Finding Keywords for Architectural Erosion Detection in GitHub Commits for Android Applications

Juan Camilo Acosta-Rojas
Universidad de los Andes, Colombia
jc.acosta2@uniandes.edu.co

Camilo Andrés Escobar-Velasquez
Universidad de los Andes, Colombia
ca.escobar2434@uniandes.edu.co

***Abstract*—Architectural Erosion (AER) is a phenomenon that occurs when the implemented architecture of a software project diverges from its intended design. This can impact the quality and performance of an application. In Android applications, the effects may be amplified due to limited resources such as memory, storage, and processing power. Previous efforts have been done to tackle AER for different platforms, mainly using static code analysis and AI-based approaches using Word-embeddings. Nevertheless, no previous study has focused on Android Apps. The goal of this research is to evaluate the applicability of the proposed approaches based on word-embeddings to identify new potential keywords in GitHub commits of Android projects, using the existing list of keywords and word similarity metrics.**

***Index Terms*—Word Embedding, AI, Android, GitHub, Commits, Software, Similarity, NLP**

I. Introduction

Architectural erosion is a phenomenon generated by the deviation between an implemented architecture and its design. This could affect the flow of a software application and its performance. In Android applications, those symptoms could affect the application's performance on a larger scale. These symptoms can affect the memory usage, processing metrics, and storage usage of Android devices. Furthermore, it could directly affect the user experience, a very important metric for Android application survivability. To avoid those issues, there are different detection methodologies: (i) using static analysis and (ii) AI models powered by NLP fundamentals. On one hand, static analysis methodologies search for code patterns using code representations to identify AER issue. On the other hand, AI-powered methodologies uses pre-trained Word Embedding models to detect keywords in developers' commits, leading to the identification of different architectural smells in source code implementations. The aforementioned methodologies has been used mainly for backend and frontend project analysis [11], [14]. Based on the existing shortfalls of static analysis, in this research we evaluate the applicability of using Word-embedding models to find similar keywords related to AER issues within GitHub commits for Android applications written in Kotlin [1]–[5].

II. Related Work

Architectural erosion detection has been investigated previously, this has led to a continuous proposal of methodologies to identify the architectural smells that are inherited from evolving architectures. The most well-known detection methodologies are based on static analysis of source code and NLP and IA techniques.

A. Static Code Analysis methodology

This methodology relies on the static analysis of source code and the identification of architectural smells based on expert evaluations across various development stacks and code . Through these expert assessments, a set of architectural smells has been identified. These smells are detected using plugins and libraries developed for specific Integrated Development Environments (IDEs). The tools are applied to identify issues related to connectivity, security, and availability in both backend and frontend development projects. [11]–[13].

B. AI Models methodology

This methodology leverages the capabilities of Artificial Intelligence (AI) and Natural Language Processing (NLP) to detect Architectural Erosion and Regression (AER) issues using textual artifacts from source code versioning platforms. The process produces a dataset suitable for training AI models aimed at automated AER issue detection and code generation. By analyzing developer messages that describe implementations, it is possible to apply NLP techniques to extract semantically rich terms that may indicate poor architectural practices at specific points in a project's development.

Keyword Extraction Using NLP. Previous research has applied this methodology to identify words with high semantic value in developer messages associated with architectural smells or poor design decisions. Using a pre-trained static word embedding model—trained on over two million Stack Overflow posts—researchers generated a list of relevant keywords from commit messages on platforms such as OpenStack and the Qt Community. The process began with mining commits from projects written in different programming languages (e.g., Python, C++), followed by a manual labeling phase conducted by four independent researchers. The resulting dataset included 50 keywords frequently found in messages that potentially signal bad architectural practices or AER issues [14].

