai pembanding karena kemampuannya dalam memetakan data secara non-linear menggunakan kernel RBF, meskipun performanya kurang optimal pada data yang tidak seimbang.

Rumus Fungsi Keputusan SVM:

$$f(x) = sign(wTx + b) \qquad (2)$$

SVM efektif untuk klasifikasi data kompleks dengan margin maksimum, tetapi sensitif terhadap ketidakseimbangan kelas pada dataset PROMISE[11]. Visualisasi mengenai batas keputusan (decision boundary) yang dihasilkan oleh model ini dalam memisahkan sebaran data diperlihatkan pada Gambar II.



Gambar II. Decision Boundary SVM Sumber: Olahan Peneliti (2025)

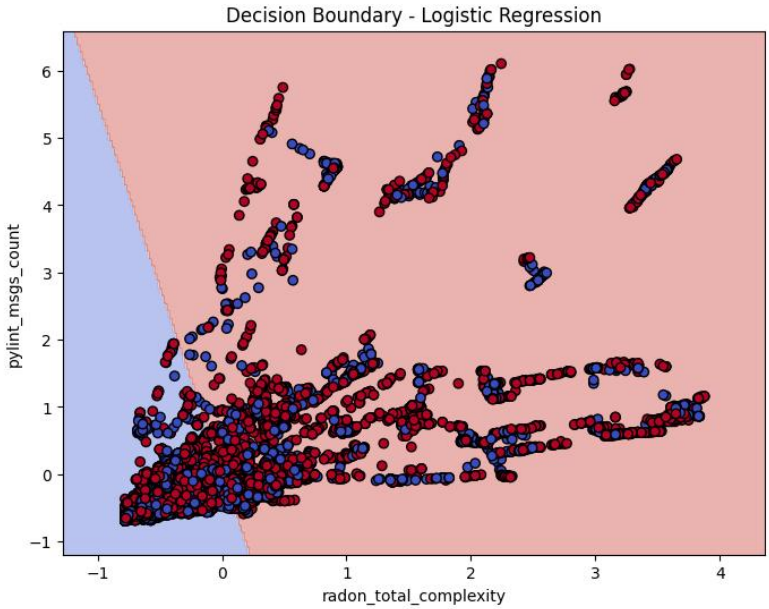
Gambar Decision Boundary SVM dengan kernel RBF menunjukkan bahwa model ini mampu membentuk batas non-linear yang mengikuti distribusi data dengan lebih fleksibel dibanding Logistic Regression, namun tetap memperlihatkan keterbatasan dalam memisahkan kelas secara bersih karena data yang sangat overlap.

E. Logistic Regresion

Logistic Regression digunakan dalam penelitian ini sebagai pembanding karena kekuatannya dalam mengklasifikasikan data kompleks dengan margin maksimum, walau performanya sensitif terhadap distribusi data yang tidak seimbang. Rumus fungsi logistik dinyatakan dalam persamaan (3):

$$P(y = 1 \mid x) = \frac{1}{1+e-(\beta 0+\beta 1x1+\cdots+\beta nxn)} \quad (3)$$

Logistic Regression digunakan untuk membandingkan performanya dengan model kompleks seperti Random Forest dan SVM, meskipun rentan overfitting pada data berdimensi tinggi dalam prediksi cacat software. Decision boundary linear-nya sederhana tapi kurang fleksibel untuk pola non-linear pada fitur seperti radon_total_complexity dan pylint_msgs_count[12]. Representasi visual dari batas keputusan yang dihasilkan oleh model ini disajikan pada Gambar III.



