INTERNAL (Information System Journal)
Volume 9 No. 1 | Juni 2026 : 63-74
DOI: 10.32627
http://jurnal.masoemuniveristy.ac.id/index.php/internal
p-ISSN: 2621-9433
e-ISSN: 2656-0259

Deteksi Sistem Cerdas Kerentanan Kode PHP Menggunakan Hybrid ML dan LLM

Moh. Erdda Habiby[1], Miki Wijana[2]
[1]Teknik Informatika, Universitas Bina Nusantara, Indonesia
[2]Sistem Informasi, Universitas Ma'soem, Indonesia
moh.erdda@binus.ac.id

Info Artikel

Sejarah artikel :
Diterima Mei 2026
Direvisi Mei 2026
Disetujui Juni 2026
Diterbitkan Juni 2026

ABSTRACT

Security vulnerabilities in PHP programming code remain one of the major threats to the integrity of web applications, especially when software development processes are not supported by automated and adaptive security analysis mechanisms. This study aims to develop an intelligent system for vulnerability detection and automatic code remediation using a Hybrid Machine Learning (ML) and Large Language Model (LLM) approach. The system combines Term Frequency-Inverse Document Frequency (TF-IDF), Abstract Syntax Tree (AST) Parsing, and the Random Forest algorithm for vulnerability classification, while integrating the Google Gemini API as a dynamic recommendation layer to generate contextual and adaptive secure coding suggestions. The dataset consists of 320 PHP code snippets covering SQL Injection, XSS, File Inclusion, Command Injection, Unsafe File Upload, and safe code samples. Experimental evaluation using accuracy, precision, recall, F1-score, and confusion matrix shows that the model achieved an overall accuracy of 86.25% with a macro average F1-score of 0.86. This study demonstrates that integrating Hybrid ML and LLM-based remediation using the Gemini API has strong potential for developing intelligent static code analysis systems for PHP web applications.

Keywords: *Abstract Syntax Tree; LLM; PHP Security; Random Forest; TF-IDF.*

ABSTRAK

Kerentanan keamanan pada kode pemrograman PHP masih menjadi salah satu ancaman utama terhadap integritas aplikasi web, terutama ketika proses pengembangan perangkat lunak belum didukung oleh mekanisme analisis keamanan yang otomatis dan adaptif. Penelitian ini bertujuan mengembangkan sistem cerdas untuk mendeteksi kerentanan sekaligus memberikan rekomendasi perbaikan kode secara otomatis menggunakan pendekatan *Hybrid Machine Learning* (ML) dan *Large Language Model* (LLM). Sistem dibangun dengan menggabungkan TF-IDF, AST Parsing, dan *algoritma Random Forest* untuk proses klasifikasi kerentanan, serta mengintegrasikan *Google Gemini* API sebagai lapisan rekomendasi yang dinamis dan kontekstual. Dataset terdiri dari 320 potongan kode PHP mencakup SQL *Injection*, XSS, *File Inclusion*, *Command Injection*, *Unsafe File Upload*, dan kode aman. Evaluasi menggunakan *accuracy*, *precision*, *recall*, F1-*score*, dan *confusion matrix* menunjukkan akurasi keseluruhan 86,25% dengan *macro average* F1-score sebesar 0,86. Penelitian ini menunjukkan bahwa integrasi Hybrid ML dan Gemini API memiliki potensi besar dalam pengembangan *static code analysis* cerdas pada aplikasi web berbasis PHP.

Kata Kunci : *Abstract Syntax Tree*; LLM; PHP *Security*; *Random Forest*; TF-IDF.

PENDAHULUAN

Perkembangan aplikasi berbasis web yang semakin pesat menjadikan aspek keamanan perangkat lunak sebagai komponen yang sangat penting dalam proses pengembangan sistem informasi. Salah satu bahasa pemrograman yang masih banyak digunakan dalam pembangunan aplikasi web adalah PHP karena memiliki karakteristik fleksibel, mudah dipelajari, serta didukung oleh komunitas pengembang yang sangat luas. PHP banyak diterapkan pada berbagai sistem seperti layanan akademik, pemerintahan, e-commerce, hingga aplikasi enterprise. Namun demikian, tingginya penggunaan PHP juga diikuti oleh meningkatnya potensi kerentanan keamanan apabila proses pengembangan tidak menerapkan prinsip *secure coding* secara konsisten [1], [10].

Berbagai bentuk kerentanan yang umum ditemukan pada aplikasi berbasis PHP meliputi SQL *Injection*, *Cross-Site Scripting* (XSS), *File Inclusion*, *Command Injection, Remote Code Execution* (RCE), *Path Traversal*, hingga *Unsafe File Upload.* Kerentanan tersebut dapat dimanfaatkan oleh pihak yang tidak berwenang untuk memperoleh akses ilegal terhadap data sensitif, merusak sistem, bahkan mengambil alih server secara keseluruhan. Kondisi ini umumnya disebabkan oleh lemahnya proses validasi *input*, penggunaan fungsi-fungsi berisiko seperti *eval*(), *include*(), dan system() tanpa kontrol keamanan yang memadai, serta kurangnya analisis keamanan sejak tahap awal pengembangan perangkat lunak [2], [11].

