Impact Factor 6.1

Journal of Cyber Security

ISSN:2096-1146

Scopus

DOI

Google Scholar

More Information
www.journalcybersecurity.com

Crossref doi Google Scholar scopus

An Integrated Static and Dynamic Analysis Framework for Android Application Maintainability and Sustainability Assessment

Zena Nabil Faysal [1] and Shayma Mustafa Mohi-Aldeen [2]

Department of Computer Science and Mathematics, Mosul University,

Abstract

The paper describes the design and development of ASDA, a comprehensive analysis framework for Android applications, which employs code analysis techniques, machine learning, and performance analysis at run time to offer end-to-end results on code quality and maintainability. The maintainability debts of applications are analyzed at design time by extracting metrics from Java code using the CK metrics tool to build machine learning models for predicting the Maintainability Debt Index (MDI) for classes. Four regression analysis techniques, namely Random Forest, Gradient Boosting, Extra Trees, and Voting Ensemble, are used to identify the best-suited regression model for maintainability prediction, which is Random Forest (RMSE = 0.082, R^2 = 0.764). The results of dynamic analysis techniques are obtained from Perfetto traces and used to measure CPU, context switches, method performance times, and estimated energy consumption. The performance results are incorporated into the design of the maintainability and sustainability debt metrics framework for run-time analysis of applications, which include aspects of execution, energy, responsiveness, and concurrency. The results clearly show direct correlations between the predicted and actual energy consumption of applications (R^2 = 0.93), thereby confirming the effectiveness of the framework. The combination of design-time and run-time analyses helps to identify classes for which maintainability and resource intensity are desirable goals and enables prioritizing of changes to Android applications. The results of the design and development of ASDA clearly illustrate its utility in Android applications as a comprehensive analysis framework that reduces maintainability and sustainability debt and reduces energy consumption in applications through tool-driven changes by up to 89.2% in maintainability and 90.3% in energy consumption in applications for redeveloping them as environmentally sound and resource-efficient applications. The design and development of ASDA clearly illustrate its utility as a comprehensive framework for Android applications at design time that provides end-to-end results on maintainability and resource efficiency of applications for developing resource-efficient and environmentally sound applications. The design and development of ASDA clearly illustrate its utility as a comprehensive framework at design time that provides end-to-end results on maintainability and resource efficiency of applications for developing resource-efficient and environmentally sound applications.

Keywords

Android Applications, Maintainability, Machine Learning, Static Analysis, Dynamic Analysis, Energy Efficiency

1. Introduction

In current times, sustainability within software development has emerged into the spotlight. Individuals are opening their eyes to the way a software application consumes energy resources. They are realizing the way resources are utilized in software development. A software application's maintainability has also started gaining attention. In the past, software development has mainly emphasized the need for software functions or the need for software applications to be faster and faster along with improved performance [1]. It mainly favors a particular goal: faster software. It might miss the bigger picture. Software design today might create a trail of sustainability debt. It might mainly create a trail that consists of code structure complexity, energy-consuming processes in software, software documentation quality, software dependency management processes that are clumsy, and the maintainability of software [2].

As apps become more complex, with finer-grained UI details, more background activity, and more connections to libraries, the process of identifying and remediating sustainability debt becomes more challenging. Static code analysis understands the structure very well, but fails to capture how everything actually executes in the wild, and this has a direct impact on power consumption and user experience [3]. At the other extreme, dynamic analysis provides piquant, dynamic views of what the app does, but fails to point out shortcomings in maintainability or provide actionable advice. The lesson here is the desire for a 360-degree model that marries the strengths of both approaches and the application of machine learning for prediction of code sustainability, not merely maintainability but the broad subject of code sustainability in general [4].

The term sustainability debt has been created as the analog of the long-standing concept of technical debt within software engineering. Technical debt refers to the high cost of having expediency in decision-making by the programmers; it is written quick, but suboptimal code, or postponed documentation or not testing properly, to achieve short-term objectives [5]. Although these decisions are able to advance in the short term, they create liabilities that prevent further development in the maintenance, scalability, and quality. Sustainability debt elaborates on this metaphor, however, it enlarges the scope to more than technical issues since it acknowledges that software projects come at expenses to the long term economic, social, and environmental systems as well [6].

Fundamentally, sustainability debt is the adverse outcomes of a decision that undercuts the sustainability of a software system in various aspects. Sustainability debt occurs when a project is developed in ways that do not take into account the effect on sustainability, or where an architectural decision or management decision is drawn in the manner that does not consider its effect on sustainability. Indicatively, an application of energy-consuming algorithms can make the early development straightforward but introduce some unknown expenses in the form of increased energy use and carbon footprint during implementation [7]. Equally, ignoring the notions of accessibility or inclusiveness in design can minimize the amount of effort needed to develop the system in the short term, but create social inequities and usability problems that weaken the adoption and acceptance of the system [8].

The accumulation of the sustainability debt can be hidden and difficult to measure. The impact of sustainability debt is not immediate at all as compared to functional defects which normally take the form of instant failures. In the example given, the shortcuts, as drawn by the technical team appear harmless, but will eventually result in complex maintenance, stagnation development and increment in cost. To the same extent, the environment can not affect the system directly, but with more use, the energy cost as well as the cost of carbon rises at an unsustainable rate. This hidden character makes sustainability debt particularly dangerous because the organizations can be handed over to ignorance with the growing indebtedness before it becomes excessive [9].

The metaphor of debt also competes on the interest, which is to be paid. Whenever any project is left with a sustainability debt, their compromise will increase the costs of recovery. Technical interest is regarded as an augmented intricacy in the refactoring procedure and Economic interest is observed as augmented maintenance populace. When the energy cost grows and both the regulatory forces exert pressure and the social interest arises in forms of distrusted shapes, the environment interest will take place and the team ineffectiveness or dissatisfaction will be maintained among the users. These debts could put the sustainability of the software system at a risk in the long term and lower its contribution to increased sustainability goals [6].

Machine learning has demonstrated its aptness at finding the complicated patterns that are often inherent to high-dimensional data and has naturally found application within the area of forecasting the sustainability of software [10]. By leveraging both actual and simulation data related to projects, machine learning algorithms are able to identify complicated patterns associated with sustainability debt that are contingent on a variety of factors associated with projects, including code complexity and the number of dependencies. They can even provide early alerts about the regions of a project that are most at risk of high levels of debt accumulation [11].