Gambar III. Decision Boundary Logistic Regression Sumber: Olahan Peneliti (2025)

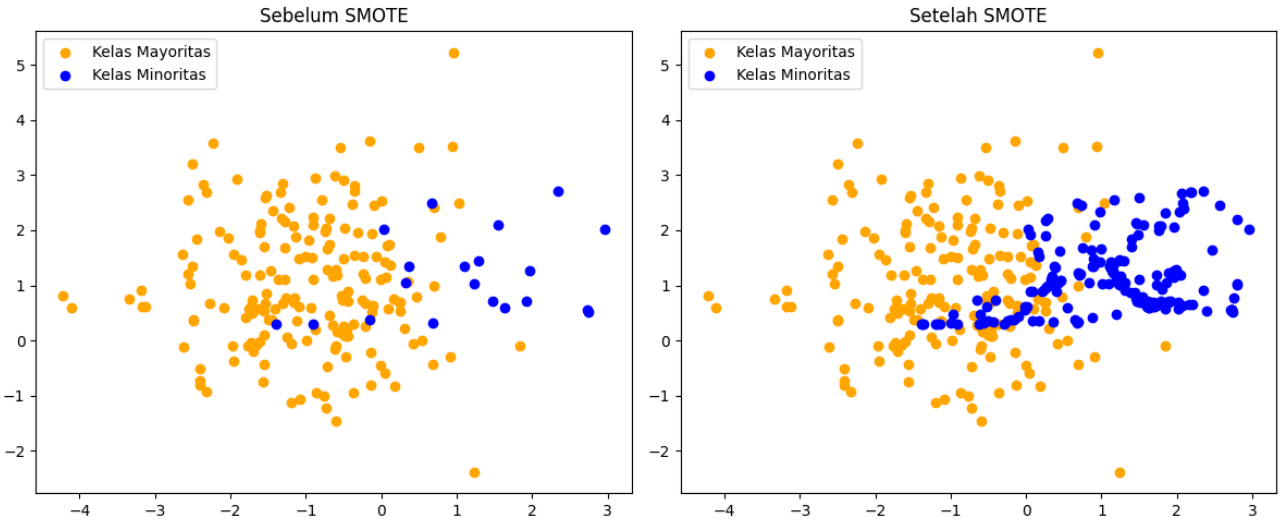
Gambar Decision Boundary Logistic Regression menunjukkan decision boundary dari model Logistic Regression terhadap dua fitur utama, yaitu *radon_total_complexity* dan *pylint_msgs_count*. *Decision boundary linear* memisahkan area prediksi *Bug-Fix* dan *Non Bug* dengan sederhana, merefleksikan sifat model regresi logistik yang cenderung linear dan kurang fleksibel terhadap pola kompleks.

F. Synthetic Minority Over-sampling Technique (SMOTE)

Ketidakseimbangan kelas adalah masalah umum dalam dataset prediksi cacat, di mana kelas minoritas (modul dengan bug) jauh lebih sedikit daripada kelas mayoritas. Hal ini dapat menyebabkan model menjadi bias ke arah kelas mayoritas. SMOTE adalah teknik *oversampling* yang bekerja dengan cara mensintesis sampel baru untuk kelas minoritas berdasarkan tetangga terdekat (*k-nearest neighbors*) dalam ruang fitur, bukan sekadar menduplikasi data yang ada[8]. Pendekatan ini membantu model mempelajari batas keputusan yang lebih akurat bagi kelas minoritas. Proses pembentukan sampel baru melalui teknik interpolasi ini secara visual diilustrasikan pada Gambar IV. Rumus pembentukan sampel sintetis pada SMOTE dinyatakan dalam persamaan (4):