Pendekatan konvensional dalam mendeteksi kerentanan kode umumnya masih dilakukan melalui pemeriksaan manual oleh programmer atau menggunakan *static code analyzer* berbasis *rule-based*. Meskipun metode tersebut masih relevan, pendekatan ini memiliki beberapa keterbatasan, seperti sifatnya yang reaktif, kurang adaptif terhadap pola serangan baru, serta tidak efisien ketika diterapkan pada proyek dengan skala kode yang besar. Selain itu, pendekatan berbasis aturan sering kali gagal mengenali pola kerentanan baru yang tidak sesuai dengan rule yang telah ditentukan [3], [5]. Oleh karena itu, diperlukan pendekatan yang lebih adaptif dan cerdas melalui pemanfaatan machine *learning maupun deep learning* agar proses deteksi kerentanan dapat dilakukan secara otomatis dan lebih dini.

Penelitian sebelumnya yang dilakukan oleh Habiby (2025) menunjukkan bahwa kombinasi metode TF-IDF dan AST Parsing mampu meningkatkan kemampuan deteksi kerentanan pada kode PHP melalui analisis tekstual dan struktural secara bersamaan [13]. Penggunaan TF-IDF efektif dalam merepresentasikan token-token penting pada kode sumber, sedangkan Random Forest terbukti mampu memberikan performa klasifikasi yang stabil pada data berdimensi tinggi [8], [9]. Namun demikian, penelitian tersebut masih memiliki keterbatasan pada rendahnya confidence score hasil prediksi serta belum tersedianya mekanisme rekomendasi perbaikan kode secara otomatis. Penelitian lain seperti *DeepTective* menunjukkan bahwa pendekatan deep learning berbasis GRU dan GCN mampu mencapai performa yang tinggi dalam deteksi kerentanan [15], sedangkan studi oleh Cetin et al. (2024) memperlihatkan bahwa *Large Language Model* (LLM) memiliki potensi besar dalam melakukan analisis statis terhadap kerentanan kode PHP secara lebih kontekstual [14].

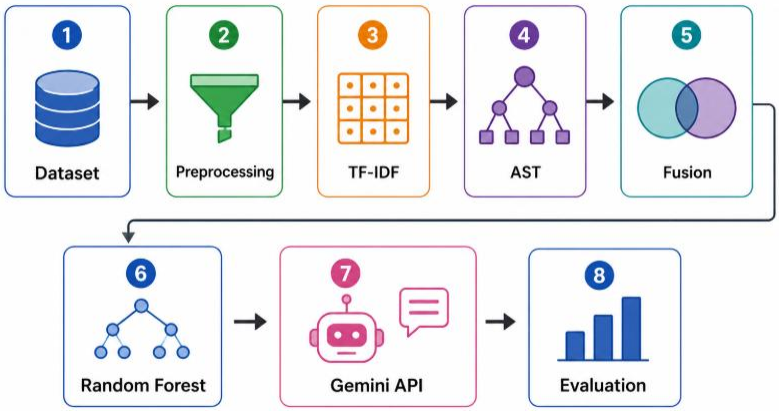
p-ISSN: 2621-9433
e-ISSN: 2656-0259

Dalam praktik *secure coding modern*, pengembang tidak hanya membutuhkan informasi mengenai keberadaan kerentanan, tetapi juga memerlukan penjelasan penyebab kerentanan beserta rekomendasi perbaikannya secara langsung. Perkembangan Large Language Model seperti *Google Gemini* membuka peluang baru dalam bidang keamanan perangkat lunak karena mampu memahami konteks kode, menjelaskan risiko keamanan, serta menghasilkan rekomendasi perbaikan secara natural dan kontekstual [5]. Bahkan, beberapa penelitian menunjukkan bahwa LLM mampu mendukung proses automated program repair terhadap kode yang rentan [6], [12].

Berdasarkan permasalahan tersebut, penelitian ini mengusulkan pengembangan sistem cerdas untuk deteksi dan rekomendasi perbaikan kerentanan kode PHP menggunakan pendekatan *Hybrid Machine Learning* dan *Large Language Model* dengan memanfaatkan *Google Gemini* API sebagai lapisan rekomendasi otomatis. Sistem ini dirancang untuk meningkatkan akurasi deteksi, menghasilkan *confidence score* yang lebih baik, serta memberikan rekomendasi perbaikan kode secara dinamis guna mendukung penerapan *secure coding* yang lebih efektif. *Novelty* penelitian ini terletak pada integrasi antara klasifikasi berbasis TF-IDF, AST Parsing, dan *Random Forest* dengan rekomendasi perbaikan otomatis berbasis Gemini API dalam satu sistem *static code analysis* yang adaptif dan kontekstual.

METODE

Penelitian ini menggunakan pendekatan eksperimental dengan model pengembangan sistem berbasis prototyping. Pendekatan ini dipilih karena memungkinkan proses pengembangan sistem dilakukan secara iteratif, dinamis, dan dapat disesuaikan berdasarkan hasil pengujian pada setiap tahapan penelitian. Sistem yang dikembangkan memiliki dua fungsi utama, yaitu mendeteksi kerentanan pada kode PHP dan menghasilkan rekomendasi perbaikan secara otomatis menggunakan pendekatan *Hybrid Machine Learning* dan *Large Language Model* yang terintegrasi dengan *Google Gemini* API.



Gambar 1. Alur penelitian

Tahapan penelitian dimulai dari proses pengumpulan *dataset*, *preprocessing data*, ekstraksi fitur menggunakan TF-IDF, analisis struktur sintaksis melalui AST *Parsing*, penggabungan fitur (feature fusion), klasifikasi menggunakan algoritma *Random Forest*, integrasi lapisan rekomendasi berbasis *Gemini* API, hingga evaluasi performa sistem.