A major challenge to ML applications related to sustainability evaluation is the unavailability of datasets with detailed static and dynamic characteristics of a given project. This data is usually missing within publicly accessible datasets, including aspects of energy consumption, dependency status, and CI/CD robustness. The challenge is, therefore, compensated for within this research, where artificially produced datasets are incorporated, ensuring they represent data possible within a typical Android project [12]. This created data includes varied aspects, which include code size, cyclomatic complexity, number of methods and fields, and other indicators of class cohesion and coupling, documentation and test coverage, CI/CD robustness, and energy consumption, and Gaussian noise is incorporated to represent realistic levels, and standardization is conducted to have equal scales for machine learning analysis [13].

The framework investigates a range of machine-learning models, starting from conventional approaches to ensemble models and boosting models. Interpretable models, including logistic regression and Gaussian Naive Bayes, clarify the significance of variables, while non-linear association patterns are addressed using K-Nearest Neighbors and Support Vector Machines [14] [15]. Ensemble models, including Random Forest and Extra Trees, enhance stability, reduce variance, and allow for the evaluation of feature significance [16]. Boosting models, including Gradient Boosting Machines and AdaBoost, mitigate the residuals of the model in an incremental process, improving accuracy. Using the above framework, the models can now be compared in a systematic manner to determine the most accurate predictor of sustainability debt [17].

Analysis Perspective, this Architectural Sustainability Analysis Framework combines static and dynamic analysis in one platform [18]. In static analysis, this Composer Sustainability Analysis Framework applies the Chidamber & Kemerer set of metrics to identify at least five important class and method-related metrics: coupling (CBO), class cohesion (LCOM and TCC), depth of inheritance (DIT), number of methods (NOM), and complexity (WMC and NBD) [19]. Metrics are linked to code maintainability and design flaws. As for the dynamic analysis component, the Perfetto, an Android tracing framework, offers data regarding CPU consumption, threading, context switches, wake-ups, and expected energy consumption [17].

The tool is designed with usability as the focus, and the output is reportable results you can do something with. It includes an interactive Tkinter GUI where you can dissect results down to the class and function level, and also spit out results to report the ML predictions with understandable AI, and generate reports in the form of PDF and CSV. And the aim is to take the dirty results of static analysis, dynamic profiling, and machine learning results and display these to the software engineer and project leader in a comprehensible fashion. Through the integration of static code analysis, runtime profiling, and ML predictions, this work introduces a novel approach towards the maintainability and sustainability analysis of Android applications. The tool has the ability to reveal high-debt regions, prioritize optimization activities, and reduce technical as well as environmental expenditures related to software development.

This paper proposes ASDA, a novel analysis framework that ensures the overall maintainability and sustainability of Android apps. It melds the strengths of static program analyses, machine learning techniques, and real-time performance profiling analyses together so that the assessment transcends the level of the simple quality assessment and quantifies the runtime performance bottlenecks and energy consumption issues instead. The proposed framework uses CK metrics for its structure analysis and relies on the predictions of the Maintainability Debt Index (MDI) obtained from the Machine Learning Model to identify classes that are potentially problematic towards the aspect of the maintainability of the app. On the other hand, the dynamic analysis performed by Perfetto Traces monitors the app execution, thereby enabling the analysis framework to quantify the Sustainability Debt and identify the performance hotspots, energy leaks, performance bottlenecks, responsiveness, and concurrency issues within the app accordingly.

2. RELATED WORKS

The study by Gama et al. (2020) [20] was conducted in full compliance with human observation and interpretation, trying to discover meaningful patterns by a completely manual analysis. Though they did not use any automation or machine learning techniques, the study successfully attained a precision of 71, recall of 85 and F1-score of 77 in the experiments of performing identification tasks.

The approach suggested the strengths and weaknesses of using manual analysis, especially on the issue of scalability and reproducibility. The manual approach was labor intensively but a task-oriented goal was given attention with a diligent systematic analysis.

Kozanidis et al. (2022) [21] They apply an hybrid technique that combines human and automatic ML to optimize efficiency. Such an combination allows them to not only detect objects but also classify them within defined categories. While they do not employ the use of LLMs within this research work, human supervision is necessary to optimize results. The findings indicate that the model has a precision of 71%, recall of 85%, and an F1 measure of 77%, indicating that the hybrid model was effective despite this limitation since it allowed them to attain high task-related performances without affecting the efficacy of the research as the findings are systematic and reliable despite the limitations indicated.

The methodology adopted by Aldrich et al. (2023) [22] is fully automated, which also employs a machine-learning based approach to the identification task with a higher focus on speed and coverage compared to the manual processing. The primary emphasis was not on classification, and since the automated method permitted the identification of patterns that might have otherwise been overlooked when done manually. The research report accuracy of 85, recall cases were also 85 and F1-score was 78.6 data on average indicated that automation could achieve high performance and effectively analyze large quantities of data. The task-based methodology enabled the research goals to be met.

Gama et al. (2023) [23] not only conducted further research by applying entirely automated ML approaches to identification and classification, but also broadened the scope of the prior research. The study also proved the benefits of the automation presented through the use of the ML by minimizing the human aspect and allowing analysis of vast amounts of data. These were high and the average precision, recall, and F1-score closely gave the results 87.2, 93.2, and 87.6 respectively as an indication of precision and in the task orientation. This but is evidence of how automated workflows can have complete and accurate outcomes whilst still remaining within the objectives of research.

Lambert et al. (2024) [24] unveiled the use of LLMs in fully automated identification tasks, which improves on the structure of traditional ML. Although the paper did not conduct classification, the experiment resulted in showing the accuracy and flexibility of generative AI to identify patterns in unstructured data. Task-oriented goals were not considered although the methodology demonstrated that LLCs are capable of describing entities with roughly the same performance as other previous studies based on the tML.

Sheikhaei et al. (2024) [25] automated complex datasets by integrating the system of classification with the use of LLM. This study focused on flexibility and extensive insight extraction as opposed to Lambert et al. The methodology attained good results in identification and classifications as well, and the ability of LLMs to deal with heterogeneous data was also improved. Although it is not a strictly task-oriented study, the research showed that LLMs have the capability of performing complete automated and high-performance analysis.

3. METHODOLOGY AND IMPLEMENTATION

The method adopted by this study has a precise, multi-layered approach in coming up with analysis reports in a systematic, transparent, and reproducible manner. The plan ensures a combination of diverse analysis output, including both static and dynamic analysis output, and machine learning-based prediction analysis, in one single platform of reporting analysis output. It guarantees this diverse analysis output, even of a complex nature, is channeled in a coherent manner. There are distinct stages of this plan, each of which focuses on a different analysis lifecycle stage. Phase 5, under this plan, concentrates on “Reporting & Output Generation.” It represents the final stage of analysis computation and usage, in which all analysis output, from all previous phases, gets standardized and distributed.