$$xnew = xi + \delta \cdot (xnn - xi) \quad (4)$$

dengan $\delta \sim U(0,1)$, dan x_{nn}adalah tetangga terdekat dari x_i

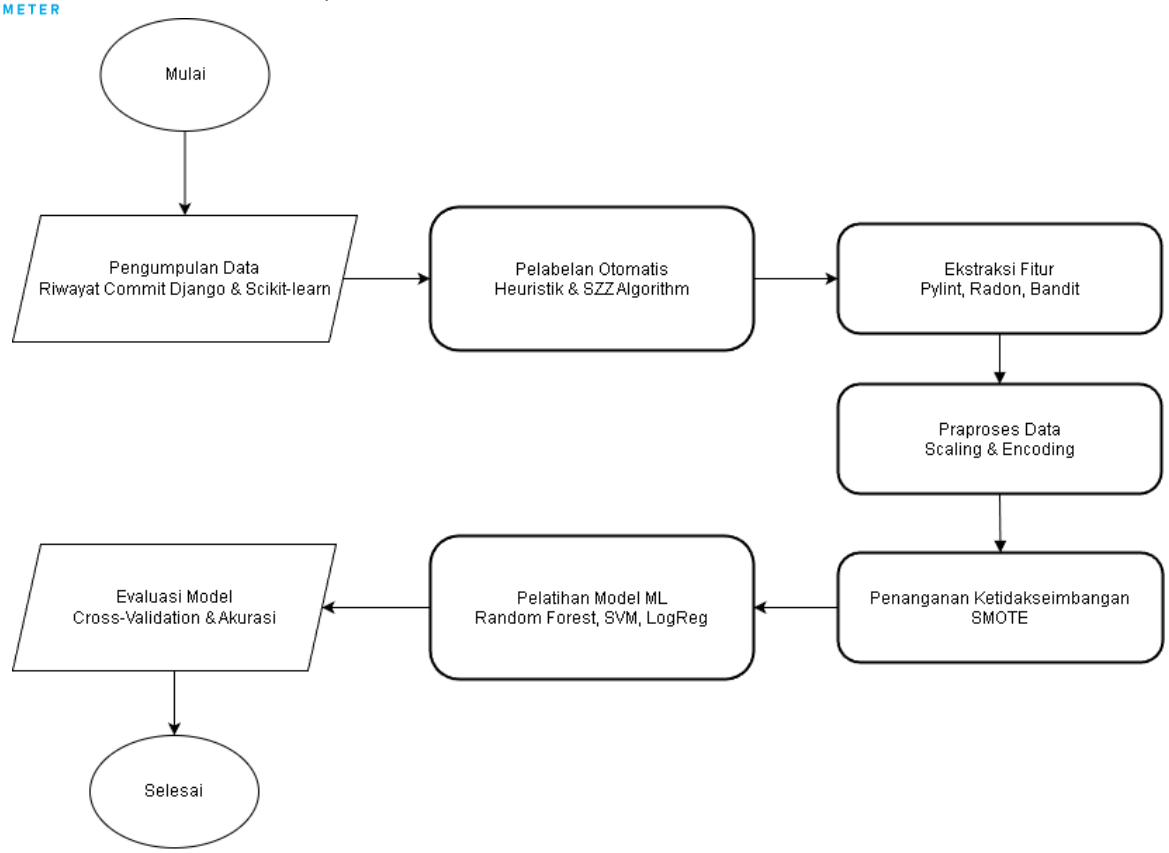


Gambar IV. arsitektur SMOTE Sumber: Olahan Peneliti (2025)

Visualisasi arsitektur SMOTE pada Gambar IV menggambarkan bagaimana sampel baru dari kelas minoritas dibentuk dengan interpolasi antara sampel yang ada dan tetangganya. Teknik ini digunakan untuk mengatasi ketidakseimbangan kelas dalam dataset, dengan menghasilkan representasi sintetis sehingga distribusi menjadi lebih seimbang dan model tidak bias terhadap kelas mayoritas.

