Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Learning Risk Factors from App Reviews: A Large Language Model Approach for Risk Matrix Construction

Vitor Mesaque Alves de Lima (vitor.lima@ufms.br)
Federal University of Mato Grosso do Sul https://orcid.org/0000-0001-8721-2855
Jacson Rodrigues Barbosa
Goias Federal University https://orcid.org/0000-0002-4837-5632
Ricardo Marcondes Marcacini
University of São Paulo https://orcid.org/0000-0002-2309-3487

Research Article

Keywords: Opinion Mining, App Reviews, Issue Detection, Issue Prioritization, Risk Matrix

Posted Date: July 19th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3182322/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Learning Risk Factors from App Reviews: A Large Language Model Approach for Risk Matrix Construction

Vitor Mesaque Alves de Lima · Jacson Rodrigues Barbosa · Ricardo Marcondes Marcacini

Received: date / Accepted: date

Abstract Context. Analyzing mobile app reviews is essential for identifying trends and issue patterns that impact user experience and app reputation in app stores. A risk matrix provides a simple and intuitive way to prioritize software maintenance actions to reduce negative ratings. However, the manual construction of a risk matrix is time-consuming, and stakeholders work to understand the context of risks due to varied descriptions and review volume. **Objective.** There is a need for machine learning-based methods to extract risks and classify their priority. Existing studies have automated risk matrix generation in software development but have not explored app reviews or utilized Large Language Models (LLMs). **Method.** To address this gap, we propose using recent LLMs, specifically the OPT model, to automatically construct a risk matrix by extracting information from app reviews, such as features and bugs. We conduct experimental evaluations using reviews from eight mobile apps, generating risk matrices and comparing them with annotated reference matrices. **Results.** Results demonstrate that OPT models generate competitive risk matrices with proper prompt optimization. **Conclusions.** Our contributions include a dynamic and automatic prompt generation approach for customized instructions, allowing accurate and automated review analysis. We also develop instructions to identify risk severity

Vitor Mesaque Alves de Lima
Faculty of Computing (FACOM) - Federal University of Mato Grosso do Sul (UFMS)
University City, Três Lagoas, 79613000, Mato Grosso do Sul, Brazil
E-mail: vitor.lima@ufms.br

Jacson Rodrigues Barbosa
Institute of Informatics (INF) - Goias Federal University (UFG)
University City, Goiânia,74690900, Goias, Brazil
E-mail: jacson@inf.ufg.br

Ricardo Marcondes Marcacini
Institute of Mathematics and Computer Sciences (ICMC) - University of São Paulo (USP)
University City, São Carlos, 13566590, São Paulo, Brazil
E-mail: ricardo.marcacini@usp.br

using zero-shot learning. Additionally, we evaluate how OPT models compare to proprietary language models like GPT, showing the feasibility of LLMs in resource-constrained and sensitive contexts. This study represents a significant step toward improving software maintenance and feature prioritization.

Keywords Opinion Mining · App Reviews · Issue Detection · Issue Prioritization · Risk Matrix

Mathematics Subject Classification (2020) MSC 68T07 · MSC 68T50 · MSC 68N01 · MSC 68T35

1 Introduction

The analysis of mobile app reviews enables the identification of trends and issue patterns that can affect user experience and app reputation in app stores (Genc-Nayebi and Abran, 2017). Based on this analysis, developers can prioritize bug fixes, add requested features, and respond to user complaints to improve app quality and increase positive ratings. To achieve this, it is necessary to link user feedback extracted from reviews with app development and maintenance practices (Araujo et al., 2021).

A simple and intuitive way to organize and prioritize actions for software maintenance, aiming to reduce negative ratings, is through a risk matrix (Xiaosong et al., 2009; Pilliang et al., 2022). This matrix consists of a graphical representation where risks are positioned on a Cartesian plane based on their probability of occurrence and impact/severity, as illustrated in Figure 1. Risks are classified according to their importance and potential to harm app quality. Thus, it assists software engineering professionals in identifying the most critical areas that require prioritized attention. However, manual construction of a risk matrix often consumes a significant amount of time as stakeholders (Paltrinieri et al., 2019), such as project managers and product owners, need help understanding the context of risks recorded by the development team. For example, using different descriptions to report the same risk and the large volume of reviews make risk assessment challenging. Therefore, there is a need for automatic machine learning-based methods to extract risks from reviews and classify their priority.

