Evaluasi Kinerja Model Machine Learning dalam Cross-Project Defect Prediction Menggunakan Library PyCaret

Rian Hidayat[1], Agus Subekti[2]

[1,2]Program Studi Ilmu Komputer,
Fakultas Teknologi Informasi,
Universitas Nusa Mandiri, Jakarta
[1]14230027@nusamandiri.ac.id, [2]agus@nusamandiri.ac.id

Abstract

Software defect prediction (SDP) is an important approach to improving software quality by identifying modules that are prone to errors early on. However, traditional intra-project defect prediction (IPDP) approaches are often not applicable to new projects due to limitations in labeled training data. To address these limitations, Cross-Project Defect Prediction (CPDP) emerges as an alternative solution by leveraging data from source projects to build prediction models for target projects. Nevertheless, CPDP faces major challenges in the form of data distribution mismatch and class imbalance across projects, which can significantly reduce prediction accuracy. This study aims to evaluate the performance of several machine learning algorithms in the CPDP scenario and analyze the impact of hyperparameter tuning on model performance. Using the AEEEM *(Automated Software Engineering and Empirical Methods)* dataset consisting of five open-source Eclipse projects *Eclipse* Apache Lucene (LC), Equinox (EQ), Eclipse JDT Core (JDT), Plugin Development Environment atau Eclipse PDE UI (PDE), Mylyn (ML), experiments were conducted in a one-to-many CPDP scheme, where each project alternately acted as the source and target project. Random Forest (RF), *Category Boosting* (CatBoost), Logistic Regression (LR), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). Evaluations before and after the hyperparameter tuning process were conducted using the PyCaret library, with evaluation metrics including Accuracy, Area Under Curve (AUC), Recall, Precision, and F1-Score. The results show that RF consistently performs as the best model, outperforming in 9 out of 20 prediction combinations, both before and after tuning. CatBoost demonstrates stable performance and shows a significant improvement after tuning, while SVM, which initially performed the worst, shows the greatest improvement (6.39% increase in F1-Score), though it remains at the bottom. LR, on the other hand, shows a slight decline in performance after tuning. Further analysis revealed that combinations of projects with similar characteristics (such as LC → EQ, ML → EQ) produced high performance, while pairs with very different data distributions (such as EQ → ML, ML → LC) yielded low results, emphasizing the importance of domain similarity in CPDP. This study confirms that CPDP can be effective if the right models and project pairs are selected, and highlights the importance of hyperparameter optimization to improve model generalization. These findings provide practical guidance for researchers and practitioners in developing more reliable cross-project defect prediction systems, and open up avenues for further research such as the application of transfer learning, domain adaptation, and the integration of semantic or dynamic code features.

Keywords: Cross-Project Defect Prediction (CPDP), Hyperparameter Tuning, Machine Learning, PyCaret, Data Distribution, Random Forest

Received: 1 Juni 2025 Revised: 15 Agustus 2025 Accepted: 15 Agustus 2025

Abstrak

Prediksi cacat perangkat lunak (*Software Defect Prediction*/SDP) merupakan pendekatan penting untuk meningkatkan kualitas perangkat lunak dengan mengidentifikasi modul yang rentan terhadap kesalahan sejak dini. Namun, pendekatan tradisional berbasis *intra-project defect prediction* (IPDP) sering kali tidak dapat diterapkan pada proyek baru karena keterbatasan data pelatihan berlabel. Untuk mengatasi keterbatasan ini, *Cross-Project Defect Prediction* (CPDP) muncul sebagai solusi alternatif dengan memanfaatkan data dari proyek sumber untuk membangun model prediksi pada proyek target. Meskipun demikian, CPDP menghadapi tantangan utama berupa perbedaan distribusi data (data distribution mismatch) dan ketidakseimbangan kelas antar proyek, yang dapat menurunkan akurasi prediksi secara signifikan. Penelitian ini bertujuan untuk mengevaluasi kinerja beberapa algoritma *machine learning* dalam skenario CPDP, serta menganalisis dampak *hyperparameter tuning* terhadap performa model. Menggunakan dataset *Automated Software Engineering and Empirical Methods* (AEEEM) yang terdiri dari lima proyek *open-source* Eclipse Apache Lucene (LC), Equinox (EQ), Eclipse JDT Core (JDT), Plugin Development Environment atau Eclipse PDE UI (PDE), Mylyn (ML), eksperimen dilakukan dalam *skema one-to-many* CPDP, di mana setiap proyek secara bergantian berperan sebagai proyek sumber dan target. *Random Forest* (RF), *Category Boosting* (CatBoost), *Logistic Regression* (LR), *K-Nearest Neighbors* (KNN), dan *Support Vector Machine* (SVM). Evaluasi sebelum dan setelah proses *hyperparameter tuning* menggunakan *library pycaret*, dengan metrik evaluasi meliputi *Accuracy*, *Area Under Curve* (AUC), *Recall*, *Precision*, dan *F1-Score*. Hasil menunjukkan bahwa RF konsisten menjadi model terbaik, unggul dalam 9 dari 20 kombinasi prediksi, baik sebelum maupun setelah *tuning*. CatBoost menunjukkan kinerja stabil dan mengalami peningkatan signifikan setelah *tuning*, sementara SVM yang awalnya paling rendah menunjukkan perbaikan paling besar (6,39% peningkatan *F1-Score*), meskipun tetap berada pada posisi bawah. LR justru sedikit menurun performanya setelah *tuning*. Analisis lebih lanjut mengungkap bahwa kombinasi proyek dengan karakteristik serupa (seperti LC → EQ, ML → EQ) menghasilkan performa tinggi, sedangkan pasangan dengan distribusi data sangat berbeda (seperti EQ → ML, ML → LC) memberikan hasil rendah, menegaskan pentingnya kesamaan domain dalam CPDP. Studi ini menegaskan bahwa CPDP dapat efektif jika dipilih model dan pasangan proyek yang tepat, serta menyoroti pentingnya optimisasi hiperparameter untuk meningkatkan generalisasi model. Temuan ini memberikan panduan praktis bagi peneliti dan praktisi dalam mengembangkan sistem prediksi cacat lintas proyek yang lebih andal, serta membuka arah penelitian lanjutan seperti penerapan *transfer learning*, *domain adaptation*, dan integrasi fitur semantik atau dinamik kode.