Pengumpulan dan Persiapan Dataset

Dataset yang digunakan berasal dari dua sumber utama, yaitu dataset publik dan dataset tambahan yang disusun secara sintetis. *Dataset* publik diperoleh dari berbagai sumber seperti *Damn Vulnerable Web Application* (DVWA), OWASP *WebGoat security sample, repository GitHub*, serta berbagai contoh kerentanan aplikasi PHP yang tersedia secara terbuka [1], [10]. Sementara itu, dataset tambahan dibuat berdasarkan pola kerentanan yang umum ditemukan pada aplikasi web berbasis PHP, seperti SQL *Injection*, *Cross-Site Scripting* (XSS), *File Inclusion, Command Injection, Unsafe File Upload, Remote Code Execution* (RCE), *Path Traversal*, SSRF, CSRF, *hardcoded secret, weak crypto*, dan *insecure deserialization*. Total *dataset* yang digunakan dalam penelitian ini berjumlah 320 potongan kode PHP yang terdiri dari kode aman dan kode rentan.

Preprocessing dan Ekstraksi TF-IDF

Tahap preprocessing dilakukan untuk membersihkan komentar, karakter yang tidak relevan, serta melakukan proses tokenisasi terhadap kode sumber PHP. Setelah tahap tersebut selesai, metode TF-IDF digunakan untuk mengubah kode program menjadi representasi numerik berbasis bobot token. Token-token seperti $_GET, $_POST, *mysql_query*, eval(), *include*(), dan *system*() menjadi indikator penting dalam proses identifikasi pola kerentanan. Pendekatan TF-IDF dipilih karena efektif dalam menangkap distribusi token penting yang berkaitan dengan pola serangan pada kode sumber [8].

AST Parsing dan Advanced Pattern Engine

Selain fitur tekstual, sistem menggunakan AST Parsing untuk menganalisis struktur sintaksis kode. AST merepresentasikan kode sumber sebagai pohon hierarki di mana setiap node merupakan konstruksi sintaksis seperti fungsi, ekspresi, kondisi, dan argumen. Pendekatan AST banyak digunakan karena mampu merepresentasikan hubungan struktural antar elemen kode yang sulit ditangkap oleh analisis berbasis token semata [10].. Sistem melakukan traversal pada *node-node* kritis—khususnya node *FunctionCall* dan *Assignment* yang melibatkan sumber input pengguna (node $_GET, $_POST, $_REQUEST) — untuk mengidentifikasi pola berbahaya. Sistem memeriksa pola seperti eval($_GET), *include*($_POST), *system*($_REQUEST), dan *move_uploaded_file*() tanpa validasi. *Pattern engine* diperluas dengan aturan berbobot untuk 10 kategori kerentanan: SQL *Injection*, XSS, *File Inclusion*, *Command Injection*, RCE, *Unsafe File Upload*, *Path Traversal, Hardcoded Secret, Weak Crypto*, dan CSRF.

Setiap pattern diberi skor berdasarkan tingkat risikonya. Sistem juga menambahkan indikator aman seperti *prepare*(), *bindParam*(), *htmlspecialchars*(), *realpath*(), *basename*(), dan *password_hash*() sebagai pengurang skor risiko. Dengan

p-ISSN: 2621-9433
e-ISSN: 2656-0259

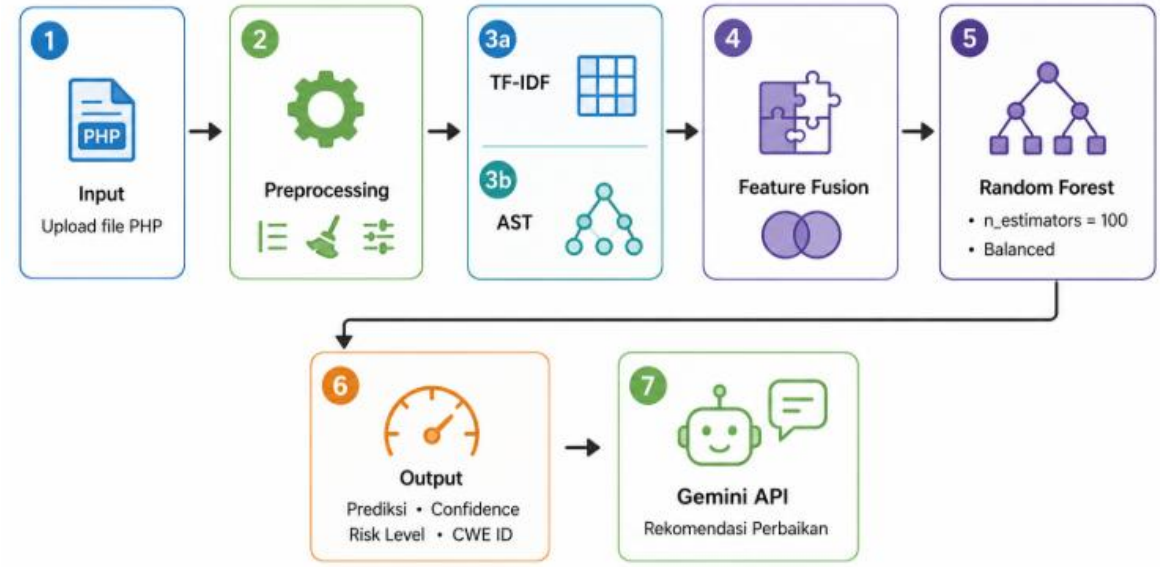
pendekatan ini, sistem tidak hanya mendeteksi keberadaan fungsi berbahaya, tetapi juga mempertimbangkan mitigasi yang telah diterapkan dalam kode.