Some initiatives in the literature already automate risk matrix generation using machine learning methods. For example, Chaouch et al. (2019) and Hammad and Inayat (2018) use Scrum with risk matrix, but with manual assessments. A recent study proposed using language models (e.g., BERT) and data clustering methods (e.g., K-Means) to automate risk matrix generation in software development projects (Pilliang et al., 2022). However, such studies have not yet explored the app review domain and rely on manually constructed resources, such as a vocabulary or lexicon to define risk priority. Another limitation is the lack of a step to extract app features from reviews, which is crucial for the development team.

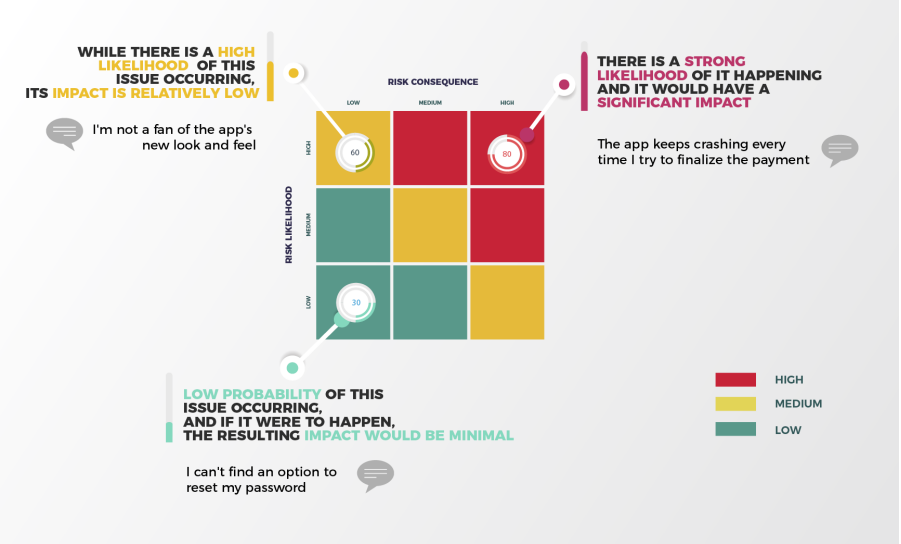

Fig. 1 Illustration of a risk matrix for app reviews.

To address this gap, we argue that recent Large Language Models (LLMs) are promising to automatically construct a risk matrix using information extracted from app reviews, such as features and bugs, organized according to the probability of occurrence and impact/severity to the app ratings. LLMs can analyze and understand complex contextual information present in review texts due to their extensive pre-training corpus (Ross et al., 2023).

In this context, we raise the following central research question: *how do we learn risk factors from app reviews using Large Language Models and prioritize app reviews and anticipate to mitigate risks?*

This paper presents a novel approach for generating Risk Matrix from App Reviews using Large Language Models, specifically the OPT model (Open Pre-trained Transformer Language Models) (Zhang et al., 2022). While large-scale language models like GPT (Brown et al., 2020) are widely used, we opted for OPT, an open-access language model. By providing specific instructions to the model through prompt engineering, it is possible to direct its attention to particular aspects of reviews, such as app features mentioned by users and the evaluation of risks' impact/severity associated with the apps. Our contributions are three-fold:

1. **Dynamic and automatic prompt generation:** We introduce an approach that enables the creation of customized instructions for each review to be analyzed, allowing the OPT model to extract app features as described by users in natural language. This enables more accurate and automated review analysis through few-shot learning, resulting in feature extraction with limited labeled data.

2. **Prompt instructions to identify risk impact:** We develop suitable instructions to automatically identify the severity or impact of risks mentioned in the reviews, classifying them into five levels: negligible, minor, moderate, major, and critical. In this case, we employ zero-shot learning, meaning there is no need to provide examples to the model.
3. **Evaluation of Open Pre-trained Large Language Models:** We evaluate how prompt engineering for OPT-based models compares to large proprietary language models such as GPT. By adopting OPT, we enable the use of large language models in scenarios with limited computational resources and constraints involving sensitive and private user data. This democratizes access to the usage of LLMs in more restricted contexts.

We conducted experimental evaluations using a database of reviews from eight mobile apps. Through the proposed approach, we constructed risk matrix for each app and compared them with their respective reference risk matrix constructed with annotated data. The experimental results demonstrate that, with proper prompt optimization, OPT models are capable of generating a competitive risk matrix compared to GPT. While there is room for improvement compared to reference risk matrices, our results indicate a significant step toward the maintenance and evolution of software products, enabling feature prioritization that requires more attention from developers.