Kata kunci: *Cross-Project Defect Prediction (CPDP), Hyperparameter Tuning*, *Machine Learning*, *PyCaret*, Distribusi Data, *Random Forest*

1. Pendahuluan

Peningkatan kualitas perangkat lunak merupakan salah satu aspek penting dalam siklus pengembangan sistem. Salah satu tantangan utama dalam hal ini adalah prediksi cacat *(defect prediction)* atau bug yang muncul selama proses pengembangan dan penggunaan aplikasi. Cacat perangkat lunak dapat menyebabkan gangguan fungsional, penurunan kualitas layanan pengguna *(user experience)*, serta meningkatnya biaya pemeliharaan [1]. Oleh karena itu, dibutuhkan pendekatan otomatis juga efektif dalam mengidentifikasi modul-modul yang rentan terhadap cacat sebelum tahap produksi [2]. Biaya pengujian perangkat lunak mencapai hampir separuh dari total biaya pengembangan, pengelolaan sumber daya yang terbatas menjadi sangat krusial. Dalam hal ini, penerapan model *Software Defect Prediction* (SDP) dapat membantu mengidentifikasi modul-modul yang paling berisiko mengalami cacat sebelum pengujian dimulai [3]. Dengan demikian, alokasi sumber daya untuk pengujian setiap modul dapat dioptimalkan.

Dalam beberapa tahun terakhir, penerapan teknik *machine learning* telah menunjukkan potensi yang signifikan dalam bidang prediksi SDP. Pendekatan berbasis model prediktif memungkinkan identifikasi modul yang bermasalah dengan memanfaatkan fitur-fitur kode seperti kompleksitas, ukuran file, dan jumlah revisi [4][5]. Namun, mayoritas pendekatan tersebut dilakukan secara *intra-project defect prediction* (IPDP), sehingga kurang efektif ketika diterapkan pada proyek baru dengan data pelatihan yang tidak tersedia.

Cross-Project Defect Prediction (CPDP) mengatasi keterbatasan data yang sering dihadapi dalam model SDP tradisional. Namun, akurasi prediksinya masih belum optimal dalam penerapan praktis karena adanya perbedaan distribusi data antara proyek sumber dan proyek target [6][7]. Selain itu, kinerja CPDP juga dipengaruhi oleh masalah dimensi tinggi pada fitur-fitur dalam dataset proyek sumber dan target, di mana beberapa fitur mungkin tidak relevan atau mengandung *outlier* [8][9].

Untuk mengatasi keterbatasan tersebut, penelitian ini mengusung pendekatan CPDD, di mana model pelatihan berasal dari proyek lain namun tetap memberikan performa prediksi yang baik. Dengan menggunakan dataset *Automated Software Engineering and Empirical Methods* (AEEEM) dan memanfaatkan *library pycaret* untuk mempercepat eksplorasi dan evaluasi model, penelitian ini bertujuan untuk mengembangkan model SDP yang dapat digunakan pada proyek yang berbeda dari proyek tempat data pelatihan dikumpulkan. Model klasifikasi dalam *machine learning* seperti *Random Forest* (RF), *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM), *Logistic Regression* (LR) dan *CatBoost Classifier* (CatBoost) merupakan model yang digunakan dalam penelitian ini yang mempunyai kemampuan baik dalam menangani klasifikasi dengan data tidak seimbang dan mampu memberikan hasil yang stabil [10][11].

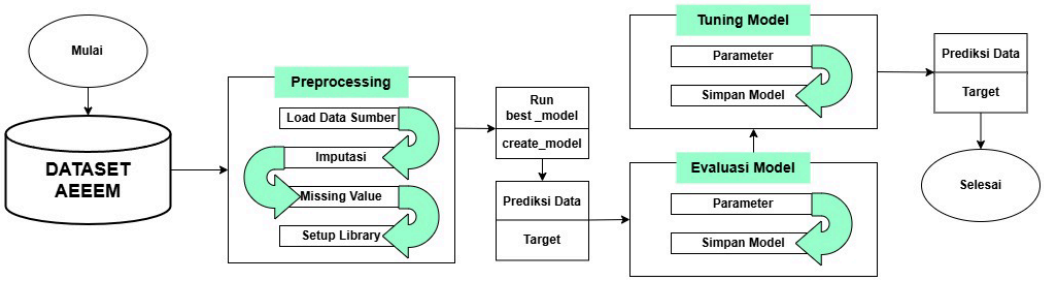
Penelitian ini juga menerapkan *library pycaret* mulai dari *preprocessing* hingga menemukan hasil, prediksi dilakukan dengan membandingkan data sebelum *tuning* dan sesudah *tuning*. Metodologi ini mencakup langkah-langkah analisis data yang komprehensif, penggunaan teknik-teknik *machine learning* atau kecerdasan buatan yang canggih, serta evaluasi terhadap berbagai parameter yang relevan. Selain itu, akan dilakukan juga validasi terhadap model klasifikasi yang dikembangkan untuk memastikan akurasi dan keandalannya dalam mengidentifikasi risiko SDP dengan tingkat kepercayaan yang tinggi [12].

Hasil dari penelitian ini dapat memberikan kontribusi yang signifikan dalam upaya prediksi defect atau non-defect pada perangkat lunak. Informasi yang diperoleh dari model klasifikasi yang dikembangkan dapat dimanfaatkan oleh para pengembang software untuk melakukan tindakan perbaikan yang tepat waktu dan tepat sasaran guna meningkatkan efisien waktu pembuatan perangkat lunak.

Selain itu, hasil penelitian ini juga diharapkan dapat menjadi langkah awal bagi pengembangan lebih lanjut terkait SDP secara *real-time* atau berkelanjutan, sehingga upaya dalam pencegahan dan perbaikan dapat dilakukan secara proaktif dan efisien. Dengan demikian, penelitian ini diharapkan dapat memberikan manfaat yang nyata dalam meningkatkan kualitas sebuah perangkat lunak.

2. Metode Penelitian

Diagram yang ditunjukan pada Gambar 1. Sebuah proses yang dilakukan pada penelitian melalui beberapa tahapan terdiri dari *preprocessing*, pembuatan model hingga pengujian model dengan proyek lain menggunakan *library pycaret*.