AI Model Development. Building on this approach, various AI models have been trained using datasets extracted from

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00026

versioning platforms like GitHub and OpenStack. These models utilize AER-related metrics—such as coupling and component dependency levels—to analyze source code. Techniques such as Decision Trees, Logistic Regression, and perceptron-based models have achieved high accuracy in detecting AER issues [14], [15]. More recent work has focused on training transformer-based models with large datasets of source code to improve performance in detecting code smells and generating code. Models such as BERT, CodeBERT, CodeT5, and GPT-based architectures have shown improved effectiveness in these tasks [16]–[20].

III. Using NLP for AER issues in Android

We adapted the AI-based methodology for detecting AER issues with a focus on Android applications. Using the previously defined list of keywords, we apply word similarity metrics and NLP techniques to process commit messages from 50 open-source Android applications written in Kotlin. These commits were extracted from GitHub. We then establish a data processing pipeline that incorporates both static and dynamic word embedding models to analyze the textual content of the commits. The goal is to identify terms semantically similar to the original keywords and propose new candidate keywords that may indicate AER issues in specific source code implementations.

A. Processing flow of Android applications

1) Android applications selection: A total of 50 Android applications were selected using different search and repository mining platforms. The first platform used was GHS (GitHub Search Engine for Repository Analysis and Tagging) [21]. We applied customized filters related to the number of commits (more than 1k, 10K and 50K) and the main programming language as kotlin. The second platform was F-Droid [22], an open-source catalog that provides metadata, including GitHub repository links, licenses, and project details. Different applications were selected randomly from FDroid, taking into account the similarity on features of the applications found in GHS.

2) Data Extraction: In order to extract the information of the app we used the PyDriller library [23]. This library uses web scraping techniques in GitHub repositories. In this step, we built a dataset with different extracted attributes (See Table I). Based on that format, we extracted more than 470K commits for data analysis for AER issues detection.

TABLE I
DATA EXTRACTED FROM GITHUB REPOSITORIES OF ANDROID APPLICATIONS SET

Column	Description
ID	assigned ID for code analysis
Name Repo	Repository Name
Url Repo	GitHub repository Url
Commits Hash	Specific commit hash
Commit Message	Message of specific commit
Code Changes	Source code implementation

3) Data Analysis with Word Embedding models: Using a large dataset of GitHub commits from various Android projects, we implemented both static and dynamic word embedding models to obtain numerical representations of each word in the vocabulary. After preprocessing the commits, we applied several static embedding models to identify words semantically similar to the predefined keywords. As a complementary approach, we generated lists of the most similar words for each keyword. In addition, we employed the pre-trained CodeReviewer model to discover new keywords related to AER smells within the commit dataset.

Static Word Embedding models: We implemented different pre-trained Word Embedding models and compared them between similarity metrics respect with the original defined keywords. The used static models were the SO model [14] , the mentioned model trained with millions of Stack Overflow posts. The other tested models are Word2Vec [25] and Glove [24] models. After that, we will compare the numerical representations given by each model for similarity words measurement. Finally, we added the most similar keywords based on the similarity of those words.

Dynamic Word Embedding models: We selected a pre-trained AI model based on transformer architecture like Code Reviewer [17]. AI models based on transformer architecture has an integrated Word Embedding model component with a positional encoding. The main difference with the use of static Word Embedding models is the dynamic context that handles this kind of Word Embedding models. With a training process oriented to AER issues detection, we extracted the Word Embedding component of Code Reviewer model and used the word similarity metrics for finding new keywords related to AER issues in the extracted GitHub commits dataset. We trained the CodeReviewer model to evaluate the words similarity with a set of commits of Android projects.

4) Testing and evaluation criteria: We realized a comparison between the three mentioned word embedding models. We used the cosine similarity average and selected the 10 most similar words to all the set of keywords. Cosine similarity average could be useful in specific contexts. For the trained AI model, we used as additional testing criteria the tagging process of a commits subset. The tagging was implemented using an AI agent of the gpt-4o model of openAI [26].