The rest of this paper is structured as follows. Section 2 provides the background and discusses related works in the field. Section 3 presents the LLM-based risk matrix learning approach applied to app reviews with a focus on dynamic prompt construction for feature extraction (3.1), the estimation of review impact (3.2), and addresses the estimation of occurrence likelihood 3.3. Section 4 presents the experimental evaluation, defining the research questions (4.1), experiment definition (4.2), preparation and planning (4.3), sample selection (4.4), experimental package (4.5), variables (4.6), experimental design (4.7), operation of the experiment (4.8), results and discussion (4.9), and findings to research questions. Section 5 highlights the threats to validity of this study. Finally, Section 6 offers the concluding remarks of the paper, summarizing the main findings and contributions.

2 Background and Related Works

Early initiatives for analyzing reviews focusing on software maintenance explore named entity extraction techniques (such as software features) from reviews using predefined linguistic rules, such as Safe (Johann et al., 2017), GuMA (Guzman and Maalej, 2014), and ReUS (Dragoni et al., 2019). These approaches require sets of linguistic patterns, relying on experts to constantly update them. With the advancement of machine learning methods, models are trained to identify the entities of interest from a large set of labeled data. An example of such an approach is RE-BERT (Araujo and Marcacini, 2021), which uses annotated data to fine-tune a pre-trained BERT model to identify features or software requirement candidates mentioned by users in reviews.

With the recent emergence of Large Language Models (LLMs), such as GPT-3 and OPT (Zhang et al., 2022), opinion mining and sentiment analysis have also evolved to incorporate such models, although still with few applications in the context of app reviews. These models have architectures with billions of parameters and are pre-trained on large amounts of text, thereby providing capabilities for understanding and extracting knowledge from textual data. While the general aim of these models is text generation, their outputs can be conditioned through instructions or prompts (Strobelt et al., 2022). For example, the task of feature extraction or entity extraction from reviews can be performed using a paradigm called few-shot learning (Logan IV et al., 2022). In this case, the model receives a prompt that provides information about the type of feature to be extracted and a few samples related to that feature. It is worth noting that, in addition to feature extraction, the same LLM can also be used to identify the impact of reviews according to their severity level, where prompts can be used to guide the model in classifying the severity of the evaluations.

Another common strategy for app review analysis is the use of clustering methods or topic modeling based on review characteristics to organize them according to their similarities (Noei et al., 2021). This allows for identifying groups of reviews that mention similar problems, indicating the likelihood of bugs or complaints related to those issues. By combining feature extraction, severity classification, and the identification of the likelihood of app reviews, we can automate the construction of risk matrices (Pilliang et al., 2022). These matrices can provide an overview of the risks associated with an app based on the information extracted from user reviews.

A review of the existing studies reveals various aspects related to risk management in software projects. In Xiaosong et al. (2009); Chaouch et al. (2019); Hammad and Inayat (2018), the authors discuss risk management in agile software development projects. Xiaosong et al. (2009) presents basic risk management concepts for software development projects, while Chaouch et al. (2019) proposes a framework for integrating risk management in agile projects using Scrum. Additionally, Hammad and Inayat (2018) also explores the integration of risk management in the Scrum framework, highlighting the importance of an iterative risk management process for project success.

In Hammad et al. (2019) and Ionita et al. (2019), the authors focus on identifying the risks faced by agile development practitioners and mitigation strategies. Hammad et al. (2019) reveals that project deadlines and changing requirements are the most commonly encountered risks, while Ionita et al. (2019) proposes a framework that uses a risk assessment process to prioritize security requirements.

Pilliang et al. (2022) propose a risk matrix model for software development projects, using natural language processing and machine learning techniques to prioritize risks. The proposed model offers an approach for the automated construction of risk matrices by combining sentiment analysis based on lexicons or vocabularies to identify the impact of the risk and clustering methods to identify the likelihood of occurrence.

Although these studies address different aspects of risk management in software projects and the application of risk matrices, it is important to highlight some general limitations. Most studies focus on specific contexts, such as agile development or information security, which may limit the generalizability of their findings to other types of software projects. There is a gap in the literature on generating risk matrices from app reviews. Our focus is on generating a risk matrix based on reviews, especially in scenarios with little labeled data, which can be used in a wider range of applications at different stages of software development, from its conception to its maintenance.