Gambar 1. Metode Penelitian

Metode yang diusulkan untuk prediksi cacat perangkat lunak pada dataset AEEEM yang terdiri dari lima proyek, diantaranya adalah EQ, ML, LC, JDT dan PDE. Pendekatan CPDP yang diterapkan merupakan *one to many*, artinya dari 5 proyek pada dataset AEEEM masing-masing saling melakukan testing. Contohnya proyek EQ akan dilakukan tes dengan proyek JDT, ML, LC serta PDE dan di ulang kembali proyek JDT akan dilakukan test dengan proyek EQ, ML, LC serta PDE dan terus sampai proyek akhir. Studi ini bertujuan untuk menilai performa berbagai model *machine learning* dalam melakukan prediksi cacat perangkat lunak. Evaluasi dilakukan menggunakan sejumlah metrik seperti *Accruacy*, AUC, *Recall*, Presisi, dan *F1-Score* untuk memastikan hasil prediksi yang tepat dan konsisten di berbagai proyek perangkat lunak [13]. Datasets dan code tersedia secara publik https://github.com/rianhidyt/cpdp-aeeem.

2.1. Dataset AEEEM

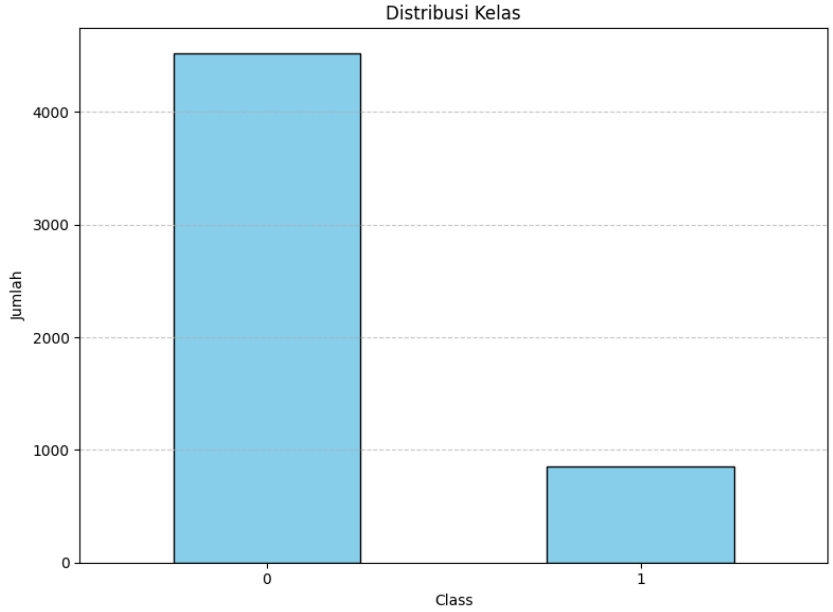
Dataset ini dikompilasi oleh Marco D'Ambros dan rekan-rekannya, diterbitkan dalam studi [6][14]. AEEEM berisi data dari 5 proyek perangkat lunak *open-source* yang dikembangkan dibawah naungan Eclipse Fondation dengan total 5.371 data, mencakup: *Apache Lucene* (LC) merupakan sebuah perpustakaan pencarian teks berbasis java, *Equinox* (EQ) salah satu komponen *framework eclipse* yang bertindak sebagai waktu ketika program sedang berjalan, *Eclipse JDT Core* (JDT) merupakan alat pengembangan java dalam *eclipse tools*, *Plugin Development Environment* (PDE) atau Eclipse PDE UI merupakan alat antarmuka *(User Interface)* pengguna untuk pengembangan *plugin eclipse* dan Mylyn (ML) adalah alat manajemen tugas dan produktivitas yang menghubungkan sistem pelacakan bug seperti bugzilla ke dalam IDE *Eclipse*. Format yang digunakan pada dataset AEEEM adalah *Attribute-Relation File Format* (ARFF), mendukung dengan alat pengolahan *machine learning* [14][15]. Atribut dalam setiap dataset mencakup 61 metrik diantaranya terdapat kolom *class* berisi label sebagai *defect* atau *non-defect* dengan jumlah persentase *defect* yang bervariasi antara proyek. Variasi jumlah entries data setiap proyek ditunjukan pada Tabel 1 dan Tabel 2 berikut.

Tabel 1. Rincian Dataset AEEEM yang digunakan

Dataset	Jumlah Modul	Jumlah Fitur	Defect	Non-Defect	Rasio Defect
LC	691	61	64	627	9.26%
EQ	324	61	129	195	39.81%
JDT	997	61	206	791	20.66%
PDE	1862	61	245	1617	13.16%
ML	1497	61	209	1288	13.96%

Tabel 2. Rincian Dataset AEEEM yang digunakan

Aspek	Deksripsi
Jumlah Dataset	5 (LC, EQ, JDT, PDE, ML)
Total Jumlah Data (*Entries*)	5.371
Jumlah Kolom (Fitur + Target)	62 (termasuk 61 fitur + 1 kolom target class)
Kolom Target	class (menunjukan status kecacatan modul)
Kode pada kolom *class*	b'0' = *Non=defect*, b'1' = *Defect*
Jumlah Modul Cacat (*Defect*)	853
Jumlah Modul Non-Cacat (*Non-Defect*)	4.518
Rasio Cacat	15.88% (853 / 5.371)
Tipe Data Utama	float64 (numerik)
Missing Value	diperiksa – tidak ditemukan atau minimal (asumsi data bersih)
Langkah Pra-Pemrosesan	1. Pengecekan struktur dataset (jenis data, entri, atribut, tipe data) 2. Pengecekan nilai hilang (missing value) 3. Normalisasi data untuk mengatasi ketidakseimbangan kelas
Tools Library	Python, PyCaret (untuk otomatisasi eksperimen dan pra-pemrosesan)
Tujuan Normalisasi	meningkatkan kinerja model dengan mengurangi dampak ketidakseimbangan data dan perbedaan skala fitur



Gambar 2. Distribusi atribut *class defect* dan *non-defect*

2.2. Preprocessing

Preprocessing dilakukan untuk memastikan data sudah bersih dari *missing value* dengan dilakukan imputasi data menggunakan *simpleimputer.* Selanjutnya dilakukan cek data kolom apakah ada yang hilang nilai (NaN) atau ada nilai yang belum numerik karena akan menggangu proses *modelling* menjadi kurang baik[16]. Setelah penanganan nilai selesai tahap berikutnya *setup library pycaret* dengan konfirgurasi pemilihan target kolom *class*, tipe target *binary*, *transformed* data dari 691 menjadi 1080 bertujuan meningkatkan kualitas data secara keseluruhan, pembagian data train dan test dengan proporsi 80/20%. Mengaktifkan fitur *fix imbalance* dengan metode *Synthetic Minority Over-sampling Technique* (SMOTE) dan

fitur *normalized* dengan metode *zscore* bertujuan menghilangkan duplikasi atau data yang *double*. Terakhir menerapkan *StratifiedKFold generator* untuk mendukung validasi silang meningkatkan generalisasi dan mencegah *overfitting* [17].