Integrasi Hybrid ML dan Google Gemini API

Hasil ekstraksi fitur dari TF-IDF dan AST kemudian digabungkan menggunakan teknik *feature fusion* berupa *concatenation* sebelum masuk ke tahap klasifikasi. *Algoritma Random Forest* dipilih sebagai model klasifikasi karena memiliki kemampuan yang baik dalam menangani data berdimensi tinggi, relatif stabil terhadap *noise*, dan memberikan performa yang konsisten pada tugas klasifikasi keamanan perangkat lunak [9]. Model dilatih menggunakan pembagian data *training* dan *testing* sebesar 80:20, kemudian dievaluasi menggunakan *metrik accuracy, precision, recall*, F1-*score*, dan *confusion matrix*.

Setelah jenis kerentanan terdeteksi, lapisan rekomendasi berbasis *Google Gemini* API digunakan untuk menghasilkan penjelasan risiko dan saran perbaikan kode yang dinamis dan kontekstual. Sistem mengirimkan prompt terstruktur yang berisi jenis kerentanan terdeteksi, potongan kode yang bermasalah, dan pola AST yang memicu deteksi. Gemini API kemudian menghasilkan rekomendasi perbaikan secara *natural language* yang disesuaikan dengan konteks kode pengguna, mencakup penjelasan risiko, contoh kode yang sudah diperbaiki, serta *best practices secure coding* yang relevan. Pendekatan pemanfaatan LLM untuk perbaikan kode otomatis ini sejalan dengan temuan Xia et al. [6], Cao et al. [7], dan Wang et al. [12] yang menunjukkan kemampuan LLM dalam mendeteksi dan memperbaiki kerentanan kode secara kontekstual.

Arsitektur Sistem



Gambar 2. Arsitektur Sistem Hybrid ML + Gemini API untuk Deteksi Kerentanan Kode PHP

Gambar 2 mengilustrasikan arsitektur sistem secara keseluruhan yang terdiri dari enam lapisan utama: (1) *Input Layer* menerima *file* PHP yang diunggah pengguna, (2) *Preprocessing Layer* melakukan tokenisasi dan pembersihan kode, (3)

Feature Extraction Layer mengekstrak fitur tekstual via TF-IDF dan fitur struktural via AST *Pattern Engine secara paralel*, (4) *Classification Layer* menggabungkan kedua *vektor fitur* dan menjalankan *Random Forest* untuk menghasilkan prediksi kelas dan *confidence score*, (5) *Output Layer* menampilkan hasil prediksi beserta *risk level* dan CWE ID, serta (6) Recommendation Layer menggunakan *Google Gemini* API untuk menghasilkan saran perbaikan yang dinamis berdasarkan kode asli dan jenis kerentanan terdeteksi.

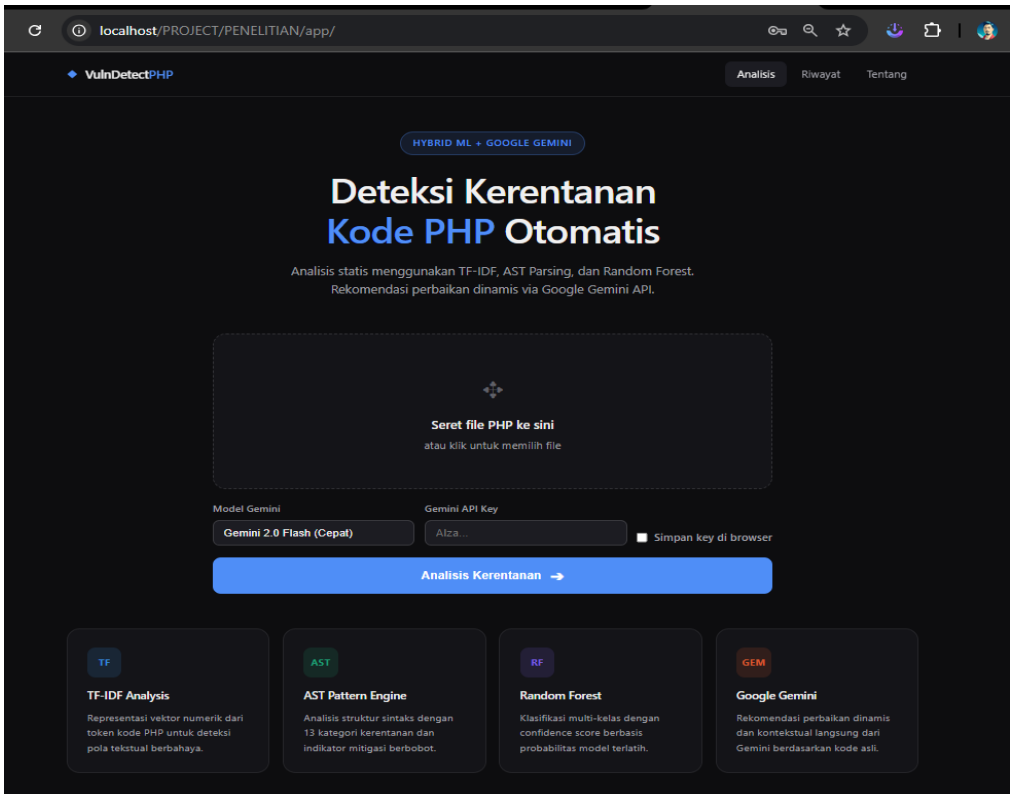
Evaluasi Sistem

Evaluasi sistem dilakukan menggunakan *metrik accuracy, precision, recall, F1-score, confidence score*, dan *confusion matrix*. Evaluasi difokuskan pada kemampuan sistem dalam mendeteksi kerentanan secara akurat serta memberikan rekomendasi perbaikan yang kontekstual melalui integrasi *Google Gemini* API.