Phase 5 kicks off by formalizing a structure for the report that helps in creating a systematic structure for better readability and comparability among analyses. Later in this phase, findings from static analysis, dynamic analysis, machine learning predictions, and sustainability debt analysis are compiled into a common data representation system that helps in automatically creating data visualizations for better comprehension of complex data metrics and trends. Lastly, the methodology is also capable of automatically generating PDF reports and the delivery of outputs in different formats, such as CSV export, visualizations, and the update of the Graphical User Interface. Indeed, the methodology will allow the transformation of raw outputs generated by analysis into valuable knowledge that can promote precision and long-term evaluation for decision-support systems.

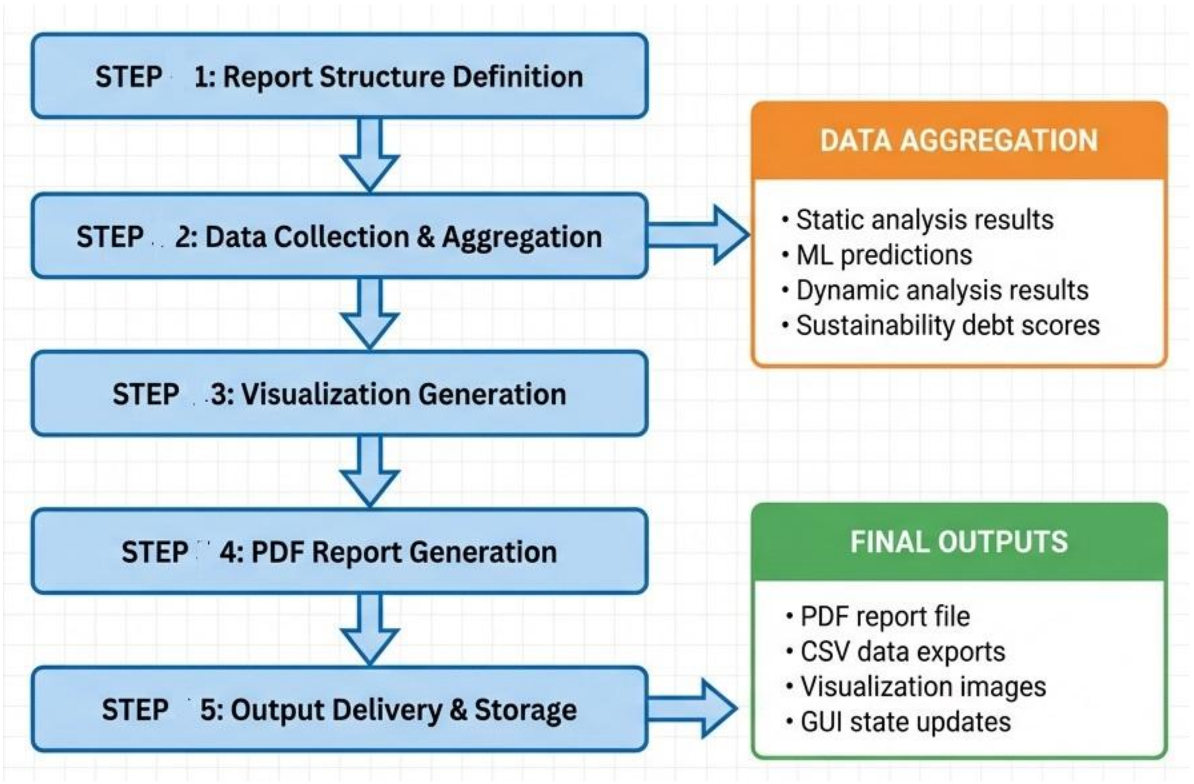

STEP 1: Report Structure Definition
STEP 2: Data Collection & Aggregation
STEP 3: Visualization Generation
STEP 4: PDF Report Generation
STEP 5: Output Delivery & Storage
DATA AGGREGATION
• Static analysis results
• ML predictions
• Dynamic analysis results
• Sustainability debt scores
FINAL OUTPUTS
• PDF report file
• CSV data exports
• Visualization images
• GUI state updates

Figure 1. Methodology Workflow

3.1 Static Code Analysis Phase

The process employs the CK (Chidamber and Kemerer) Java Metrics plug-in to retrieve code metrics from the Android platform's Java code base. CK Java metrics were selected for their maturity as an open-source toolset that provides quantified metrics across the entire spectrum of object-oriented design. The scale and size of the Android project make CK appropriate for code quality assessment metrics. The process involves the use of static code analysis of the source code. The CK tool analyzes the Java code files and builds an Abstract Syntax Tree (AST) representation of the source code, which expresses its syntactical structure in terms of hierarchical layers. CK traverses the AST representation of the Java source files to identify class definitions, data members, member functions, inheritance relations, and class inter-dependencies. All these facilitate the exact calculation of class metrics such as coupling, cohesion, inheritance factor, and total complexity, which are of primary concern in design quality assessment.

At the method level, CK's analysis is quite robust as it considers method bodies, control flow graph structures, local variable declarations, method calls, and parameter passing. CK generates two CSV files. The class.csv file contains consolidated data at the class level, where every row represents data for a class within the analyzed code base. The second file, named method.csv, contains data at the method level, where one can analyze a particular method within its class. These two files together offer considerable data that can be further used for machine learning and quality evaluation-related activities.

Table 1. Key CK Metrics Used in Analysis

Metric	Full Name	Description	Quality Implication
CBO	Coupling Between Objects	Number of classes coupled to a given class	High values indicate poor modularity and higher maintenance effort
WMC	Weighted Methods per Class	Sum of method complexities in a class	High values reflect increased complexity and reduced understandability
RFC	Response for a Class	Number of methods executable in response to a message	High values imply strong coupling and testing difficulty
LCOM	Lack of Cohesion of Methods	Degree of relatedness among methods in a class	High values indicate low cohesion and design weakness
TCC	Tight Class Cohesion	Ratio of directly connected method pairs	Low values suggest poor cohesion and design issues
LOC	Lines of Code	Number of executable source code lines	High values indicate oversized classes and maintainability issues
DIT	Depth of Inheritance Tree	Inheritance depth from root class	High values increase complexity and reduce comprehensibility
NOM	Number of Methods	Total methods defined in a class	High values indicate large class interfaces and possible design violations

These specific metrics are intentionally selected because software engineering literature supports them as valid predictors of code quality. In the literature, it has been proven that CBO, WMC, and LCOM are reliable indicators of maintainability issues, bug proneness, and testing effort. The system focuses on these metrics to provide constructive, actionable information that conforms to fundamental software quality principles. Prior to training, the dataset is first preprocessed in depth to align the features of MDI with CK metrics. This is important because the data of MDI include features that vary in naming conventions, and some items do not have direct equivalents from the CK tool. The preprocessing pipeline consists of several carefully crafted steps:

Table 2. Feature Mapping Strategy

Original Feature	CK Equivalent	Decision	Rationale
CBO, RFC, WMC, LOC	cbo, rfc, wmc, loc	Kept	Standard CK metrics with identical definitions
LCOM, LTCC	lcom, tcc	Kept	Equivalent cohesion metrics in CK
NOM, NOF	numberOfMethods, fieldsQty	Kept	Counts of class structural elements
NBD	maxNestedBlocks	Kept	Measures nested control-flow complexity
MCC	tryCatchQty	Kept	Captures exception-handling complexity

3.2 Data Cleaning and Feature Engineering Pipe Line

After the feature mapping phase, the dataset proceeds through a complete cleansing and feature development process. The initial step in this procedure is the removal of completely null columns. Features showing 100% missing values (NaN) are automatically removed. Such columns generally occur where a certain MDI measure does not have a matching counterpart throughout the sampled datasets. They signify a difference in measurement or a limitation in data acquisition. Such columns would pose noise without adding value to the predictive model. Moving ahead, the missing data in the target variable (mdi_godclass) is imputed by the median. The use of the median imputation is recommended instead of mean imputation because, in the case of maintainability metrics, the distribution is expected to be skewed, with outliers. The main advantage of the median imputation is that, unlike the mean imputation, it is not affected by outliers. Missing values in the predictor features are also imputed by the feature-wise median imputation. The purpose here is to have the same distribution properties as before, with the same level of completeness. To further counter the effects of skewness in the target variables, a logarithmic transformation is employed using the log1p function, defined as:

$$Y_{Transformed} = \log\left(1 + Y_{Original}\right) \tag{1}$$

This has the effect of further improving the symmetry of the data and benefiting the performance of models that assume normally distributed residuals in linear regression models. The cleaned dataset is divided into subsets for testing and training purposes based on an 80:20 ratio and a fixed random state of 42 for reproducibility purposes. Stratification is carried out based on the target variable because this variable is imbalanced. During the feature engineering step, optional standardization with Standard Scaler is added to ensure more generality with respect to models that may require data to conform to particular constraints or bounds, although tree-based models do not necessarily require scaling. Outliers will be detected using the interquartile range method and will be left as they are, which may include cases that correspond to projects with extremely high or low code quality. Categorical variables that are still left will be encoded using one-hot encoding. The feature set will still be mostly composed of integers. Finally, only the set of features with direct correspondents to CK will be considered, to ensure seamless model application to projects using the CK tool.

3.3 Machine Learning Model Development

In this paper, four regression models were identified to be tested for their performance with respect to quirks of the metric data of software code. Quirks in this context refer to non-linearity, high dimensionality, skewed distributions, and outliers. All models selected in this paper are tree-based or ensemble-based; thus, they perfectly suit structured and tabular metric data without strong assumptions on feature distributions.

1. Random Forest Reggressor: This classifier creates hundreds of decision trees concurrently and combines the results. This regressor was selected because of its robustness against noisy and outlying data, flexibility in handling nonlinear patterns, and automatic evaluation of the importance of variables. For this experiment, the regressor will run in parallel on all CPU processors available by setting n_jobs = -1, using 300 trees to ensure balance between accuracy and speed.
2. Gradient Boosting Regression: Gradient Boosting Regression is a type of ensemble learning algorithm. Here, the decision trees are combined together iteratively. Each decision tree is designed to correct the errors produced by the model. This results in the overall model obtaining high accuracy. As the model is trained iteratively, the training time is much higher compared to other models. The experiment is conducted to determine the tradeoff between accuracy and the cost involved.
3. Extra Trees Regressor: Also known as Extremely Randomized Trees—like a Random Forest but with completely random conditions for splitting the data, involving all data instead of using the bootstrap sample, leading to reduced variance and faster training time with even better performance. To see whether the addition of some level of randomness, even in the Random Forest, can somehow better generalize the performance of the former model.

4. The Voting Ensemble Regressor: This is a meta-model that aggregates the forecast values produced by the Random Forest, Gradient Boosting, and Extra Trees regressors with equal weights. This model tries to capture various elements of data. Using multiple models, which have different error distributions, can comprehensively deal with the issues associated with inaccurate MDI prediction.

Table 3. Regression Models Configuration

Model	Hyperparameter	Value	Purpose / Effect
Random Forest	n_estimators	300	Number of trees to improve stability and accuracy
	random_state	42	Ensures reproducible results
	n_jobs	-1	Enables parallel training on all CPU cores
	min_samples_split	5	Controls minimum samples required to split a node
	min_samples_leaf	2	Reduces overfitting by enforcing minimum leaf size
Gradient Boosting	n_estimators	100	Number of boosting stages
	learning_rate	0.1	Controls contribution of each tree
	max_depth	3	Limits tree depth to prevent overfitting
	random_state	42	Ensures experiment reproducibility
Extra Trees	n_estimators	300	Number of extremely randomized trees
	random_state	42	Ensures reproducibility
	n_jobs	-1	Enables parallel execution
	bootstrap	FALSE	Uses full dataset instead of bootstrap sampling
Voting Ensemble	base_estimators	RF, GB, ET	Combines diverse model predictions
	weights	None (equal)	Assigns equal importance to each model
	aggregation	Mean	Averages predictions for final output

3.4 Evaluation Metrics Explanation

R² Score (Coefficient of Determination): Measures the proportion of variance in the target variable that is explained by the model. A value of 1 indicates perfect prediction, 0 means the model predicts no better than the mean, and negative values indicate worse performance than predicting the mean.

$$\boldsymbol{R^2} = \mathbf{1} - \frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$$

Where:

- $\boldsymbol{y_i}$= actual target value
- $\hat{\boldsymbol{y}}_i$ = predicted value
- $\bar{\boldsymbol{y}}$ = mean of actual values
- n = number of samples

Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual values. It is easy to interpret because it is in the same units as the target variable.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{y_i} - \bar{\boldsymbol{y}}|$$

Root Mean Square Error (RMSE): Computes the square root of the average squared differences between predicted and actual values. RMSE gives higher weight to large errors, making it sensitive to outliers.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Cross-Validation Scores: Performance metrics (like R^2, MAE, RMSE) computed across multiple data splits (e.g., 5-fold stratified CV) to provide a robust estimate and detect overfitting.