2.3. Run Model dan Create Model

Dalam *pycaret*, fungsi *create_model* dan *best_model* (melalui *compare_models*) memiliki peran penting dalam alur kerja pembangunan model *machine learning*. Fungsi create_model bertujuan untuk membuat dan melatih model spesifik, seperti RF, KNN, SVM, LR dan CatBoost menggunakan data yang telah disiapkan melalui *setup()*. Fungsi ini memungkinkan evaluasi awal performa model melalui validasi silang, seperti *StratifiedKFold*, dan memberikan fleksibilitas untuk eksperimen dengan *hyperparameter* atau skema validasi [18][19]. Di sisi lain, *best_model* yang dihasilkan dari *compare_models* bertujuan untuk membandingkan berbagai algoritma secara otomatis dan memilih model dengan performa terbaik berdasarkan metrik tertentu, seperti akurasi atau *F1-Score*. Model terbaik ini kemudian dapat digunakan untuk prediksi, *tuning* lebih lanjut, atau *deployment*. Keduanya saling melengkapi: *create_model* ideal untuk analisis mendalam pada satu model, sementara *best model* mempercepat proses pemilihan model optimal dalam waktu singkat [20].

2.4. Prediksi Data

Proses prediksi data dalam eksperimen CPDP dilakukan dengan memanfaatkan model yang dilatih pada data proyek sumber yang memiliki jumlah data pelatihan yang memadai untuk kemudian diterapkan pada proyek target. Pendekatan ini bertujuan untuk mengidentifikasi bagian kode atau modul dalam proyek target yang berpotensi mengandung cacat perangkat lunak, meskipun tidak ada data pelatihan lokal yang tersedia. Dengan demikian, CPDP memungkinkan prediksi cacat dilakukan secara efektif di proyek baru dengan memanfaatkan pengetahuan dari proyek lain yang sudah terdokumentasi dengan baik. Proses prediksi ini selanjutnya diikuti dengan evaluasi kinerja model menggunakan metrik seperti *F1-Score*, *Precision, Recall, Accuracy*, dan AUC, sehingga dapat memberikan gambaran menyeluruh mengenai kemampuan model dalam prediksi keberadaan cacat serta membantu tim pengembang dalam menentukan prioritas pengujian dan perbaikan.

2.5. Evaluasi Model

Proses evaluasi model dilakukan terhadap data uji dengan memanfaatkan sejumlah metrik kinerja, seperti *Accruacy*, AUC, *Recall, Precision*, dan *F1-Score*. Metrik-metrik tersebut digunakan untuk menilai kemampuan prediktif model dalam membedakan antara kelas cacat dan tidak cacat secara efektif [21]. Setiap metrik memberikan informasi spesifik mengenai aspek berbeda dari kinerja model: mulai dari kemampuan dalam prediksi kasus positif *Recall*, ketepatan hasil prediksi positif *Precision*, hingga keseluruhan keakuratan dan keseimbangan prediksi *F1-Score* dan *Accruacy*. Dengan kombinasi metrik ini, dapat diperoleh gambaran yang lebih komprehensif mengenai kemampuan model dalam mengidentifikasi pola-pola kerusakan atau cacat perangkat lunak serta efektivitasnya dalam melakukan klasifikasi pada data yang belum pernah diproses sebelumnya [22].

Evaluasi ini juga membantu mengidentifikasi kekuatan dan keterbatasan model dari sisi stabilitas prediksi dan tingkat keakuratan yang konsisten. Hasil evaluasi empiris terhadap model RF yang diuji pada berbagai dataset menunjukkan performa yang sangat baik, di mana model mampu mencapai keseimbangan yang optimal antara nilai Recall dan Precision. Hal ini menunjukkan bahwa model tidak hanya mampu prediksi sebagian besar kasus cacat dengan benar, tetapi juga minim dalam menghasilkan prediksi positif palsu.

Pendekatan ensemble learning yang menjadi dasar dari model ini memberikan kontribusi signifikan terhadap stabilitas dan ketahanan model terhadap variasi data. Dengan demikian, model tersebut menunjukkan kemampuan generalisasi yang kuat, sehingga layak diterapkan pada berbagai skenario dan dataset yang berbeda dalam prediksi cacat perangkat lunak.

2.6. Tuning Model

Proses penyetelan *tuning model* dapat diartikan sebagai peningkatan *hyperparameter* dilakukan menggunakan metode *grid search* untuk menemukan kombinasi optimal dari parameter seperti *max_depth*, *min_samples_split*, *n_estimators* dan *max_features* [23][24]. Tujuan utamanya adalah meningkatkan kinerja model dengan memilih nilai-nilai yang dapat memaksimalkan *accruacy* prediksi. Selain itu, penyesuaian ini membantu mencegah terjadinya *overfitting*, sehingga model tidak hanya baik dalam memprediksi data latih tetapi juga memiliki kemampuan generalisasi yang kuat terhadap data baru [25][26]. Dengan demikian, *hyperparameter tuning* menjadi langkah penting untuk meningkatkan keandalan dan efektivitas model dalam berbagai skenario prediksi.

3. Hasil dan Pembahasan

Setiap eksperimen yang dilakukan akan ada hasil yang didokumentasikan, begitu juga terjadi pada penelitian ini. Eksperimen dilakukan menggunakan *library pycaret* python, sebuah library yang dengan sedikit code memberikan hasil yang kompleks. Dataset AEEEM yang digunakan memberikan hasil yang variasi, proses hasil dibagi menjadi dua tahap yaitu sebelum *tuning* dan setelah *tuning*.