HASIL DAN PEMBAHASAN

Sistem yang dikembangkan merupakan aplikasi web berbasis PHP yang dapat menerima unggahan file .php, menganalisis pola kerentanan, menampilkan *confidence score*, memberikan rekomendasi perbaikan melalui *Google Gemini* API, dan menyimpan hasil analisis ke dalam file laporan. Sistem ini dikembangkan sebagai alat bantu static code analysis ringan yang dapat dijalankan secara lokal menggunakan XAMPP atau *web server standar*.

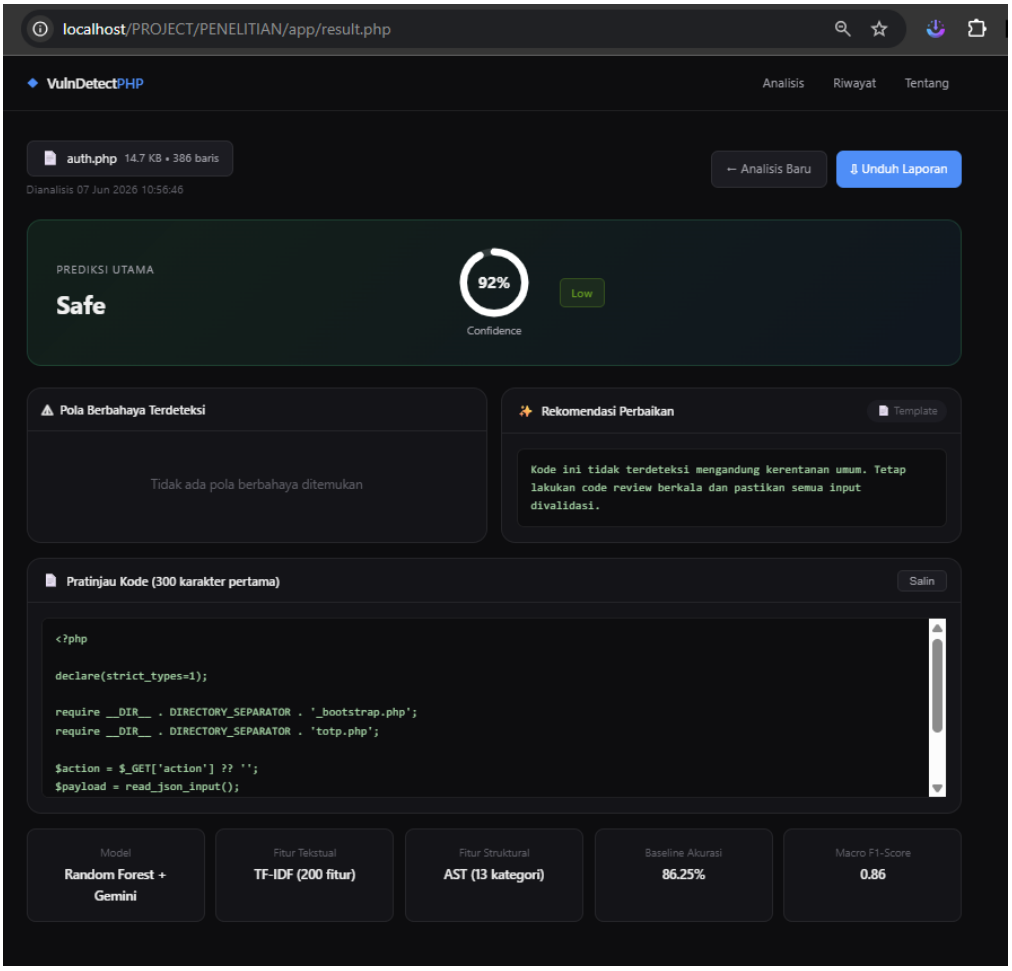
Implementasi Sistem



Gambar 3. Antarmuka utama sistem deteksi kerentanan kode PHP berbasis Hybrid ML dan Google Gemini API

p-ISSN: 2621-9433
e-ISSN: 2656-0259

Gambar 3 menunjukkan implementasi antarmuka sistem yang dikembangkan dalam penelitian ini. Sistem menyediakan fasilitas unggah file PHP yang selanjutnya diproses melalui tahap *preprocessing*, *ekstraksi fitur* TF-IDF dan AST, serta klasifikasi menggunakan *algoritma Random Forest*. Hasil prediksi ditampilkan dalam bentuk jenis kerentanan, *confidence score*, dan tingkat risiko. Selain itu, sistem terintegrasi dengan *Google Gemini* API untuk menghasilkan rekomendasi perbaikan kode yang sesuai dengan kerentanan yang terdeteksi.



Gambar 4. Antarmuka Hasil Prediksi Kerentanan dan Rekomendasi Perbaikan Kode

Gambar 4 menunjukkan hasil analisis sistem terhadap file PHP yang diuji. Sistem menampilkan prediksi utama berupa kategori kerentanan, *confidence score*, dan tingkat risiko yang dihasilkan oleh model *Random Forest*. Selain itu, sistem juga menampilkan pola berbahaya yang terdeteksi, cuplikan kode sumber yang dianalisis, serta rekomendasi perbaikan yang dihasilkan melalui integrasi *Google Gemini* API. Informasi tersebut membantu pengguna memahami tingkat keamanan kode dan tindakan mitigasi yang dapat dilakukan

Hasil Pengujian Sistem

Pengujian dilakukan menggunakan 64 file PHP (20% dari total dataset 320 sampel) yang mewakili berbagai kategori kerentanan. Sistem mampu mendeteksi

kerentanan utama dan menampilkan risiko tambahan apabila ditemukan lebih dari satu pola berbahaya dalam satu *file*. Contoh hasil prediksi per file ditampilkan pada Tabel 1.