III. METODE PENELITIAN

A. Desain dan Alur Penelitian



Gambar V. Flowchart Sumber: Olahan Peneliti (2025)

Penelitian ini dirancang sebagai studi eksperimental dengan pendekatan komparatif yang mengintegrasikan *static code analysis* dan *machine learning*. Proses penelitian mengikuti alur kerja (*pipeline*) yang sistematis, dimulai dari ekstraksi riwayat commit dan pelacakan isu (*issue tracking*) dari repositori perangkat lunak, sebagaimana diilustrasikan pada Gambar V. Data mentah kemudian diproses melalui pelabelan otomatis untuk memisahkan kelas buggy dan clean. Tahap inti melibatkan analisis statis menggunakan *toolchain* khusus untuk mengekstraksi metrik kualitas kode, yang selanjutnya ditransformasikan menjadi fitur numerik. Fitur-fitur ini digunakan untuk melatih model klasifikasi *machine learning* guna memprediksi keberadaan cacat perangkat lunak.

B. Pengumpulan dan Pelabelan Dataset

Dataset dikonstruksi dari dua proyek *open-source* Python berskala besar, yaitu Django dan scikit-learn, untuk memastikan keberagaman karakteristik kode. Pengumpulan data dilakukan dengan mengunduh riwayat commit dan informasi *bug tracker*. Identifikasi commit perbaikan bug dilakukan secara semi-otomatis dengan mendeteksi kata kunci seperti "*fix*" atau referensi nomor isu dalam pesan *commit*, serta menggunakan algoritma SZZ untuk menautkan perbaikan ke *commit* penyebabnya (*bug-inducing commit*). Setiap perubahan pada file Python diberi label kelas 1 (*Bug-Fix*) jika terlibat dalam perbaikan bug, dan kelas 0 (*Non Bug-Fix*) jika tidak.

C. Ekstraksi Fitur Analisis Statis

Fitur prediktif diekstraksi menggunakan tiga alat analisis statis utama:

1. Pylint: Digunakan untuk memeriksa kepatuhan terhadap standar penulisan (PEP 8) dan mendeteksi kesalahan potensial, menghasilkan fitur berupa skor kualitas dan jumlah peringatan.
2. Radon: Menghitung metrik kompleksitas kode, khususnya *Cyclomatic Complexity*, serta mengukur metrik ukuran struktural dari *Abstract Syntax Tree* (AST) kode sumber.
3. Bandit: Memindai kode untuk menemukan kerentanan keamanan umum (seperti injeksi SQL atau penggunaan fungsi berisiko). Output dari ketiga alat tersebut dinormalisasi menggunakan teknik

Standard Scaling agar memiliki skala seragam dan distribusi yang mendekati normal, sehingga optimal saat diproses oleh algoritma yang sensitif terhadap variansi fitur seperti SVM

D. Skenario Eksperimen dan Evaluasi

Eksperimen dilakukan menggunakan tiga algoritma klasifikasi: Random Forest (RF) sebagai model utama, serta Support Vector Machine (SVM) dan Logistic Regression (sebagai baseline linear). Random Forest dikonfigurasi dengan 100 pohon keputusan (n_estimators=100) untuk meningkatkan stabilitas prediksi. Mengingat adanya ketidakseimbangan kelas yang signifikan pada dataset, teknik *Synthetic Minority Over-sampling Technique* (SMOTE) diterapkan khusus pada data pelatihan untuk menyeimbangkan proporsi kelas tanpa menyebabkan kebocoran data (*data leakage*) pada data uji.

Evaluasi model menggunakan skema *Stratified K-Fold Cross-Validation* (k=10) untuk memastikan validitas hasil. Kinerja model diukur menggunakan metrik standar klasifikasi yang meliputi akurasi, *precision*, *recall*, dan *F1-score*, dengan fokus utama pada keseimbangan antara *precision* dan *recall* dalam mendeteksi kelas minoritas (*bug*).