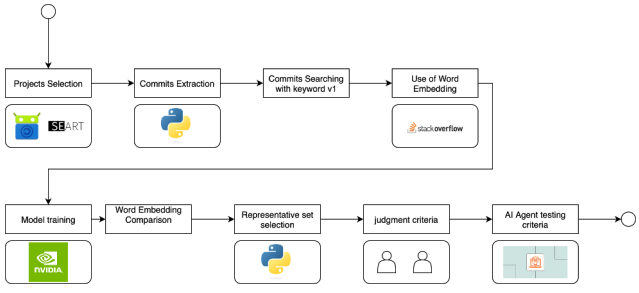

Fig. 1. Workflow for finding new AER keywords in Android projects

IV. Detection Methodology Implementation

A. Static Word Embedding models

With the selected repositories of Android applications, we tagged the extracted commits with an integer value. If the commit contained one or more keywords of the selected set, the commit would be tagged with 1. Otherwise, the commit would be tagged as 0. We implemented NLP techniques in the corpus of the commits dataset. We used stop word removal, lemmatization, stemming, and tokenization of every word. We loaded the static word embedding models and extracted the value of every token of the processed corpus. With those values, we used the cosine similarity average to identify the most similar words compared to the word embedding value of each keyword in the selected set. We extracted the top 10 words founded in every word embedding model and compared the value of every word. The results are shown in Table II. The extracted keywords from the SO model were the keywords with the highest cosine similarity. The other word embedding models provided words that were not related in the technical context of software development. Effectiveness of static word embedding models depends on the training context. With the top 10 similar words, we added the words that were not included in the keywords set previously. We extracted a representative sample of 350 commits. We used manual tagging with two judges to evaluate the detection accuracy of the new set of keywords. We compared the tagged commits against the commits that contain one or more keywords in the representative sample. After that, we obtained an accuracy of 70% of AER issues detection in the repositories of the Android projects. The use of the SO word embedding model is considered as useful to find new keywords in the Android context.

B. Code Reviewer Training

We loaded the standard CodeReviewer model, which is based on the RoBERTa architecture and encoder. This model was trained on multiple tasks involving both data generation and classification. For our experiments, training was performed on an NVIDIA virtual machine with 16 GB of RAM.

From the original dataset of extracted commits, we selected approximately 280K commits, constrained by infrastructure memory limitations. Each commit was labeled as positive if its message contained a predefined keyword, and negative otherwise. The model was trained for three epochs, with 15% of the data reserved for validation.

Using this setup, the CodeReviewer model was trained to classify code as an AER (Automated Error Recognition) issue. The target labels were determined according to the initial keyword list. After three epochs, the model achieved high accuracy on the test set, with results summarized in Table III.

With the trained CodeReviewer model, we extracted its dynamic word embedding model and used the average of the cosine similarity of each word in its tokenizer compared with the value of every keyword. The founded keywords are shown in Table IV.

We observed that the cosine similarity values of each keyword are higher compared to the values of the keywords obtained with static word embedding models.

C. Testing CodeReviewer

We implemented an AI agent to evaluate a sample of code fragments previously tagged by CodeReviewer. Specifically, we used the GPT-4o model [26] to simulate automated tagging of architectural erosion (AER) issues. The integration was carried out using the OpenAI and Azure OpenAI libraries [27].

A custom prompt was designed to guide the AI agent, including detailed instructions and examples of architectural erosion in code snippets. The prompt defined specific architectural violations related to various software quality attributes and highlighted poor implementation patterns typically found in Android projects written in Kotlin. These patterns included class coupling and improper dependency management, both considered indicative of AER issues. We randomly selected 200 commits for evaluation, constrained by cost limits that restricted the data volume. Commits were chosen based on their length in lines of code. For each commit, we compared the AI-generated labels with those produced by CodeReviewer. The AI agent achieved an accuracy of 55%, which represents a promising initial result for this type of learning task. Further improvements could be achieved by increasing the dataset size and refining both the CodeReviewer annotations and the use of smaller, specialized AI models. Here the implemented prompt.