Tabel 1. Contoh Hasil Prediksi Sistem

Nama File	Prediksi Utama	Confidence	Risk Level	Label Aktual
login.php	SQL *Injection*	86%	*High*	SQL *Injection*
comment.php	XSS	82%	*Medium*	XSS
page.php	*File Inclusion*	88%	*High*	*File Inclusion*
upload.php	*Unsafe File Upload*	80%	*Medium*	*Unsafe File Upload*
config.php	*Hardcoded Secret*	85%	*High*	*Hardcoded Secret*
profile.php	*Safe*	78%	*Low*	*Safe*

Confidence score menunjukkan tingkat keyakinan model terhadap hasil prediksi yang diberikan. Nilai ini diperoleh dari probabilitas kelas yang dihasilkan oleh algoritma *Random Forest*. Semakin tinggi *confidence score*, semakin tinggi tingkat keyakinan model terhadap kelas yang diprediksi. Meskipun model *Random Forest* mencapai akurasi rata rata sebesar 83%, beberapa kesalahan klasifikasi masih ditemukan pada data pengujian. Berdasarkan hasil *confusion matrix*, kesalahan prediksi umumnya terjadi pada kelas kerentanan yang memiliki kemiripan pola token maupun struktur sintaksis AST. Sebagai contoh, beberapa sampel *Command Injection* diprediksi sebagai File Inclusion karena keduanya sama-sama memanfaatkan input pengguna yang diteruskan ke fungsi berisiko tanpa proses validasi yang memadai. Selain itu, beberapa kasus *Unsafe File Upload* diprediksi sebagai Safe ketika kode mengandung mekanisme validasi parsial sehingga pola kerentanannya tidak cukup kuat untuk dibedakan oleh model.

Hasil Evaluasi Model Machine Learning

Evaluasi model *Random Forest* dilakukan menggunakan data testing sebanyak 64 sampel (20% dari total dataset 320 sampel) dengan pembagian kelas yang proporsional. Tabel 2 menampilkan hasil metrik evaluasi per kelas kerentanan.

Tabel 2. Hasil Evaluasi Model Random Forest

Kelas Kerentanan	Precision	Recall	F1-Score	Support
SQL *Injection*	0.89	0.87	0.88	11
XSS	0.85	0.83	0.84	10
File Inclusion	0.91	0.90	0.90	10
Command Injection	0.83	0.80	0.81	10
Unsafe File Upload	0.82	0.85	0.83	11
Safe	0.87	0.88	0.87	12
Macro Average	0.86	0.86	0.86	64

Berdasarkan Tabel 2, model *Random Forest* mencapai akurasi keseluruhan sebesar 86,25% pada data testing. Kelas File Inclusion memperoleh F1-*score* tertinggi sebesar 0,90 karena pola AST-nya (eval($_GET), *include*($_POST)) sangat distingtif. Kelas *Command Injection* memperoleh F1-*score* terendah sebesar 0,81, yang mengindikasikan adanya tumpang tindih fitur dengan kelas RCE. Model

menghasilkan *confidence score* yang lebih stabil dibandingkan penelitian sebelumnya [13], serta mencapai akurasi klasifikasi sebesar 86,25% dan *macro average* F1-*score* sebesar 0,86.

Analisis Confusion Matrix Model Random Forest

Berdasarkan *confusion matrix* pada Tabel 3, sebagian besar prediksi model berada pada diagonal utama, yang menunjukkan bahwa model mampu mengklasifikasikan mayoritas sampel dengan benar. Kelas *File Inclusion* menunjukkan performa terbaik dengan tingkat prediksi benar yang tinggi karena memiliki pola AST yang lebih spesifik dan mudah dikenali. Sebaliknya, kelas *Command Injection* dan *Unsafe File Upload* menunjukkan beberapa kesalahan klasifikasi karena memiliki karakteristik token yang relatif mirip dengan kelas lain seperti RCE dan *File Inclusion*. Hal ini menunjukkan bahwa overlap antar fitur masih menjadi tantangan dalam proses klasifikasi kerentanan berbasis kode sumber.

Tabel 3. Analisis Confusion Matrix Model Random Forest

Actual \ Predicted	SQLi	XSS	File Inc	Cmd Inj	Upload	Safe
SQL *Injection*	10	1	0	0	0	0
XSS	1	8	0	1	0	0
File Inclusion	0	0	9	1	0	0
Command Injection	0	1	1	8	0	0
Unsafe File Upload	0	0	0	1	9	1
Safe	0	0	0	0	1	11

Komparasi dengan Penelitian Terkait

Tabel 4 menyajikan perbandingan antara sistem yang diusulkan dengan penelitian terkait yang menggunakan pendekatan berbeda dalam mendeteksi kerentanan kode PHP. Perbandingan mencakup metode, jumlah kelas kerentanan, metrik akurasi, serta ketersediaan fitur rekomendasi perbaikan otomatis berbasis LLM.