3 LLM-based Risk Matrix Learning from App Reviews

The proposed approach leverages Large Language Models (LLMs) to extract relevant information from user reviews and utilize it in constructing a risk matrix. In general, the idea is to exploit the text generation capability of an LLM, but conditioned for a specific task.

Figure 2 provides an overview of the method used for constructing the risk matrix.

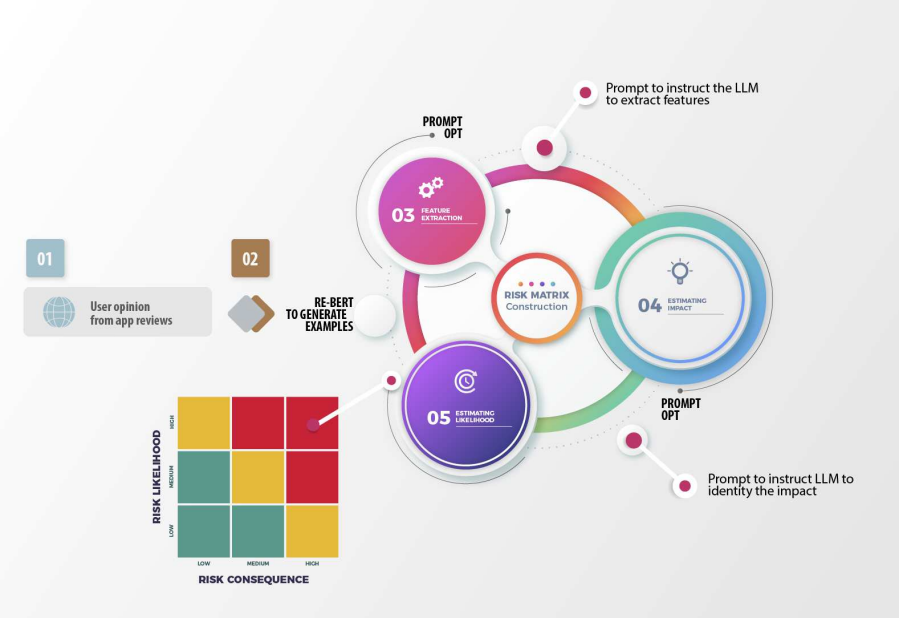

Fig. 2 The overview of our Risk Matrix construction method

Formally, the task of predicting the next word in an LLM can be formulated as finding the most likely word $P(w_{i+1}|w_1, w_2, ..., w_i)$ given a sequence of words $w_1, w_2, ..., w_i$. This probability is estimated using the neural weights

of the LLM, which is pre-trained on large textual corpora. The pretraining of the LLM is accomplished through a process called autoregressive language modeling in a Transformer neural architecture. During pretraining, the model learns to capture linguistic patterns and construct representations of words and phrases that can be used to predict the next word in a sequence.

Considering the context of software reviews as input, the proposed approach aims to condition the prediction of the next word through a prompt. Now, the conditioned probability $P(w_{i+1}|w_1, w_2, ..., w_i; \Theta)$, where Θ represents the prompt, is used to guide the prediction of the next word.

In the proposed method, we use the Open Pre-trained Transformers (OPT) as the Large Language Model (LLM). Most models available through APIs do not provide access to the full model weights obtained during pretraining, making it difficult to study them in detail and reproduce the experimental results. On the other hand, the OPT was developed to overcome this limitation by providing pre-trained models with different numbers of parameters. For example, models range from 125 million to 175 billion parameters. The authors of OPT conclude from their experiments that OPT-175B is comparable to GPT-3 (Zhang et al., 2022). However, some smaller models can achieve promising results through the appropriate use of prompts, as proposed in the next section.

3.1 Dynamic Prompt Construction for Feature Extraction

The first step of the proposed method involves the dynamic construction of prompts from a knowledge base of reviews from other apps, different from the target app, thereby avoiding the need for labeled data from the target application to be analyzed.

The knowledge base is represented through embeddings of reviews, which are numerical vectors that capture the semantics and context of words and phrases. These embeddings are obtained using deep learning algorithms, such as Sentence-BERT (Reimers and Gurevych, 2019), which map texts into vector representations in latent spaces. Formally, given a set of software reviews in the knowledge base, we can represent them as $R_1, R_2, ..., R_n$, where R_i represents a specific review.