You are an expert in software architecture for Android applications written in the Kotlin programming language. You will receive a code fragment extracted from a GitHub commit in an Android app repository. This code fragment contains only the lines that were changed during the commit: Lines starting with '+' represent added code. Lines starting with '-' represent removed code. Your task is to analyze only the lines that were added or removed, and determine whether the changes introduce or contribute to architectural erosion. Architectural erosion refers to the progressive degradation of a system's design and structure due to poor development practices. It can be caused by: Violations of SOLID principles, High coupling between modules or components, Dispersion of business logic, Repetitive or duplicated changes across multiple architectural layers, Leaking logic between layers (e.g., UI handling persistence directly), Adding code without tests or separation of concerns, Implementation of blocking functions in asynchronous contexts, Poor or missing exception handling, Improper exception handling within ViewModel classes. Return only a number: - 1 if the added or removed code could introduce architectural erosion

V. Conclusions

Both static and dynamic word embedding models can be useful for identifying potential keywords in GitHub commits from Android projects. Static word embedding models rely on the training context, and when trained within a software development domain, they can yield more accurate results by uncovering new keywords related to quality attributes in Android applications. To measure similarity, the cosine similarity metric proves effective, as its averaged values can highlight semantically related words within a given context. On the other hand, dynamic word embedding models, often

TABLE II
COSINE SIMILARITY BETWEEN STATIC WORD EMBEDDING MODELS

Word	general	respect	design	notion	common	complex	formal	depend	adopt	differ	hing	to-do	borderless	christian	rend	list	misc	non-act	contributor	wiki	conform	disregard	mismatch	implement	distinct	contradict	non-standard	rigid
SO model	0.31	0.3	0.29	0.28	0.266	0.265	0.264	0.262	0.262	0.26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Glove model	0	0	0	0	0	0	0	0	0	0	0.1019	0.1017	0.1	0.09	0.086	0.085	0.083	0.081	0.0809	0.802	0	0	0	0	0	0	0	0
Word2Vec model	0	0	0.2379	0	0	0	0	0	0	0.2269	0	0	0	0	0	0	0	0	0	0	0.2382	0.231	0.23	0.2298	0.22292	0.2291	0.2280	0.2263

TABLE III
METRICS OF CODEREVIEWER RESPECT TO TEST DATA IN AER ISSUES CLASSIFICATION

Class	Precision	Recall	F1-Score
0	93%	94%	93%
1	94%	93%	93%

TABLE IV
KEYWORDS FOUND IN THE WORD EMBEDDING MODEL OF TRAINED CODEREVIEWER MODEL

Word	Cosine Similarity
layout	0.4696
concept	0.4535
package	0.4523
controller	0.4483
settings	0.4440
generic	0.4414
interface	0.4414
element	0.4393
application	0.4281
purpose	0.4280

derived from advanced AI architectures, offer an even more powerful approach for detecting new keywords. Their training process allows them to adapt to specific contexts, making them well-suited for both classification tasks and the generation of new data.

VI. FUTURE WORK

Nowadays, there are many AI models and trained word embedding models for different training tasks. Using these models can improve the quality of the software development process. It is possible to extend the implemented methodology to learning tasks in Large Language Models (LLMs). Creating AI agents for keywords detection of specific quality attributes. This research could be helpful for analyzing large amounts of data in Android projects to evaluate architecture quality. This solutions could decrease cost in any software project. It is essential to leverage process optimizations at every stage of software development. The use of AI and NLP could ensure better development quality and adherence to standards and policies for a specific architecture [28], [29].

REFERENCES

[1] Perry, D and Wolf, A. "Foundations for the Study of Software Architecture". ACM Sigsoft. 1992

[2] Li, R. Liang, P. Soliman, M. Avgeriou, P. Understanding software architecture erosion: A systematic mapping study. Wiley Online Library. 2022.

[3] Li, R. Soliman, M. Liang, P. Avgeriou P. Symptoms of Architecture Erosion in Code Reviews: A Study of Two OpenStack Projects. ArXiv. 2022.

[4] Chauhan, K. Kumar, S. Sethia, D. Alam M. "Performance analysis of kotlin coroutines on android in a model-view-intent architecture pattern". Science Direct. 2022.