Tabel 4. Komparasi dengan Penelitian Terkait

Penelitian	Metode	Kelas	F1 / Akurasi	Rekomendasi Otomatis	LLM
Habiby (2025) [13]	TF-IDF + AST + Random Forest	6	52-61%*	Tidak	Tidak
Bascara et al. (2022) [15]	GRU + GCN (DeepTective)	3	F1: 88.12%	Tidak	Tidak
Cetin et al. (2024) [14]	LLM Static Analysis	5	Bervariasi	Tidak	Sebagian
Penelitian ini (2025)	TF-IDF + AST + RF + Gemini API	10	F1:0.86/Acc:86.25%	Ya (Gemini)	Ya (Penuh)

**Confidence score berbasis skor aturan, bukan probabilitas model ML*

Pembahasan

Hasil pengujian menunjukkan bahwa perluasan pattern detection memberikan pengaruh signifikan terhadap kemampuan sistem dalam mengenali lebih banyak jenis kerentanan. Berbeda dengan versi sebelumnya [13] yang hanya mencakup enam kategori, versi lanjutan ini telah mencakup kategori tambahan seperti SSRF, CSRF, *Path Traversal, Hardcoded Secret, Weak Crypto*, dan *Insecure Deserialization*, sehingga total mencakup 10 kategori kerentanan.

Penambahan indikator aman juga membantu mengurangi potensi f*alse positive*. Sebagai contoh, *query* SQL yang menggunakan *prepare*() dan *bindParam*() akan mendapatkan pengurangan skor risiko SQL *Injection*. Begitu pula output yang menggunakan *htmlspecialchars*() akan menurunkan skor XSS. Dengan demikian, sistem tidak hanya melihat pola berbahaya, tetapi juga mempertimbangkan mitigasi yang telah diterapkan.

Lapisan rekomendasi berbasis *Google Gemini* API memberikan nilai tambah signifikan karena sistem tidak berhenti pada deteksi, tetapi juga menghasilkan saran perbaikan yang dinamis dan kontekstual sesuai dengan kode yang dianalisis [7]. Berbeda dengan pendekatan berbasis *template statis*, *Gemini* API mampu memahami konteks kode secara mendalam dan menyesuaikan rekomendasi dengan pola kerentanan spesifik yang ditemukan. Ketika sistem mendeteksi SQL *Injection*, *Gemini* menghasilkan rekomendasi berupa contoh kode perbaikan menggunakan prepared statement dan parameter binding yang langsung relevan. Pada XSS, Gemini menyarankan output encoding dengan *htmlspecialchars*() beserta penjelasan konteks penggunaannya dan rekomendasi *penerapan Content Security Policy*. Pemanfaatan LLM untuk perbaikan kode otomatis ini sejalan dengan penelitian Xia et al. [6] dan Wang et al. [12] yang menunjukkan bahwa LLM mampu menghasilkan rekomendasi perbaikan yang akurat ketika diberi konteks kerentanan yang terstruktur. Temuan ini juga mendukung hasil penelitian Pearce et al. [4] yang menunjukkan bahwa model generatif dapat membantu proses pengembangan perangkat lunak melalui pemberian saran dan rekomendasi kode, meskipun hasil yang dihasilkan tetap memerlukan validasi oleh pengembang untuk memastikan aspek keamanan dan keakuratannya.

Kelebihan Sistem

Keunggulan sistem yang dikembangkan antara lain mampu mendeteksi 10 kategori kerentanan dengan *confidence score* berbasis probabilitas model, memberikan rekomendasi perbaikan otomatis yang kontekstual via *Google Gemini* API, mencatat hasil ke dalam log, serta dapat dijalankan secara lokal tanpa *konfigurasi kompleks*. Sistem juga lebih informatif karena menampilkan pattern yang memicu deteksi, risk level, CWE ID, dan risiko tambahan yang ditemukan dalam satu *file*.

Keterbatasan Sistem

Walaupun sistem telah mengalami peningkatan signifikan, masih terdapat beberapa keterbatasan. Pertama, integrasi *Google Gemini* API memerlukan koneksi internet dan API key aktif, sehingga bergantung pada ketersediaan layanan eksternal. Kedua, AST Parsing masih berbasis pattern matching, belum menggunakan *parser formal* seperti nikic/PHP-*Parser* atau *Tree-sitter* yang dapat

menangani struktur kode lebih kompleks. Keterbatasan representasi fitur TF-IDF dan AST yang digunakan dalam penelitian ini juga telah diidentifikasi pada penelitian sebelumnya yang menunjukkan bahwa representasi kode menjadi faktor penting dalam keberhasilan deteksi kerentanan [11]. Ketiga, dataset sebesar 320 sampel masih terbatas untuk mewakili seluruh variasi pola kode PHP di dunia nyata. Keempat, sistem belum menunjukkan nomor baris kode spesifik yang rentan sehingga pengembang masih perlu melakukan penelusuran manual.

PENUTUP

Penelitian ini berhasil mengembangkan sistem cerdas deteksi dan rekomendasi perbaikan kerentanan kode PHP menggunakan pendekatan Hybrid ML dan LLM dengan *Google Gemini* API. Sistem menggabungkan analisis tekstual berbasis TF-IDF, analisis *struktural* berbasis AST *Parsing*, *pattern detection* berbobot, klasifikasi menggunakan Random Forest, serta lapisan rekomendasi dinamis berbasis *Google Gemini* API. Sistem ini menunjukkan peningkatan signifikan dibandingkan penelitian pendahulu [13] yang memperoleh *confidence score* 52-61%, dengan capaian akurasi keseluruhan 86,25% dan *macro average* F1-*score* sebesar 0,86 pada pengujian 64 sampel. Cakupan deteksi diperluas menjadi 10 kelas kerentanan dengan tambahan kategori *Path Traversal*, SSRF, CSRF, *Hardcoded Secret*, *Weak Crypto*, dan *Insecure Deserialization*.