Each review R_i is converted into a vector representation using a pre-trained embedding model. This representation is denoted as $e(R_i)$, where $e()$ represents the embedding function. In this way, we have a set of vectors representing the reviews in the knowledge base: $e(R_1), e(R_2), ..., e(R_n)$.

To retrieve the most similar reviews to a target review of interest, we employ the k-nearest neighbors technique. In this approach, we calculate the similarity between the embedding vector of the target review and the embedding vectors of all the reviews in the knowledge base. The similarity is commonly measured by the cosine of the angle between the vectors. Formally, to find the k-nearest neighbors of a target review R_i, we denote this list as $KNN(R_i)$ and define it as $KNN(R_i) = argmax_k(sim(e(R_i), e(R_k)))$, where

$sim()$ represents the similarity function and $argmax_k$ returns the k indices corresponding to the most similar reviews to R_i.

The k-nearest neighbors are then used to generate prompts related to the extraction of text snippets that describe software features. This nearest neighbors search approach allows the method to leverage the existing knowledge base and learn from similar examples, becoming a type of few-shot learning for the task of feature extraction from software reviews.

Figure 3 shows a prompt generated for the Instagram app. In blue are examples identified by similarity from the knowledge base generated through reviews and features of other applications. In red, it is the review to be processed. The model is induced to generate a list of features from the review after the "@" symbol.

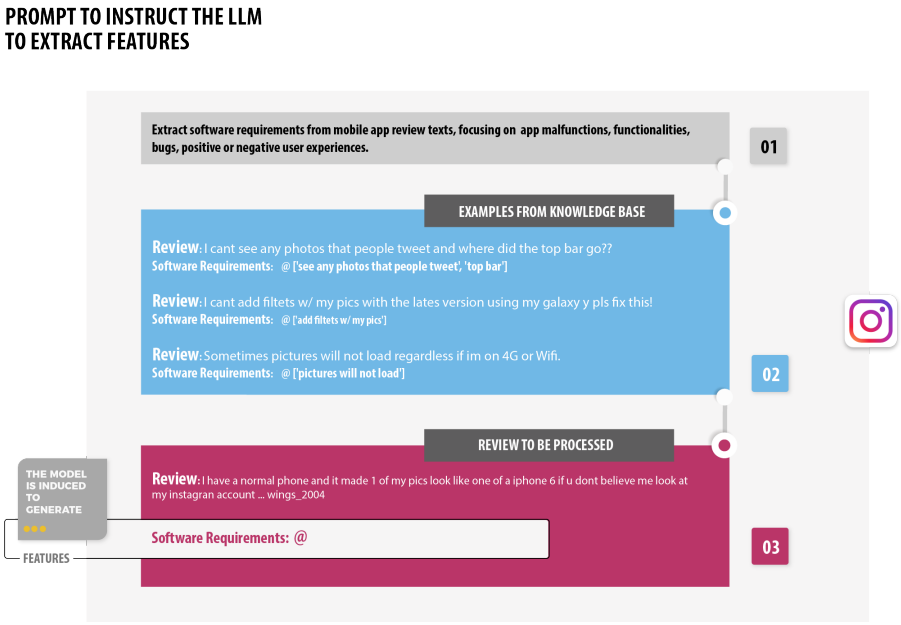

Fig. 3 Example of a prompt generated for the Instagram app

3.2 Estimating Review Impact

Building upon the previous step, we have a list of features extracted from software reviews. Thus, the second step of the method utilizes each extracted feature from the previous step into a prompt to instruct the LLM to identify the severity or impact on five levels: negligible, minor, moderate, major, and critical. Figure 4 presents an example of the prompt used. Note that we provide the prompt constructed along with the feature and let the model complete the

severity classification. Unlike the previous step, we condition the model to offer an answer within a limited set of options.

Fig. 4 Prompt generated to obtain the severity classification for the Instagram app

This zero-shot learning process enables the model to identify the severity of features even without receiving specific prior examples for each feature. Although the model has not been explicitly trained on specific examples of severity classification in software reviews, it is capable of inferring patterns and generalizing based on the information captured during model pre-training.

3.3 Estimating Occurrence Likelihood

While the first two steps allow mapping reviews onto the "impact" dimension of the risk matrix, the third step is responsible for mapping reviews onto the "occurrence likelihood" dimension. In this step, a graph-based strategy is employed.