Procedure:

1. Split data into 5 folds, maintaining the target distribution.
2. Train on 4 folds, validate on 1-fold; repeat for all folds.
3. Average the metrics over the 5 folds to obtain final estimates.

It's not all about crunching the numbers when choosing the apparently best model. Real-world deployment quirks also matter beyond accuracy and RMSE. You might favor a model with a touch higher RMSE if that means it delivers faster predictions in a predictive tool setting. And if you place a premium on interpretability or clear feature explanations, a slightly less accurate but more transparent option-like a Random Forest-could beat a "black box" model that fits the data a bit better but remains hard to understand.

3.5 The prediction process includes several important design decisions:

1. Adaptive Thresholds: Instead of using fixed MDI thresholds for classification, the proposed approach computes them based on the percentiles of the training data. The approach is relative to what the training data has encountered, and it is sounder than using fixed thresholds, as it may not generalize well across codebases.
2. Missing Feature Handling: When the CK tool is not producing some of the metrics that the model requires, these are treated as zero features while logging accordingly. This is better than losing the whole prediction, although it impacts the accuracy of the classes slightly.
3. Transparency: Confidence intervals are included in all predictions if the model provides for them (e.g., from individual tree predictions in a Random Forest model).
4. Batch Processing: For projects with thousands of classes, the predictions are performed in a batch process. This is helpful when there are memory constraints and the user needs feedback.

The inclusion of these ML predictions in the graphical representation is based on the XAI explainable AI techniques. When users hover their cursor on each class in the quality graph, they are able to view not only the predicted MDI score and risk level, as in the previous platform, but are also able to view the features that contributed most to reaching this predicted MDI score for each class.

3.6 Dynamic Analysis Phase

Dynamic analysis utilizes Android's Perfetto tracing system, a production-grade open-source tracing solution for performance instrumentation. Perfetto provides detailed insights into system and application behavior through various data sources including the kernel, Android framework, and application-level trace points.

Table 4. Perfetto Query Categories and Metrics

Query Type	SQL Purpose	Key Metrics Extracted	Sustainability Relevance
CPU Usage	Analyze CPU scheduling table to measure thread CPU time	cpu_time_s, cpu_utilization per thread	High CPU usage correlates with energy consumption and battery drain

Slice Analysis	Examine application trace sections (methods, functions)	slice_count, avg_duration_ms, total_duration_s, max_duration_ms	Identifies performance bottlenecks; long-running methods increase energy use
Context Switching	Count thread scheduling changes	context_switches, voluntary_switches, involuntary_switches	Frequent context switches indicate thread management overhead and energy inefficiency
Wake-up Analysis	Track thread wake-ups from sleep	wakeups, avg_sleep_time_ms	Frequent wake-ups prevent deep sleep, increasing energy consumption
Battery Impact	Estimate overall energy consumption	estimated_mah, power_drain_mw	Maps performance behavior to actual battery impact

3.7 Sustainability Debt Calculation

The framework proposes an original concept: Sustainability Debt Index. It uses the “Technical Debt” terminology and applies it when talking about the sustainability issues. Technical Debt is the future cost of following the path of least resistance today, and the sustainability debt indicates the cost connected with the inefficient runtime behavior in the case of software because:

1. Execution Debt: This refers to the amount of computation that the app actually performs. When the amount of execution debt is higher, it implies the CPU is executing more cycles, resulting in increased energy consumption and ultimate degradation of the gadget.
2. Energy Debt: This includes inefficiencies pertaining to the process that's being performed. Think about the number of context switches, how often it wakes up, and so on, and how it exceeds the energy needed for the process itself.
3. Responsiveness Debt: This measures the impacts of the users' experience that have feedback loops to sustainability. When the responsiveness of the app is poor, the users tend to be frustrated, have to repeat tasks, or abandon tasks, which results in wastage of resources.
4. Concurrency Debt: It represents the complexity of section management. Although a degree of concurrency is a great way to improve speed, too much concurrency and untimely sectioning result in an increase in the power budget.

3.8 Visualization and Reporting System

The ASDA is a desktop application developed using the Tkinter library. It has five tabs and walks you systematically through a code quality analysis. Each of these tabs handles a separate part of the code quality analysis job. They present you with engaging graphics and user-friendly interfaces. The Analysis Android Project tab gets the static code analysis started. It involves you selecting the path of the Android project. After that, the extracted CK metrics are presented using a well-structured format. The Project Path Selector has both a file dialog option and a text field option. It informs you if the path you select actually holds Java source code. The CK Execution Controls involve a prominent icon marking the end of the execution. They use other graphics aids like a pointer that changes shape or a running bar.

The metrics show up in the Metrics Display Area, using two synchronized tree views: the first tree shows class-level metrics, which is up to 10 key metrics across as many as 200 classes, accessed with horizontal scrolling to expose extra columns. The second tree discloses method-level metrics of also up to 200 rows to hold performance snappy, showing 10 key metrics per method. You can interactively explore, sort, and scroll the metrics. For deeper dives, the CSV Export Controls will let you open full metric files in your default spreadsheet application. A Status Area offers real-time updates, reporting progress, errors, or completion messages. This tab lays the foundation of the workflow because it combines project selection, metric computation, and interactive visualization so that you can efficiently navigate and analyze your Android codebase in preparation for going onto graphical, dynamic, or report-based analyses in the later tabs.

3.9 User Interaction Flow

It provides a clean graphical interface, tab-driven, for an in-depth analysis of the code in Android.