IV. HASIL DAN PEMBAHASAN

A. Deskripsi Dataset dan Statistik Fitur

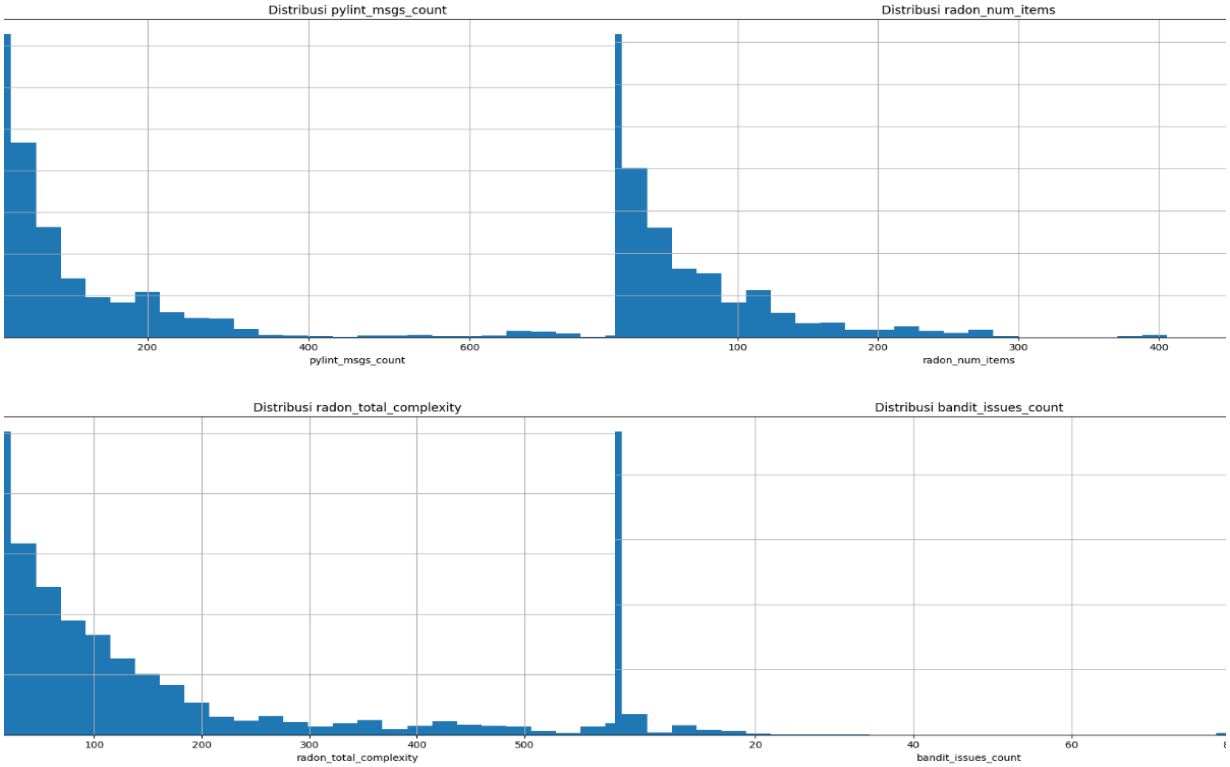
Dataset penelitian terdiri dari 10.757 file Python yang diekstraksi dari repositori Django dan scikit-learn. Berdasarkan analisis awal, ditemukan ketidakseimbangan kelas yang signifikan: kelas Bug-Fix mendominasi sebanyak 9.539 data (88,68%), sedangkan kelas *Non Bug-Fix* hanya 1.218 data (11,32%) . Ketimpangan ini dikonfirmasi melalui visualisasi distribusi kelas yang menunjukkan dominasi mayoritas pada aktivitas perbaikan bug.

Analisis statistik deskriptif terhadap enam fitur numerik menunjukkan variasi yang tinggi. Fitur *radon_total_complexity* memiliki rata-rata 126,96 dengan nilai maksimum mencapai 690, mengindikasikan keberadaan modul dengan kompleksitas ekstrem yang sulit dipelihara[9]. Sementara itu, fitur *pylint_msgs_count* memiliki distribusi yang sangat timpang (*heavy-tailed*), di mana sebagian besar file memiliki sedikit peringatan, tetapi terdapat file tertentu dengan jumlah pelanggaran mencapai 922. Rangkuman hasil statistik deskriptif untuk seluruh fitur numerik disajikan pada Tabel I.