The reviews and extracted features from the previous step are represented as textual expressions of interest and treated as vertices in a graph. Similar pairs of vertices are connected through edges. The similarity between the expressions is measured using embeddings and cosine similarity. In this case, consider a set of expressions extracted from software reviews, represented as

$E = \{t_1, t_2, ..., t_m\}$, where each t_i is an expression from the review containing the extracted feature. Similar to the first step, these expressions are converted into embeddings, which maps each expression to a feature vector.

The similarity between two embedding vectors is calculated using a metric such as cosine similarity. Let $sim(e(t_i), e(t_j))$ be the function that computes the similarity between two expressions. If the similarity value exceeds a predefined threshold, an edge is created between the corresponding vertices. Based on these similarities, we can construct the graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, as illustrated in Figure 5.

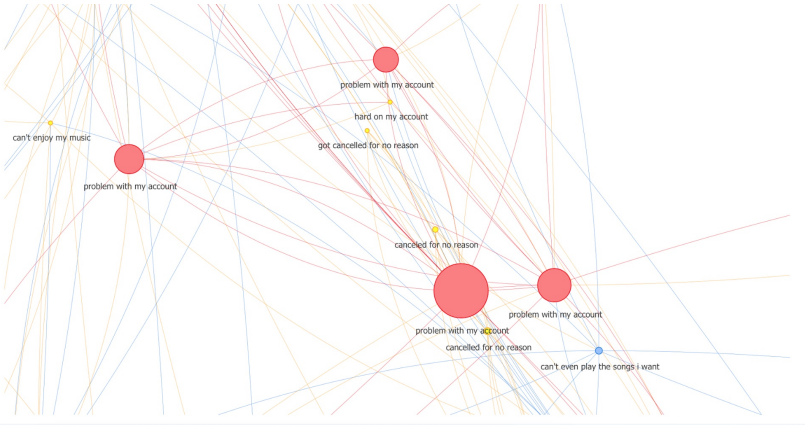

Fig. 5 Edge is created between the corresponding vertices if the similarity value exceeds a predefined threshold

The degrees of the graph's vertices identify expressions that have a higher likelihood of occurrence. The degree values are discretized into five levels representing different levels of occurrence likelihood. For this purpose, the discretization also considers the average degree of the graph, using this value for normalization following a normal distribution. This normalization allows mapping the node degree values onto a standardized scale. Using the mean and standard deviation of the degree values, the normal distribution function is applied, where values close to the mean have a higher probability and values farther from the mean have a lower probability.

Finally, the risk matrix is constructed considering the previous steps' impact and occurrence likelihood dimensions. The next section presents an experimental evaluation of the proposed approach.

4 Experimental Evaluation

We conducted an experiment to evaluate the approach presented in this paper. To do so, we followed the guidelines proposed by Wohlin et al. (2012). The experimental design is detailed in the remainder of this section.

4.1 Definition of Research Questions

Our central research question is: *how do we learn risk factors from app reviews using Large Language Models and prioritize app reviews and anticipate to mitigate risks?*

To answer this main question, we divided it into two specific research questions, as follows:

- **RQ1**: How can we extract features using LLMs with limited labeled data?
- **RQ2**: How can we identify the severity or impact of risks mentioned in the reviews and automatically organize them into a risk matrix?

The experiment was conducted to address the research questions presented in this section.

4.2 Experiment Definition

The experiment is defined as follows (Wohlin et al., 2012):

- analyze risk matrix construction method using LLM,
- to evaluate feature extraction, impact estimation, and likelihood estimation,
- with respect to model performance,
- from the point of view of the researcher,
- in the context of crowd feedback from app user reviews.

4.3 Preparation and Planning

The experiment plan comprises the sample selection, description of the experimental package, definition of variables, and description of employed design principles.

To evaluate the performance of the risk matrix construction approach, we compare our strategy with other state-of-art approaches.

4.4 Sample Selection

To evaluate the risk matrix construction approach, we selected app review datasets used in previous studies of review mining (Dabrowski et al., 2020). In this context, we utilized human-labeled data consisting of reviews, app features, and corresponding sentiment to generate the impact of a risk matrix.

We used eight mobile apps from these datasets, as described in Table 1. We included apps from different categories to enhance the generalizability of our results. The ground truth consists of 1,000 reviews for the eight analyzed apps, with 1,255 distinct features, meaning they are mentioned only once, making extracting app features from reviews more challenging.

Table 1 The overview of the datasets used for automatic risk matrix construction.