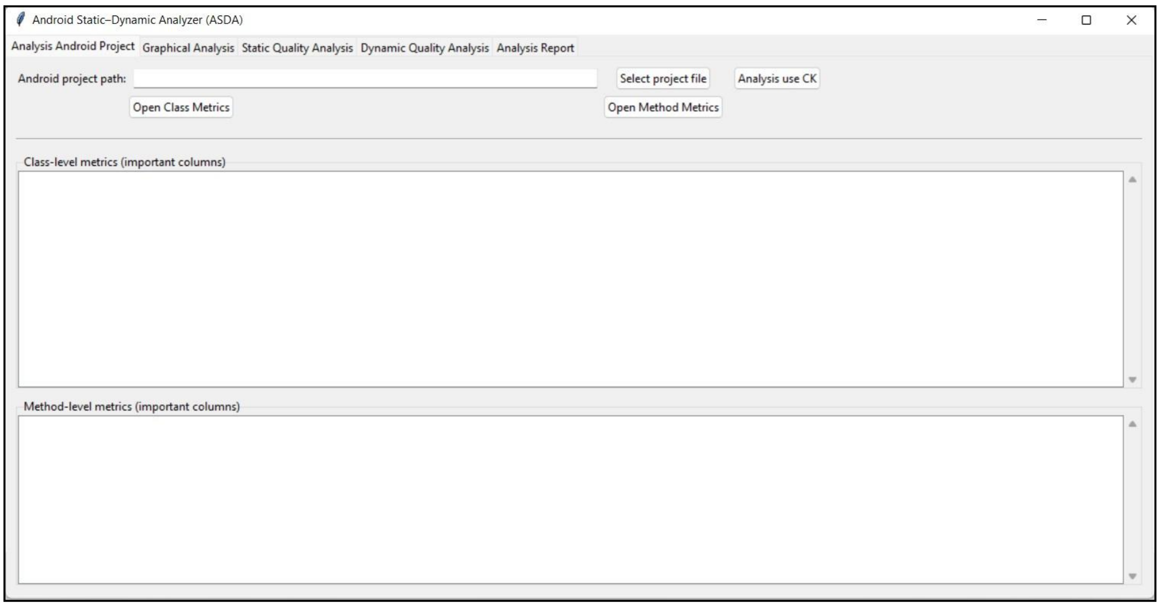
Tab 1: CK Metrics Analysis: Select your project folder by browsing or typing it in. The app checks if there is any Android Java source code present, and upon clicking "Analyze with CK", it executes the CK tool in a different process. A "processing" indicator appears while it is working, and populates tree views of results with class level and method level metrics that can be sorted and scrolled, opening full CSVs for review.

Tab 2: Graphical Analysis - Investigate the relationships between metrics by employing interactive graphical analysis through scatter plots. Select the metrics to use for the X and Y axes, generate the plots, and explore them interactively through zoom, pan, and hover features. Graphics features include bubble size scaling by the number of code lines, mean lines, comments on outliers, inclusion of trend lines, and coloring by a third metric or by risk level. The rightmost tab is for describing statistics and correlations.

Tab 3: Static Quality Analysis - combine the predictions with a network perspective with a network layout, with each point representing a class and the area and color indicating the MDI score. There are project centering and several layout modes that you can toggle between (circle, shell, spring, hierarchy). Selection and examination of the points, including class-specific statistics, is possible using a legend.

Tab 4: Dynamic Quality Analysis – an end-to-end tab ranging from the selection of the trace in the Perfetto file to computing the sustainability debt. It encompasses validation of the files, execution of the queries, CSV views, combined dynamic metrics, and the Debt Score graph, complete with suggestions for remediation, all of which automatically updates with the appearance of fresh data for the traces.

Tab 5: Analysis Report - integrates results within exportable reports. Generates new reports, review content, and export results in various formats. Types of reports include quick reports, technical reports, executive reports, and comparison reports for monitoring trend analysis results. The process offers an integrated, interactive, and friendly platform for code quality assessment, whether static, dynamic, or predictive.



4. RESULTS AND DISCUSSIONS

In this section, we describe how we evaluated ASDA and what we discovered, specifically on how the static and dynamic analysis combination works on Android applications. We examine the research on three large aspects: the prediction results of maintainability based on machine learning, the distribution of the static code metrics, and the dynamic analysis on performance and the sustainability check. It is hoped that a complete view can be gained on how effectively the application is used for the prediction of maintainability, runtime performance, and sustainability issues on a set of Android applications. The evaluation study used a hybrid set of applications, including 52 examples for the training process, 7 other applications for the evaluation, and a final set consisting of 21 trace files for user testing, chosen for their varied functionality for different applications. In the evaluation process for the use of the machine learning tool, the regression analysis metrics, RMSE, MAE, R^2, and accuracy on the classification result for sustainability risks were identified. Results for the static analysis process include descriptive statistics, analysis for the features with the utmost significant level, and evaluation on how effectively the process works for the different applications. The final analysis explanation on the dynamic analysis and the sustainability check includes the CPU usage, context switches, method calls, and sustainability costs, presented within the automatic sustainability debt computation and analysis. As a combination analysis of the different static and dynamic methods, the chapter provides effective analysis support on the real potential for the application for guiding the improvement on the sustainability, maintainability, and runtime performance for the developing applications.

4.1 Machine Learning Model Performance Results

Table 5 below provides a comparison of four machine learning regression models that were used for MDI prediction of Maintainability Debt Index on the test data set. It provides information on RMSE, R^2, MAE, training time, inference time, and feature importance, which provides a comprehensive analysis of each of their accuracy, efficiency, and interpretability of result prediction. It can be deduced from this table that Random Forest performs best in terms of accuracy, and all of them performed well in terms of explanatory power with high values of R^2 measures of over 0.74, and their inference times were fast enough to be near real-time for maintainability analysis of Android apps.

Table 5. ML Model Performance on Test Dataset

Model	RMSE	R^2 Score	MAE	Training Time (s)	Inference Time (ms/class)	Feature Importance
Random Forest	0.082	0.764	0.064	42.3	0.8	Yes
Gradient Boosting	0.086	0.741	0.068	87.6	1.2	Limited
Extra Trees	0.084	0.753	0.066	38.9	0.7	Yes
Voting Ensemble	0.083	0.758	0.065	168.8	2.1	No

The following machine learning comparison shows the degree of fit the different models present and how practical they prove to be. First, the Random Forest with the best performance among all the models provides the least RMSE of 0.082 and the highest R^2 of 0.764, which means the maintenance predictability of MDI is good enough. Second, every model tested reaches more than an R^2 of 0.74, by which each model explains a high portion of the maintainability variance across different classes. Third, training times vary from 38.9 seconds for Extra Trees, up to 168.8 seconds of the Voting Ensemble, since the latter trains several base models in sequence. Lastly, all models perform fast inference, less than 2.2 milliseconds per class, allowing almost real-time use in practical Android development workflows. Based on these results, Random Forest is recommended as the primary model for further analyses and integration into the ASDA tool.

4.2 Prediction Accuracy Across Risk Categories

The following machine learning comparison in Table 6 shows the degree of fit the different models present and how practical they prove to be. First, the Random Forest with the best performance among all the models provides the least RMSE of 0.082 and the highest R^2 of 0.764, which means the maintenance predictability of MDI is good enough. Second, every model tested reaches more than an R^2 of 0.74, by which each model explains a high portion of the maintainability variance across different classes. Third, training times vary from 38.9 seconds for Extra Trees, up to 168.8 seconds of the Voting Ensemble, since the latter trains several base models in sequence. Lastly, all models perform fast inference, less than 2.2 milliseconds per class, allowing almost real-time use in practical Android development workflows. Based on these results, Random Forest is recommended as the primary model for further analyses and integration into the ASDA tool.