Tabel I. Statistik Deskriptif Fitur Numerik

	radon_total_complexity	**radon_num_items**	**pylint_msgs_count**	**pylint_rc**	**bandit_issues_count**	**bandit_rc**
count	10757	10757	10757	10757	10757	10757
mean	126,9630938	59,14753184	102,2022869	27,62201357	2,453472158	0,392023798
std	156,8382176	75,95116998	140,788419	4,388034238	9,08560156	0,48822464
min	0	0	0	0	0	0
25%	25	12	22	26	0	0
50%	70	33	50	30	0	0
75%	150	80	126	30	2	1
max	690	529	922	30	94	1

Visualisasi histogram memperlihatkan bahwa seluruh fitur memiliki distribusi *right-skewed* (miring ke kanan). Pola ini menegaskan bahwa data bersifat heterogen dan mengandung outlier, yang mendukung keputusan penggunaan model non-linear seperti Random Forest dibandingkan model linear[10]. Karakteristik sebaran data fitur tersebut diperlihatkan secara visual pada Gambar VI.



Gambar VI. Distribusi fitur yang menunjukkan pola right-skewed Sumber: Olahan Peneliti (2025)

B. Penanganan Ketidakseimbangan Kelas

Sebelum pelatihan model, distribusi kelas yang tidak seimbang (88,7% vs 11,3%) berpotensi menyebabkan bias prediksi. Penerapan SMOTE pada data pelatihan berhasil menyeimbangkan proporsi kelas menjadi 50:50. Hal ini krusial untuk memastikan model dapat mempelajari pola kelas minoritas tanpa mengabaikan kelas mayoritas.

C. Evaluasi Performa Model

Hasil evaluasi eksperimen menggunakan Stratified Cross-Validation ditampilkan pada Tabel II.

Tabel II. Perbandingan Performa Model Klasifikasi.

Model	Accuracy	Precision	Recall	F1
Random Forest	0.7691	0.8993	0.8329	0.8648
SVM	0.5007	0.9025	0.4898	0.6348
Logistic Regression	0.4357	0.8988	0.4098	0.5628

Berdasarkan tabel tersebut, Random Forest memberikan performa terbaik dengan akurasi 76,91% dan F1-score 86,48%. Meskipun Support Vector Machine (SVM) dan Logistic Regression memiliki nilai *precision* yang sangat tinggi (>90%), keduanya gagal dalam *recall* (<50%), yang berarti banyak kasus bug gagal terdeteksi. Random Forest menunjukkan keseimbangan terbaik antara *precision* (89,93%) dan *recall* (83,29%), menjadikannya model yang paling robust untuk mendeteksi cacat perangkat lunak dalam skenario ini.

D. Analisis Kontribusi Fitur

Analisis feature importance dari model Random Forest mengungkapkan faktor dominan yang memengaruhi prediksi cacat.

Tabel III. Peringkat Feature Importance.

No	Fitur	Nilai Importance
1	radon_total_complexity	0.3064
2	pylint_msgs_count	0.2870
3	radon_num_items	0.2371
4	pylint_rc	0.0734
5	bandit_issues_count	0.0691
6	bandit_rc	0.0269

Seperti terlihat pada Tabel III, *radon_total_complexity* memiliki kontribusi tertinggi (0.3064), diikuti oleh *pylint_msgs_count* (0.2870). Temuan ini mengonfirmasi hipotesis bahwa kompleksitas kode dan pelanggaran gaya penulisan adalah indikator terkuat adanya *bug*[10]. Sebaliknya, metrik keamanan (*bandit_issues_count*) memiliki kontribusi yang relatif kecil (0.0691) pada level file, menunjukkan bahwa isu keamanan mungkin memerlukan analisis yang lebih granular daripada sekadar hitungan jumlah isu.