App	Reviews	Labeled reviews	Sentences	Features	Distinct features
eBay	1,962	125	294	206	167
Evernote	4,832	125	367	295	259
Facebook	8,293	125	327	242	204
Netflix	14,310	125	341	262	201
Photo editor	7,690	125	154	96	80
Spotify	14,487	125	227	180	145
Twitter	63,628	125	183	122	99
WhatsApp	248,641	125	169	118	100

4.5 Experimental Package

In our experiment, we have three objects:

- feature extraction;
- impact estimation; and
- likelihood estimation.

The following components are part of the experiment package:

- Reference dataset description: Our experiment utilized human-labeled data containing reviews, app features, and sentiment. This dataset was obtained from Dabrowski et al. (2020).
- Object definition: We use Python programming to define each object.
- Machine learning classification methods: We compared our proposed OPT-based approach with three rule-based methods (GuMa, SAFE, ReUS), a fine-tuning method of language models (RE-BERT), and a large language model (GPT 3.5).

4.6 Variables

The independent variables (factors) controlled in the experiment were feature extraction and risk estimation, and their respective treatments are described below.

4.6.1 Feature Extraction Factor

The values assigned to this variable (treatments) are six classifiers (GuMa, SAFE, ReUS, RE-BERT, GPT, and our OPT-based Proposal). The dependent variable, which is affected by the treatment, is (i) the F1-Score, a measure of the accuracy of a test (RQ1), and (ii) MAPE / MAE employs typical measures from the regression field for prediction error (RQ2).

To evaluate the feature extraction step, we use the F1 measure for feature matching, as proposed by (Dabrowski et al., 2020), that corresponds to the harmonic mean of Precision (1) and Recall (2), where TP (True Positive)

refers to the number of features that were both extracted and annotated; FP (False Positive) are features that were extracted but not annotated, and FN (False Negative) refers to the features annotated but not extracted. Equation 3 defines the F1 measure.

This measure allows us to assess the precision and recall of feature extraction in relation to annotated reference features. The parameter n of the Feature Matching allows for flexible matching, where $n = 0$ indicates exact matching, while $n > 0$ represents the difference between the sizes of the extracted and labeled sequences. We used $n = 2$ in the experimental evaluation.

$$P = \frac{TP}{TP + FP} \tag{1}$$

$$R = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 * P * R}{P + R} = \frac{2 * TP}{2 * TP + FP + FN} \tag{3}$$

4.6.2 Risk Estimation Factor

The risk matrix is evaluated in the impact dimension by comparing the numerical level of the reference impact with the impact estimated by our method. For this evaluation, we employ typical measures from the regression field, such as the Mean Absolute Percentage Error (MAPE) and the Mean Absolute Error (MAE), as defined in Equation 4 and 5 respectively,

$$MAPE = \frac{1}{n} \sum_{t=1}^{n} \frac{|real_t - pred_t|}{real_t} \tag{4}$$

$$MAE = \frac{\sum_{t=1}^{n} |real_t - pred_t|}{n} \tag{5}$$

where $real_t$ is the real value and $pred_t$ is the predicted value by the method, and n is the sample size.

4.7 Experimental Design

In this experiment, the experimental design has two factors (feature extraction and risk estimation) with the following treatments (Wohlin et al., 2012):

- Feature extraction treatments: GuMa, SAFE, ReUS, RE-BERT, GPT, and OPT-based approach.
- Risk estimation treatments: GPT and OPT-based approach.

For the step of extracting features from reviews of applications, we compared the proposed OPT-based approach with three rule-based methods (GuMa, SAFE, ReUS), a fine-tuning method of language models (RE-BERT), and a large language model (GPT 3.5).

Concerning the second aspect of evaluation, we compared the proposed approach with GPT. In this scenario, both models operate in the zero-shot learning format.

4.8 Operation of the Experiment

Concerning the step of extracting features, GuMa performs feature extraction using a collocation search algorithm, which identifies commonly used expressions of two or more words that convey a specific meaning through co-occurrence-based measures. SAFE, relies on manually identified linguistic patterns, including patterns of parts of speech and sentences, to extract features from applications. ReUS utilizes linguistic rules composed of patterns of parts of speech and semantic dependency relations. These rules allow for simultaneous feature extraction and sentiment analysis. To determine sentiment, the method employs lexical dictionaries. In contrast, RE-BERT uses pre-trained language models to generate semantic textual representations, focusing on the local context of software requirement tokens. However, RE-BERT is a supervised learning method, i.e., it requires a labeled dataset for model training.