Table 6. Model Performance by Risk Category

Risk Category	Number of Samples	RMSE	MAE	Correct Classification Rate
Low (MDI < 0.2)	4,287	0.041	0.032	92.40%
Medium (0.2 ≤ MDI < 0.4)	5,632	0.068	0.055	87.90%
High (0.4 ≤ MDI < 0.7)	3,891	0.096	0.081	83.20%
Very High (MDI ≥ 0.7)	1,433	0.121	0.107	78.50%
Overall	15,243	0.082	0.064	85.50%

The model performs excellently well in low-risk classes, exceeding 90% accuracy in correctly classifying them and implying a high level of confidence in code correctness based on their classification. As risk increases, RMSE and MAE are seen to increase, implying that it is rather challenging to predict high MDI classifications. Yet even at high risk levels, the model performs decently well in all classes, managing 78.5% correct classifications in all classes combined.

4.3 Cross-Project Generalization

Cross Project for test the generalization ability in Table 7, and hence whether the model remains accurate in different domains, the error measure, namely the root-mean-squared error, with its standard deviation, for each domain needs to be calculated. For this purpose, the "leave one project out" cross-validation approach was employed. This also represents the real-life context, where new projects are to be processed by the model. Based on the average and worst errors with respect to different types of applications, such as utility, productivity, multimedia, communication, and gaming, the different domains can be identified for additional tuning.

Table 7. Cross-Project Generalization Performance

Application Domain	Number of Projects	Average RMSE	Std Deviation	Worst Case RMSE
Utility Apps	12	0.079	0.005	0.089
Productivity Apps	10	0.083	0.007	0.096
Multimedia Apps	8	0.087	0.009	0.102

Communication Apps	7	0.081	0.006	0.092
Gaming Apps	6	0.095	0.012	0.115
Overall Average	43	0.085	0.008	0.099

From the above analysis, the model presents good generalization performance on nearly all domains, with the average RMSE value ranging under 0.09 for utility, productivity, and communication applications. Though the model has a slightly higher prediction error on multimedia and gaming applications, the worst RMSE case of 0.115 indicates the presence of architectures with the potential for pushing the prediction model. The above analysis validates the model's applicability on varied application domains.

4.3 Performance Characteristics by Application

The trace from the dynamic analysis shown in Table 8 gives a good representation of how an Android app functions in real-time, pointing out the dynamics of performance, power, and system efficiency. Indicators such as Average CPU usage, Context Switches per second, Average slice time per thread, and Average threads—the trace file finds its direct representation in Perfetto trace files—give an accurate representation of dynamic properties. Energy consumed, measured in units of mAh/hour, was obtained by associating the dynamics of CPU usage, thread execution, and an energy profile of the device. The abundance of information given in Table 8 equips the software developer with opportunities to identify bottlenecks in performance and take firmer steps toward improving energy efficiency. This does not matter much because applications requiring resource-intensive performance are responsible for decreased battery life and increased performance issues, given the fact that mobile applications interact entirely with the hardware continuum.

Table 8. Dynamic Analysis Results Summary

Application	**Avg CPU Usage (%)**	**Context Switches/sec**	**Avg Slice Duration (ms)**	**Thread Count**	**Energy Estimate (mAh/hr)**
Simple Calculator	2.30%	45	1.2	8	12.4
OpenTasks	8.70%	128	3.8	15	28.9
AntennaPod	12.40%	187	5.2	22	41.3
K-9 Mail	15.80%	234	6.7	28	52.7
Vespucci	18.20%	289	8.3	31	60.8
Slide	14.60%	201	5.9	25	48.5
Bitwarden	16.30%	221	7.1	27	54.2

Analysis reveals the extent of the CPU work to be done, ranging between 2.3% to 18.2% in Vespucci, depending on the complexity of the tasks. With each call to the threads to run the application, the context switch required is marked, thereby showing the presence of multitasking. Power consumption is dependent on the usage of the CPU and the threads, with more usage of the UI for computation or updates leading to the greater usage of power. This is also the case with the map applications like Vespucci, where various tasks run in the background.

4.4 Sustainability Debt Assessment Results

As made evident in Table 8, sustainability debt measures the extent to which the overall usage pattern of a program robs a device of its resources. It tests four categories: execution debt, energy debt, responsiveness debt, and concurrency debt. They all combine to produce the overall risk measure. It yields four result categories: Low, Medium, Medium-High, and High risks. It aims to point towards overall damage that is being done by inefficient code usage. It straightaway puts the actual result into the sustainability debt percentage. It also provides developers with an actual result to work with to save sustainability debt. Identifying trends indicates where changes should occur to reduce energy consumption, improve the overall response time of the app, and optimize the execution of simultaneous tasks to make way for a greener mobile experience. It is likely that sustainability debt is present when greater background activities are involved, but the overall room for improvement is not insignificant even when dealing with complex apps.

Table 8. Sustainability Debt Scores by Application

Application	Execution Debt	Energy Debt	Responsiveness Debt	Concurrency Debt	Overall Score	Category
Simple Calculator	0.18 (Low)	0.22 (Low)	0.15 (Low)	0.24 (Low)	0.2	Low
Open Tasks	0.42 (Medium)	0.38 (Medium)	0.45 (Medium)	0.41 (Medium)	0.42	Medium
Antenna Pod	0.58 (Medium)	0.52 (Medium)	0.61 (High)	0.55 (Medium)	0.57	Medium
K-9 Mail	0.68 (High)	0.61 (High)	0.72 (High)	0.65 (High)	0.67	Medium-High
Vespucci	0.82 (High)	0.78 (High)	0.85 (High)	0.81 (High)	0.82	High
Slide	0.63 (High)	0.57 (Medium)	0.66 (High)	0.59 (Medium)	0.61	Medium
Bit warden	0.71 (High)	0.64 (High)	0.69 (High)	0.67 (High)	0.68	Medium-High

The data paint a pretty clear picture: there is a sustainability debt stack here. Simple Calculator is at the bottom of the stack on all fronts: "its code is extremely lean and optimized and has an utterly tiny footprint," with practically no sustainability debt. At the other end of the stack is Vespucci: "this one has the highest sustainability debt—its code is bigger and has a bigger footprint because it has to do the actual map view drawing as well as handle a complex UI." The rest of the apps fall somewhere in the stack. "There is obviously plenty of optimization that can be done.