V. KESIMPULAN

Penelitian ini berhasil mengembangkan kerangka kerja cerdas untuk memprediksi cacat perangkat lunak pada proyek Python dengan mengintegrasikan analisis kode statis dan machine learning. Berdasarkan hasil eksperimen, dapat disimpulkan bahwa:

1. Kompleksitas kode (*Cyclomatic Complexity*) dan pelanggaran gaya penulisan (*Code Style Violations*) merupakan indikator paling kuat dalam menentukan kerentanan *bug* pada file Python.
2. Penerapan teknik SMOTE terbukti efektif menyeimbangkan distribusi data latih, sehingga mencegah model menjadi bias terhadap kelas mayoritas.
3. Model Random Forest mengungguli SVM dan Logistic Regression dengan mencapai keseimbangan terbaik antara precision (89,93%) dan recall (83,29%), menjadikannya model yang paling andal untuk mendeteksi modul cacat.

Penelitian ini merekomendasikan penggunaan model prediksi ini sebagai alat bantu prioritas dalam pipeline pengujian perangkat lunak, sehingga pengembang dapat fokus pada modul-modul yang diprediksi berisiko tinggi.

VI. DAFTAR PUSTAKA

[1] S. Stradowski and L. Madeyski, "Industrial applications of software defect prediction using machine learning: A business-driven systematic literature review," *Inf. Softw. Technol.*, vol. 159, no. March, p. 107192, 2023, doi: 10.1016/j.infsof.2023.107192.

[2] V. Ganesh, T. Leek, and M. Rinard, "Taint-based Directed Whitebox Fuzzing," pp. 474–484, 2009, doi: https://doi.org/10.1109/ICSE.2009.5070546.

[3] W. Albattah and M. Alzahrani, "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach," *AI*, vol. 5, no. 4, pp. 1743–1758, 2024, doi: 10.3390/ai5040086.

[4] K. Aimin, "Software defect prediction using machine learning techniques," *Adv. Artif. Hum. Intell. Mod. Era*, pp. 180–195, 2023, doi: 10.4018/979-8-3693-1301-5.ch010.

[5] R. Ferenc, D. Bán, T. Grósz, and T. Gyimóthy, "Deep learning in static, metric-based bug prediction," *Array*, vol. 6, no. March, p. 100021, 2020, doi: 10.1016/j.array.2020.100021.

[6] P. Mahbub, O. Shuvo, and M. Masudur Rahman, "Defectors: A Large, Diverse Python Dataset for Defect Prediction," *Proc. - 2023 IEEE/ACM 20th Int. Conf. Min. Softw. Repos. MSR 2023*, pp. 393–397, 2023, doi: 10.1109/MSR59073.2023.00085.

[7] L. E. O. Breiman, "Random Forests," pp. 5–32, 2001, doi: https://doi.org/10.1023/A:1010933404324.

[8] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE : Synthetic Minority Over-sampling Technique," vol. 16, pp. 321–357, 2002, doi: https://doi.org/10.1613/jair.953.

[9] T. J. Mccabe, "A Complexi-ty," no. 4, pp. 308–320, 1976, doi: https://doi.org/10.1109/TSE.1976.233837.

[10] S. Lessmann, S. Member, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction : A Proposed Framework and Novel Findings," vol. 34, no. 4, pp. 485–496, 2008, doi: https://doi.org/10.1109/TSE.2008.35.
[11] A. Priyadarshini and V. Krishnapriya, "Software Defect Prediction across Multiple Datasets using Classification Techniques," vol. 10, pp. 158–167, 2025, doi: https://doi.org/10.52783/jisem.v10i41s.7807.
[12] R. Oueslati and G. Manita, "Software Defect Prediction Using Integrated Logistic Regression and Fractional Chaotic Grey Wolf Optimizer," no. Enase, pp. 633–640, 2024, doi: 10.5220/0012704600003687.