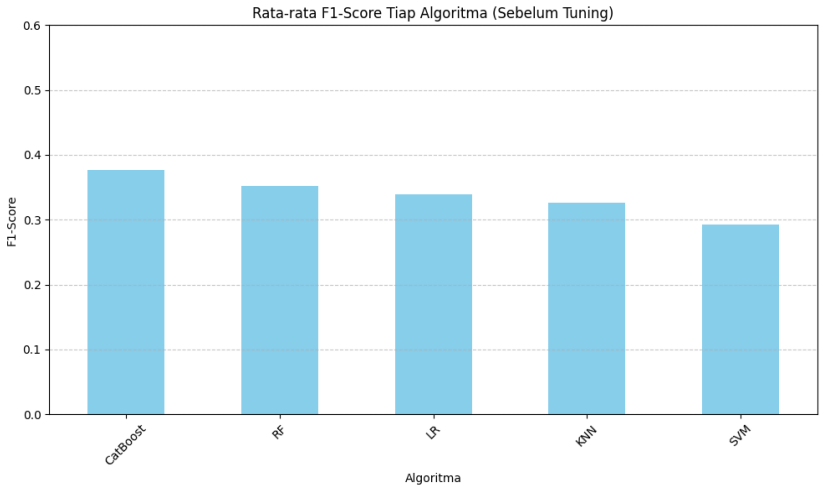
3.1. Hasil Pengujian Sebelum menggunakan *Hyperparameter Tuning*

Pada tahap evaluasi model pertama, dilakukan pengujian terhadap 5 algoritma klasifikasi pada skenario *Cross-Project Defect Prediction* (CPDP). Sebagaimana disajikan dalam Tabel 2.

Dari Tabel 2, hasil menunjukkan bahwa RF merupakan model dengan performa terbaik, unggul di 9 dari 20 kombinasi sumber–target. Model ini menunjukkan konsistensi tinggi dalam memprediksi cacat perangkat lunak antar proyek yang berbeda, seperti pada kombinasi EQ→JDT dan LC→EQ. CatBoost menjadi model kedua dengan kinerja stabil dan mendekati RF, berhasil unggul di 6 kombinasi. Meskipun tidak selalu menjadi yang terbaik, CatBoost tetap memberikan akurasi yang baik secara umum. LR juga menunjukkan kompetitivitasnya dengan mencatatkan akurasi tertinggi di 4 kombinasi, meskipun cenderung kalah dari RF dan CatBoost dalam sebagian besar kasus. Di sisi lain, KNN memiliki hasil yang bervariasi dan cenderung rendah di banyak pasangan proyek. Sementara itu, SVM secara konsisten menjadi model dengan performa terburuk tanpa sekali pun mencapai akurasi tertinggi dalam setiap kombinasi. Beberapa pasangan proyek seperti LC→EQ, ML→EQ, dan PDE→EQ menghasilkan akurasi tinggi untuk hampir semua model, menunjukkan bahwa prediksi lintas proyek dapat efektif jika ada kesamaan atau karakteristik yang relevan antar dataset. Namun, beberapa kombinasi seperti EQ → ML dan ML → LC memberikan hasil yang sangat rendah untuk semua model, menunjukkan tantangan utama dalam CPDP ketika proyek sumber dan target memiliki distribusi data yang sangat berbeda. Dengan demikian, dapat disimpulkan bahwa RF merupakan model paling andal sebelum proses *tuning* dilakukan, diikuti oleh CatBoost dan LR, sedangkan KNN dan SVM kurang optimal dalam skenario CPDP ini.

Tabel 2. Data Hasil Prediksi Sebelum menggunakan *Hyperparameter Tuning* Berdasarkan *F1 Score*

Source	Target	KNN	SVM	RF	LR	CatBoost
EQ	JDT	0.3803	0.3605	**0.4166**	0.3794	0.4069
EQ	LC	0.2241	0.1812	**0.2850**	0.2016	0.2582
EQ	ML	0.2290	0.2209	0.2251	0.2165	0.2236
EQ	PDE	0.2565	0.2734	**0.2777**	0.2753	**0.2777**
JDT	EQ	0.3787	0.2930	0.3038	**0.4000**	0.3478
JDT	LC	0.2369	0.3378	**0.3956**	0.3824	0.3883
JDT	ML	0.2927	0.3162	0.2462	**0.3458**	0.2663
JDT	PDE	0.2903	0.3493	0.3323	**0.3484**	0.3256
LC	EQ	**0.5198**	0.4563	0.4333	0.4365	0.4023
LC	JDT	0.3818	0.2739	0.5026	0.3527	**0.5610**
LC	ML	0.2736	0.0145	0.3048	0.2078	**0.3475**
LC	PDE	0.2759	0.1509	0.3122	0.2702	**0.3158**
ML	EQ	**0.4696**	0.3256	0.4176	0.4314	0.4333
ML	JDT	0.2843	0.4126	0.3691	0.3922	**0.4396**
ML	LC	**0.3153**	0.2222	0.2056	0.1167	0.2955
ML	PDE	0.3227	0.2773	0.3937	0.3284	**0.3940**
PDE	EQ	0.4591	0.6622	0.4393	**0.6931**	0.4681
PDE	JDT	0.4154	0.4476	0.4697	0.4090	**0.5239**
PDE	LC	0.2973	0.2075	0.3762	0.3420	**0.3956**
PDE	ML	0.2247	0.0728	0.2864	0.2439	**0.3357**



Gambar 3. Algoritma sebelum menggunakan *Tuning*

Dari Gambar 3 histogram di atas, dapat dilihat rata-rata *F1-Score* algoritma sebelum *tuning* dalam konteks eksperimen CPDP, dengan *F1-Score* sebagai metrik evaluasi. Sumbu horizontal mencantumkan 5 algoritma yang diuji, yaitu CatBoost, RF, LR, KNN dan SVM, sementara sumbu vertikal menunjukkan *F1-Score* dari 0 hingga 0.6. Hasilnya, CatBoost mencatat *F1-Score* tertinggi mendekati 0.4, diikuti RF dengan skor serupa, kemudian LR dan KNN sekitar 0.35, serta SVM dengan skor terendah di atas 0.3. Grafik ini, yang kemungkinan dihasilkan menggunakan PyCaret melalui fungsi *compare_models*, menunjukkan bahwa CatBoost dan RF memiliki performa awal terbaik untuk memprediksi cacat pada proyek target, meskipun perbedaan skor tidak terlalu besar, mengindikasikan tantangan generalisasi akibat perbedaan distribusi data antara proyek sumber dan target. Hasil ini dapat menjadi dasar untuk memilih model terbaik guna optimasi lebih lanjut atau penerapan pada proyek target.