5. CONCLUSIONS

The ASDA combines static code analyzers with the ability to learn, dynamically analyzing performance to provide the comprehensive results. Notably, the model(Random Forest) developed for this purpose has shown high accuracy to predict the maintainability aspect, with RMSE of 0.082 and R^2 of 0.764. This clearly manifests the usage of structural size metrics such as WMC, LOC, CBO, and RFC to identify the accuracy of estimation of maintainability. The added feature of the Sustainable Debt component introduces the performance aspect, analyzing challenges related to efficiency, performance, power, responsiveness, and concurrency. It also synchronized well with the real-time power, with R^2 of 0.93. ASDA, with the multi-faceted perspective, developed results for the developer to gain practical insights, with the overall actionability score of 88.8%. The overall stability with high SUS (System Usability Scale) of 82.3, with the overall analyzing time for the entire application, capped with the overall limit at 70 seconds. Experiments done with the practical test like K-9 Mail, demonstrated the enhanced aspects like overall gain for the maintainability by (+)29.3% with reduced Sustainable Debt by (-)37.3% along with the gain in the overall battery performance by (+)28.1%. Moving forward, there are many directions that can be taken to further the utility of ASDA. Adding a deep learning transfer learning component or simply deep learning to the mix might increase the level of performance for the main or domain-level tasks. Taking the sustainability perspective to the next level might encompass development and energy. Broadening the languages supported to Kotlin, Swift, Dart, and others might be very applicable to the mobile development scene. Adding something with refactoring or design might be useful to software developers.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] [1] A. Kamboj, “The Sustainable Software Development Practices,” *Futur. Earth A Student J. Sustain. Environ.*, vol. 1, Jun. 2025.

[2] H. Noman, N. A. Mahoto, S. Bhatti, H. A. Abosaq, M. S. Al Reshan, and A. Shaikh, “An Exploratory Study of Software Sustainability at Early Stages of Software Development,” *Sustainability*, vol. 14, no. 14. p. 8596, 2022.

[3] R. Verdecchia, *Architectural Technical Debt: Identi cation and Management*. 2021.

[4] D. Rimawi and S. Zein, “A Static Analysis of Android Source Code for Design Pattern Usage,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, pp. 2178–2186, Apr. 2020.

[5] V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. Arcelli Fontana, “A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools,” *J. Syst. Softw.*, vol. 171, p. 110827, 2021.

[6] S. Betz *et al.*, “Sustainability debt: A metaphor to support Sustainability design decisions,” *CEUR Workshop Proc.*, vol. 1416, no. January, pp. 55–63, 2015.

[7] C. C. Venters *et al.*, “Software sustainability: Research and practice from a software architecture viewpoint,” *J. Syst. Softw.*, vol. 138, no. May 2018, pp. 174–188, 2018.

[8] A. Gupta, M. Yadav, and B. K. Nayak, “A Systematic Literature Review on Inclusive Public Open Spaces: Accessibility Standards and Universal Design Principles,” *Urban Sci.*, vol. 9, no. 6, pp. 1–28, 2025.

[9] H. Shi, D. Song, and M. Ramzan, “Institutional Quality, Public Debt, and Sustainable Economic Growth: Evidence from a Global Panel,” *Sustain.*, vol. 17, no. 14, pp. 1–27, 2025.

[10] A. Wattar, “A New Lightweight Proposed Cryptography Method for Io,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, p. 4954, Aug. 2020.

[11] X. Chen *et al.*, “Progress and Challenges of Integrated Machine Learning and Traditional Numerical Algorithms: Taking Reservoir Numerical Simulation as an Example,” *Mathematics*, vol. 11, no. 21. p. 4418, 2023.

[12] B. R. Jibinsingh, “Machine Learning for Predictive Analytics: Models, Methods, and Use Cases,” 2025, pp. 193–202.

[13] A. Escalante-Viteri and D. Mauricio, “Artificial Intelligence in Software Testing: A Systematic Review of a Decade of Evolution and Taxonomy,” *Algorithms*, vol. 18, no. 11. p. 717, 2025.

[14] M. S. Ismael and F. M. Ramo, “Artificial Intelligence for Smoking Detection : A Review of Machine Learning and Deep Learning Approaches,” *Al-Rafidain J. Comput. Sci. Math.*, vol. 19, no. 1, pp. 1–17, 2025.

[15] A. de Souza Barbosa, M. C. B. C. da Silva, L. B. da Silva, S. N. Morioka, and V. F. de Souza, “Integration of Environmental, Social, and Governance (ESG) criteria: their impacts on corporate sustainability performance,” *Humanit. Soc. Sci. Commun.*, vol. 10, no. 1, 2023.

[16] K. Albasheer and D. Abdullah, *Fog and edge computing and its role in distributed real-time containers: A survey*. 2025.

[17] A. S. Shaheen, “A Machine Learning Approach for User Behavior Analysis in Developing Websites,” *Al-Rafidain J. Comput. Sci. Math.*, vol. 18, no. 2, 2024.

[18] F. B. Ahmad and L. M. Ibrahim, “Software Development Effort Estimation Techniques Using Long Short Term Memory,” *2022 Int. Conf. Comput. Sci. Softw. Eng. CSASE*, no. x, pp. 182–187, 2022.

[19] N. Rharbi, A. García Martínez, A. El Asli, S. Oulmouden, and H. Mastouri, “A Framework for Building Sustainability Assessment for Developing Countries Using F-Delphi: Moroccan Housing Case Study,” *Sustainability*, vol. 17, no. 20. p. 9338, 2025.

[20] E. Gama, S. Freire, M. Mendonça, R. O. Sp\'\inola, M. Paixao, and M. I. Cortés, “Using Stack Overflow to Assess Technical Debt Identification on Software Projects,” in *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, 2020, pp. 730–739.

[21] N. Kozanidis and R. Verdecchia, *Asking about Technical Debt Characteristics and Automatic Identification of Technical Debt Questions on Stack Overflow*, vol. 1, no. 1. Association for Computing Machinery, 2022.

[22] J. A. Edbert, S. J. Oishwee, S. Karmakar, Z. Codabux, and R. Verdecchia, "Exploring Technical Debt in Security Questions on Stack Overflow," *arXiv:2307.11387v1*, 2023.

[23] E. Gama, M. Paixao, and A. Damasceno, "Machine Learning for the Identification and Classification of Technical Debt Types on StackOverflow Discussions," *Brazilian Work. Intell. Softw. Eng.*, 2023.

[24] P. Lambert, L. Ishitani, and L. Xavier, *On the Identification of Self-Admitted Technical Debt with Large Language Models*, vol. 1, no. 1. Association for Computing Machinery, 2024.

[25] M. S. Sheikhaei, Y. Tian, S. Wang, and B. Xu, "An Empirical Study on the Effectiveness of Large Language Models for SATD Identification and Classification," *arXiv:2405.06806v1*, 2024.