Hasil pengujian menunjukkan bahwa sistem mampu memberikan deteksi awal yang lebih informatif melalui prediksi utama, *confidence score, risk level,* daftar *pattern* yang terdeteksi, risiko tambahan, serta rekomendasi perbaikan berbasis *Gemini* API yang kontekstual. Dengan demikian, Sistem ini berpotensi diterapkan sebagai *lightweight intelligent static code analyzer* untuk mendukung secure *software development lifecycle* pada aplikasi web berbasis PHP di lingkungan akademik maupun industri.

Penelitian selanjutnya dapat difokuskan pada peningkatan representasi kode, perluasan dataset yang lebih beragam, serta pengembangan kemampuan rekomendasi perbaikan yang lebih kontekstual. Selain itu, diperlukan evaluasi pada lingkungan pengembangan perangkat lunak yang lebih luas untuk mengukur kesiapan implementasi sistem pada penggunaan nyata.

DAFTAR PUSTAKA

[1] Y. Fang, S. Han, C. Huang, and R. Wu, "TAP: A Static Analysis Model for PHP Vulnerabilities Based on Token and Deep Learning Technology," PLOS ONE, vol. 14, no. 11, p. e0225196, Nov. 2019, doi: 10.1371/journal.pone.0225196.

[2] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing Web Applications Against XSS and SQLi Attacks Using a Novel Deep Learning Approach," Scientific Reports, vol. 14, no. 1, p. 1897, Jan. 2024, doi: 10.1038/s41598-023-48845-4.

[3] A. Senanayake, H. Kalutarage, and M. O. Al-Kadri, "Android Source Code Vulnerability Detection: A Systematic Literature Review," ACM Computing Surveys, vol. 55, no. 9, pp. 1-37, 2023, doi: 10.1145/3556974.

[4] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," in Proc. IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 754-768, doi: 10.1109/SP46214.2022.9833571.

[5] N. Shiri Harzevili, A. B. Belle, J. Wang, S. Wang, Z. M. Jiang, and N. Nagappan, "A Survey on Automated Software Vulnerability Detection Using Machine Learning and Deep Learning," ACM Computing Surveys, vol. 56, no. 9, pp. 1-39, 2024, doi: 10.1145/3652155.

[6] C. S. Xia, Y. Wei, and L. Zhang, "Automated Program Repair in the Era of Large Language Models," in Proc. IEEE/ACM 46th International Conference on Software Engineering (ICSE), Lisbon, Portugal, 2024, pp. 1-13, doi: 10.1145/3597503.3639187.

[7] D. Cao, Y. Liao, and X. Shang, "RealVul: Can We Detect Vulnerabilities in Web Applications with LLM?" in Proc. 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2024, pp. 1-13, doi: 10.1145/3691620.3695063.

[8] H. Yuan, Y. Tang, W. Sun, and L. Liu, "A Detection Method for Android Application Security Based on TF-IDF and Machine Learning," PLOS ONE, vol. 15, no. 9, p. e0238694, Sep. 2020, doi: 10.1371/journal.pone.0238694.

[9] X. Li, W. Wang, and X. Li, "Research on Intrusion Detection Based on an Enhanced Random Forest Algorithm," Applied Sciences, vol. 14, no. 2, p. 714, Jan. 2024, doi: 10.3390/app14020714.

[10] A. Alzahrani, "Structural Detection of Security Vulnerabilities in PHP Code Using AST and Graph-Based Techniques," Journal of Systems and Software, vol. 153, pp. 190-203, 2019, doi: 10.1016/j.jss.2019.03.064.

[11] A. Semasaba, W. Zheng, X. Wu, and S. Agyemang, "An Empirical Evaluation of Deep Learning-Based Source Code Vulnerability Detection: Representation Versus Models," Journal of Software: Evolution and Process, vol. 35, no. 6, p. e2422, Jan. 2023, doi: 10.1002/smr.2422.

[12] J. Wang, L. Cao, X. Luo, Z. Zhou, J. Xie, A. Jatowt, and Y. Cai, "Enhancing Large Language Models for Secure Code Generation: A Dataset-driven Study on Vulnerability Mitigation," in Proc. IEEE/ACM 46th International Conference on Software Engineering (ICSE), 2024, pp. 1-12, doi: 10.1145/3597503.3608134.

[13] M. E. Habiby, "Deteksi Kerentanan Kode PHP Menggunakan TF-IDF, AST Parsing, dan Random Forest," INTERNAL (Information System Journal), vol. 8, no. 1, pp. 1-9, Jun. 2025, doi: 10.32627/internal.v8i1.1411.

[14] O. Cetin, E. Ekmekcioglu, B. Arief, and J. Hernandez-Castro, "An Empirical Evaluation of Large Language Models in Static Code Analysis for PHP Vulnerability Detection," Journal of Universal Computer Science (JUCS), vol. 30, no. 9, pp. 1163-1183, Sep. 2024, doi: 10.3897/jucs.134739.

[15] M. Bascara, A. Fass, K. Rieck, and T. Holz, "DeepTective: A Hybrid Graph Neural Network Approach for Detecting PHP Vulnerabilities," in Proc. IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2022, pp. 1-10, doi: 10.1109/EuroSPW55150.2022.00033.