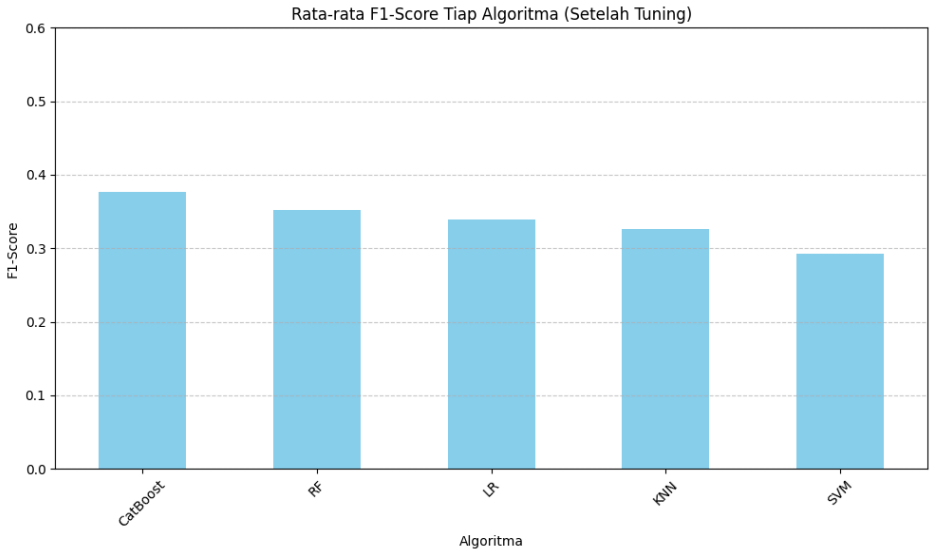
3.2. Hasil Pengujian Setelah Menggunakan *Hyperparamer Tuning*

Pada tahap evaluasi model kedua, dilakukan pengujian terhadap 5 algoritma klasifikasi pada skenario CPDP. Sebagaimana disajikan dalam Tabel 3.

Tabel 3. Data Hasil Prediksi Setelah menggunakan *Hyperparameter Tuning* Berdasarkan *F1 Score*

Source	Target	KNN	SVM	RF	LR	CatBoost
EQ	JDT	0.3904	0.3967	**0.4263**	0.3794	0.4228
EQ	LC	0.2295	0.2286	**0.3174**	0.2016	0.2311
EQ	ML	0.2159	0.2127	**0.2265**	0.2165	0.2061
EQ	PDE	0.2760	0.2642	**0.3009**	0.2753	0.2633
JDT	EQ	0.3312	**0.3977**	0.3038	0.3593	0.3478
JDT	LC	**0.4094**	0.3820	0.3956	0.4000	0.3883
JDT	ML	0.3047	0.2688	0.2462	**0.3213**	0.2663
JDT	PDE	0.2984	0.3955	0.3323	**0.4021**	0.3256
LC	EQ	**0.6134**	0.4974	0.6017	0.4653	0.5652
LC	JDT	0.4762	**0.5224**	0.5120	0.3297	0.5168
LC	ML	0.2234	0.0215	0.3287	0.2039	**0.3456**
LC	PDE	**0.3546**	0.1912	0.3532	0.2331	0.3501
ML	EQ	0.4726	**0.5776**	0.5752	0.4314	0.4737
ML	JDT	0.2972	0.4944	**0.5300**	0.3922	0.5274
ML	LC	0.3318	0.2872	0.3256	0.1167	**0.3333**
ML	PDE	0.3188	0.3716	0.3652	0.3284	**0.4040**
PDE	EQ	0.5185	0.4947	0.5913	**0.6931**	0.6349
PDE	JDT	**0.5741**	0.5448	0.5276	0.4090	0.4850
PDE	LC	0.3494	0.3776	0.3377	0.3420	**0.3529**
PDE	ML	**0.3306**	0.3058	0.3298	0.2439	0.3081

Proses *hyperparameter tuning* dilakukan, hasil menunjukan bahwa RF tetap menjadi model dengan kinerja terbaik secara konsisten, unggul pada 9 dari total kombinasi prediksi. Diikuti oleh CatBoost yang menunjukkan peningkatan signifikan dan berhasil menjadi model terbaik pada 4 kombinasi. KNN dan SVM juga mengalami peningkatan performa, masing-masing unggul di 3 dan 2 kombinasi. Sementara itu, LR mencatatkan hasil terbaik pada 2 kombinasi. Hasil ini menunjukkan bahwa meskipun RF tetap dominan, proses *tuning* memberikan dampak positif yang cukup berarti bagi CatBoost, KNN, dan SVM dalam beberapa skenario prediksi.



Gambar 4. Algoritma Setelah di *Tuning*

Dari Gambar 4 di atas, menunjukkan rata-rata *F1-Score* dari beberapa algoritma *machine learning* setelah melalui proses *hyperparameter tuning*. Sumbu X menampilkan nama-nama algoritma seperti CatBoost, RF, LR, KNN dan SVM, sedangkan sumbu Y menunjukkan nilai *F1-Score* sebagai metrik evaluasi. Nilai *F1-Score* berkisar antara 0 hingga 1, dengan semakin tinggi nilainya menandakan performa model yang semakin baik. Berdasarkan grafik tersebut, CatBoost memiliki performa terbaik dengan *F1-Score* sekitar 37%, diikuti oleh RF 35%, LR 33%, KNN 0.32%, dan SVM dengan nilai terendah sekitar 0.30. Meskipun selisih antar algoritma tidak terlalu besar, hasil ini menunjukkan bahwa CatBoost lebih efektif dalam menangani masalah prediksi yang dianalisis, meskipun algoritma lain juga masih memiliki potensi untuk digunakan tergantung pada konteks dan kebutuhan spesifiknya.