[5] St. Amour. LTilevich, E. "Toward Declarative Auditing of Java Software for Graceful Exception Handling". ACM Library. 2024.

[6] Budanitsky, A. Hirst, Semantic distance in word-net: An experimental, application-oriented evaluation of five measures. Proceedings of the NAACL Workshop on Word-Net and Other Lexical Resources. 2001.

[7] C.Y . Lin, E.H. Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. HLT-NAACL'03.

[8] Patwardhan, S. Banerjee, S. Pedersen, T. Using mea-sures of semantic relatedness for word sense disambiguation. CICLing'03.

[9] Martinez-Gil, J. "A comprehensive review of stacking methods for semantic similarity measuremen". ScienceDirect. 2022.

[10] Jurafsky, D. Martin, J. "Speech and Language Processing". Stanford University. 2025.

[11] Mendoza, C. Bocanegra, J. Garcés, K. Casallas, R. "Architecture violations detection and visualization in the continuous integration pipeline". Wiley Online Library. 2021

[12] Senanayake JKalutarage HAl-Kadri MPetrovski APiras L. "Android Source Code Vulnerability Detection: A Systematic Literature Review". Wiley Online Library. 2023

[13] Mazuera-Rozo, A. Esccobar-Velázquez, C. Espitia-Acero, J. Linares-Vázquez, M. Bavota, G. "CONAN: Statically Detecting Connectivity Issues in Android Applications". Wiley Online Library. 2023

[14] Li, R. Soliman, M. Liang, P. Avgeriou P. Warnings: Violation Symptoms Indicating Architecture Erosion. ArXiv. 2022.

[15] Juneja, S. Nauman, A. Uppal, M. Gupta, D. Alroobaea, R. Muminov, B. Tao, Y. "Machine learning-based defect prediction model using multilayer perceptron algorithm for escalating the reliability of the software". The Journal of SuperComputing. 2024.

[16] Devlin, J. Chang, M. Lee, K. Toutanova, K. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". ArXiv. 2018.

[17] Li, Z. Lu, S. Guo, D. Duan, N. Jannu, S. Jenks, G. Majumder, D. Green, J. Svyatkovskiy, A. Fu, S. Sundaresan, N. Automating code review activities by large-scale pre-training. ArXiv. 2022.

[18] Feng, Z. Guo, D. Tang, D. Duan, N. Feng, X. Gong, M. Shou, L. Qin B, Liu T. Jiang, D. Zhou, M. CodeBERT: A Pre-Trained Model for Programming and Natural Languages ArXiv. 2020.

[19] Yue, W. Weishi, W. Shafiq, J. Steven, C.H. Hoi. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. ArXiv. 2022.

[20] Zong, M. Krishnamachari, B. "A SURVEY ON GPT-3". ArXiv. 2022.

[21] Dabic, O. Aghajani, E. Bavota, G. "Sampling Projects in GitHub for MSR Studies", MSR'21.

[22] F-Droid. [Online] Available: https://f-droid.org/es/

[23] PyDriller. [Online] Available: https://pydriller.readthedocs.io/en/latest/

[24] Pennington. J. Socher, R. Manning, C. "GloVe: Global Vectors for Word Representation". Stanford. [Online] https://nlp.stanford.edu/projects/glove/

[25] Word2Vec. TensorFlow. [Online] www.tensorflow.org/text/tutorials/word2vec

[26] GPT-4o. OpenAI. [Online] https://platform.openai.com/docs/models/gpt-4o

[27] OpenAI libraries. [Online] https://platform.openai.com/docs/libraries

[28] Jean de Dieu, M. Liang, P. Shahin, M. Yang, C. Li, Z. "Mining architectural information: A systematic mapping study". Springer. 2024.

[29] Zan, D. Chen, B. Zhang, F. Lu, D. Wu, B. Guan, B. Wang, Y. Lou, J. "Large Language Models Meet NL2Code: A Survey". ArXiv. 2023.