3.3. Perbandingan Hasil Prediksi Sebelum dan Sesudah Lakukan *Hyperparameter Tuning*

Berdasarkan hasil prediksi, baik dengan optimasi *hyperparameter tuning* maupun tidak, dapat disimpulkan dengan Tabel 4 di bawah ini:

Tabel 4. Perbandingan Hasil Prediksi Sebelum dan Sesudah *Tuning*

Models	Sebelum *Tuning*	Setelah *Tuning*	Selisih
KNN	0.3264	0.3658	0.0394
SVM	0.2928	0.3567	0.0639
RF	0.3520	**0.4033**	0.0514
LR	0.3393	0.3337	(-) 0.0056
CatBoost	**0.3762**	0.3902	0.0140

Berdasarkan Tabel 4 di atas, memperlihatkan perbandingan performa beberapa model *machine learning* sebelum dan setelah proses *hyperparameter tuning*, dengan menggunakan *F1-Score* sebagai metrik evaluasi. Model-model yang dievaluasi meliputi KNN, SVM, RF, LR, dan CatBoost. Terlihat bahwa *tuning* secara umum berhasil meningkatkan performa keempat model, kecuali pada LR yang justru mengalami sedikit penurunan sebelum *tuning*, CatBoost memiliki performa tertinggi dengan *F1-Score* 37,62% namun setelah *tuning*, RF menjadi model terbaik dengan *F1-Score* mencapai 40,33%. Peningkatan paling signifikan terjadi pada model SVM sebesar 06,39% diikuti oleh KNN sebesar 03,94%.

Meskipun CatBoost tetap menunjukkan hasil yang baik, peningkatannya lebih kecil dibandingkan model lainnya. Hasil ini menunjukkan bahwa proses *hyperparameter tuning* efektif untuk meningkatkan akurasi model, meski tidak selalu berdampak positif bagi semua algoritma.

4. Kesimpulan

Pendekatan ini bertujuan mengatasi keterbatasan data pelatihan dalam skenario prediksi intra-proyek, serta membantu pengembang perangkat lunak dalam mengidentifikasi modul-modul berisiko tinggi secara lebih efisien. Dengan menggunakan dataset AEEEM yang terdiri dari 5 proyek open-source (EQ, ML, LC, JDT, PDE), penelitian ini membandingkan performa 5 algoritma *machine learning*, yaitu Random RF, CatBoost, LR, KNN, dan SVM, baik sebelum maupun setelah proses hyperparameter *tuning*. Proses eksperimen dilakukan dalam skema *one-to-many* CPDP, di mana setiap proyek digunakan sebagai sumber dan target secara bergantian. Diketahui bahwa sebelum dilakukan *tuning hyperparameter*, RF menunjukkan performa terbaik dengan unggul di 9 dari 20 kombinasi prediksi, diikuti oleh CatBoost yang memiliki hasil stabil dan mendekati RF dengan 6 kombinasi terbaik. Logistic Regression (LR) masih kompetitif dalam beberapa kasus dengan 4 kombinasi terbaik, sementara KNN menunjukkan hasil yang bervariasi dan cenderung rendah. Adapun SVM, secara konsisten menjadi model dengan performa terburuk tanpa pernah menjadi yang terbaik. Setelah proses *tuning* hyperparameter dilakukan, posisi RF tetap tak tergoyahkan sebagai model terbaik dengan jumlah kombinasi terbaik yang sama, yaitu 9. CatBoost mengalami peningkatan signifikan dengan bertambahnya kombinasi terbaik menjadi 4. KNN dan SVM juga menunjukkan perbaikan dengan masing-masing unggul di 3 dan 2 kombinasi, sedangkan LR justru sedikit menurun dengan hanya unggul di 2 kombinasi. Secara keseluruhan, *tuning* hyperparameter memberikan dampak positif bagi semua model kecuali LR, dengan peningkatan paling besar terjadi pada SVM sebesar 6,39% berdasarkan rata-rata *F1-Score*, diikuti oleh RF sebesar 5,14%, dan KNN sebesar 3,94%. Meskipun CatBoost tetap berada di posisi atas, peningkatannya relatif kecil, yaitu 1,4%. Dalam analisis kombinasi sumber–target, beberapa pasangan proyek seperti LC → EQ, ML → EQ, dan PDE → EQ menunjukkan performa tinggi untuk hampir semua model, menandakan bahwa CPDP dapat efektif apabila terdapat kesamaan karakteristik antar proyek. Sebaliknya, kombinasi seperti EQ → ML dan ML → LC memberikan hasil yang rendah untuk semua model, menunjukkan adanya tantangan besar ketika distribusi data antar proyek sangat berbeda. oleh sebab itu, penelitian selanjutnya dapat mempertimbangkan penerapan teknik *transfer learning* atau *domain adaptation* seperti *feature alignment, adversarial training,* atau *instance weighting* untuk mengurangi heterogenitas antar domain proyek. Terakhir, integrasi metrik tidak tradisional seperti *code churn, developer activity,* atau *change history* sebagai fitur tambahan dapat meningkatkan representasi modul dan mendukung prediksi yang lebih akurat dalam skenario CPDP.

Daftar Pustaka

[1] K. Shebl, Y. Afify, and N. Badr, "Software defect prediction approaches revisited," *Int. J. Intell. Comput. Inf. Sci.*, vol. 23, no. 3, 2023, doi: 10.21608/ijicis.2023.200737.1261.

[2] N. A. G. Gayatri, H. Soeparno, F. L. Gaol, and Y. Arifin, "Enhancing Software Quality Through Defect Prediction," *2024 3rd International Conference on Creative Communication and Innovative Technology (ICCIT)*, pp. 1–7, 2024, doi: 10.1109/iccit62134.2024.10701182.

[3] S. Abbas, S. Aftab, M. A. Khan, T. M. Ghazal, H. Al Hamadi, and C. Y. Yeun, "Data and Ensemble Machine Learning Fusion Based Intelligent Software Defect Prediction System," *C. Mater. Contin.*, vol. 75, no. 3, pp. 6083–6100, 2023, doi: 10.32604/cmc.2023.037933.

[4] G. Cauvery and D. DhinaSuresh, "Software Defect Prediction Using Machine Learning Techniques," *Data Anal. Artif. Intell.*, vol. 3, no. 2, pp. 30–33, 2023, doi: https://dx.doi.org/10.46632/daai/3/2/7

[5] E. H. A. Prastyo, M. A. Yaqin, S. Suhartono, M. Faisal, and R. A. J. Firdaus, "Naive Bayes Classification for Software Defect Prediction," Transactions on Informatics and Data Science, vol. 1, no. 1, pp. 11–20, 2024, doi: 10.24090/tids.v1i1.12192.

[6] K. Javed, S. Ren, M. Asim, and M. A. Wani, "Cross-Project Defect Prediction Based on Domain Adaptation and LSTM Optimization," *Algorithms*, vol. 17, no. 5, 2024, doi: 10.3390/a17050175.

[7] A. Taskeen, S. U. R. Khan, and E. A. Felix, "A research landscape on software defect prediction," *J. Softw.*, vol. 35, no. 12, 2023, doi: 10.1002/smr.2549.

[8] Z. Li, J. Niu, X.-Y. Jing, W. Yu, and C. Qi, "Cross-Project Defect Prediction via Landmark Selection-Based Kernelized Discriminant Subspace Alignment," *IEEE Trans. Reliab.*, vol. 70, no. 3, pp. 996–1013, 2021, doi: 10.1109/TR.2021.3074660.

[9] S. Jiang, J. Zhang, O. Teng, and J. Li, "Balanced Adversarial Tight Matching for Cross-Project Defect Prediction," *IET Softw.*, 2024, doi: 10.1049/2024/1561351.

[10] Y. Z. Bala, P. Abdul Samat, K. Y. Sharif, and N. Manshor, "Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach," *IEEE Access*, vol. 11, no. January, pp. 2318–2326, 2023, doi: 10.1109/ACCESS.2022.3231456.

[11] A. Hadianto and W. H. Utomo, "CatBoost Optimization Using Recursive Feature Elimination," Jurnal Online Informatika, vol. 9, no. 2, 2024, doi: 10.15575/join.v9i2.1324.

[12] T. Inoue, N. Hanafusa, Y. Kawaguchi, and K. Tsuchiya, "Predicting anemia management in dialysis patients using open-source machine learning libraries," *Ren. Replace. Ther.*, vol. 11, no. 1, p. 47, Jun. 2025, doi: 10.1186/s41100-025-00633-8.

[13] Y. Z. Bala, P. A. Samat, K. Y. Sharif, and N. Manshor, "Cross-project software defect prediction through multiple learning," *Bul. Tek. Elektro dan Inform.*, vol. 13, no. 3, 2024, doi: 10.11591/eei.v13i3.5258.

[14] Y. Zhao, Y. Zhu, Q. Yu, and X. Chen, "Cross-Project Defect Prediction Method Based on Manifold Feature Transformation," *Futur. Internet*, vol. 13, no. 8, p. 216, 2021, doi: 10.3390/FI13080216.

[15] M. J. Hernandez-Molinos, A. J. Sanchez-Garcia, R. E. Barrientos-Martínez, J. C. Perez-Arriaga, and J. O. Ocharán-Hernández, "Software Defect Prediction with Bayesian Approaches," *Mathematics*, vol. 11, no. 11, p. 2524, 2023, doi: 10.3390/math11112524.

[16] Y. Wang, Y. Li, H. Wang, Z. Lei, and X. Zhang, "Better Knowledge Enhancement for Privacy-Preserving Cross-Project Defect Prediction," 2024, doi: 10.48550/arxiv.2412.17317.

[17] K. Arai, J. Shimazoe, and M. Oda, "Method for Hyperparameter Tuning of Image Classification with PyCaret," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 9, p. 7, 2023, doi: 10.14569/ijacsa.2023.0140930.

[18] F. Handayanna and S. Sunarti, "Penerapan Algoritma K-Means Untuk Mengelompokkan Kepadatan Penduduk di Provinsi DKI Jakarta," *J. Appl. Comput. Sci. Technol.*, vol. 5, no. 1, pp. 50–55, Mar. 2024, doi: 10.52158/jacost.v5i1.477.

[19] A. Kumar, S. Kumar, A. Kumari, and A. Saini, "Edge Computing based IDS Detecting Threats using Machine Learning and PyCaret," *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, 2023, pp. 668–673. doi: 10.1109/CISES58720.2023.10183591.

[20] A. T. Olanipekun, "An Improved Prediction of Transparent Conductor Formation Energy using PyCaret: An Open-Source Machine Learning Library," *J. Appl. Data Sci.*, vol. 5, no. 4, pp. 1914–1924, 2024, doi: 10.47738/jads.v5i4.202.

[21] P. S. Vadar, T. T. Moharekar, and U. R. Pol, "Comparative analysis of automated machine learning libraries: pycaret, h2o, tpot, auto-sklearn, and flaml," *Int. J. Sci. Res. Eng. Manag.*, vol. 08, no. 11, pp. 1–8, Nov. 2024, doi: https://doi.org/10.55041/IJSREM39119.

[22] K. Gopalakrishnan, "Leveraging PyCaret for Time Series Analysis-A Low Code Approach," *Journal of Artificial Intelligence & Cloud Computing*, vol. 2, no. 3, pp. 1–4, 2023, doi: 10.47363/jaicc/2023(2)314.

[23] Y. Kim, Y.-C. Byun, and S.-J. Lee, "A Study on Sugar Content Improvement and Distribution Flow Response through Citrus Sugar Content Prediction Based on the PyCaret Library," *Horticulturae*, vol. 10, no. 6, p. 630, 2024, doi: 10.3390/horticulturae10060630.

[24] S. Gul, K. Ayturan, and F. Hardalaç, "PyCaret for Predicting Type 2 Diabetes: A Phenotype- and Gender-Based Approach with the "Nurses' Health Study" and the "Health Professionals' Follow-Up Study" Datasets," *J. Pers. Med.*, vol. 14, no. 8, p. 804, 2024, doi: 10.3390/jpm14080804.

[25] A. Munjal, R. Khandia, and B. Gautam, "A Machine learning Approach for selection of polycystic ovarian syndrome (PCOS) attributes and comparing different Classifier performance with the help of WEKA and PyCaret," *Int. J. Sci. Res.*, pp. 59–63, 2020, doi: 10.36106/IJSR/5416514.

[26] A. P. and R. Gunasundari, "An Interpretable PyCaret Approach for Alzheimer's Disease Prediction," *Int. J. Comput. Exp. Sci. Eng.*, vol. 10, no. 4, 2024, doi: 10.22399/ijcesen.655.

This Page Intentionally Left Blank