The proposed method employs the OPT-6.7b model, which contains 6.7 billion parameters, for the step of extracting features from application reviews. In this step, we employ the proposed strategy of dynamically generating prompts. We use a cross-domain approach, where reviews from other applications (source apps) are used as a knowledge base to generate specific prompts for each review of the target app. This cross-domain approach with dynamic prompt generation allows the model to be fed with information from related applications, expanding its ability to generalize without requiring prior knowledge about a specific target app. To perform this prompt generation, we use the Sentence-BERT (Reimers and Gurevych, 2019) embedding model with $k = 10$ to compute the nearest neighbors based on cosine similarity.

For identifying the impact of the feature-review pair, our proposal utilizes the OPT-IML-1.3b model (Iyer et al., 2022). This model results from a fine-tuning process of large pre-trained language models on a collection of tasks described through instructions, also known as instruction-tuning. This process aims to improve the generalization capability of these models for previously unseen tasks. Our proposal employs the zero-shot learning strategy in this step. This means that the model is capable of learning to identify the impact of a feature-review pair, even without receiving specific examples of this relationship during training.

Finally, we also use GPT 3.5 as the reference model, which is another pre-trained language model, but in the category of Large Language Models. It is used for both the extraction of features from reviews and the identification of

review impact. We employ a zero-shot learning strategy by providing instructions and only examples of how the extraction output should be formatted.

4.9 Results and Discussion

The experimental results are analyzed considering two main aspects: (1) the performance of the F1 score in the matching of feature extraction from app reviews, and (2) the error (MAPE and MAE) in constructing the risk matrix, particularly in the impact dimension. The likelihood dimension in the reference risk matrices was obtained in the same way as the proposed method. Hence there are no significant variations for comparison.

Regarding the first aspect, we analyze the proposed dynamic prompt generation for OPT and the few-shot prompt learning, compared to a supervised reference approach based on RE-BERT and classical rule-based methods. The aim is to demonstrate the performance of OPT models in the absence of labeled data and the generalization capability of LLMs for new tasks and domains. Table 2 presentes an overview of the experimental results in the task of extracting features from application reviews.

Table 2 Comparison of approaches GuMa, SAFE, ReUS, RE-BERT, GPT (zero-shot learning) and OPT (Proposal with few-shot learning) for feature extraction from app reviews.

APP	**F1 Matching Score (n = 2)**					
	GuMa	**SAFE**	**ReUS**	**RE-BERT**	**GPT**	**Proposal**
eBay	0.22	0.36	0.21	0.53	0.22	0.39
Evernote	0.24	0.33	0.28	0.63	0.14	0.46
Facebook	0.19	0.24	0.19	0.61	0.20	0.40
Netflix	0.21	0.28	0.27	0.62	0.23	0.41
PhotoEditor	0.28	0.34	0.26	0.81	0.32	0.56
Spotify	0.28	0.35	0.27	0.60	0.17	0.48
Twitter	0.27	0.35	0.26	0.67	0.25	0.47
WhatsApp	0.26	0.39	0.24	0.61	0.18	0.47

We observed that the proposed approach achieves superior results compared to rule-based methods but inferior results to the supervised RE-BERT model. However, it is important to note that supervised models require a significant amount of annotated data, necessitating the annotation of all features in each review of the training set for a model generation — a very time-consuming task. Although this strategy shows promising results, it may not be feasible in scenarios with a lack of domain experts or in dynamic settings with frequent review updates, which is common in mobile application quality monitoring and maintenance through reviews.

Our approach yielded promising results compared to the proprietary GPT model with zero-shot learning. In addition to requiring less labeled data than fully supervised models, our few-shot prompt learning strategy is based on

open models, without restrictions on proprietary APIs or limitations on processing private or sensitive data.

Concerning the second aspect of evaluation, we compared the proposed approach with GPT. In this scenario, both models operate in the zero-shot learning format. However, it should be noted that we used OPT-IML (instruction meta-learning), which is fine-tuned with hundreds of instructions but with a smaller number of parameters. In this case, the utilized OPT-IML model has 1.3 billion parameters, and we analyzed the risk matrices generated with the features extracted from the previous step. As illustrated in Table 3, OPT-IML exhibits a lower error in constructing the risk matrix in the impact dimension, highlighting it as a promising alternative compared to the proprietary GPT model.

Table 3 Error comparison in the Risk Matrix construction.

APP	**GPT-3.5**		**OPT-IML**	
	MAE	**MAPE**	**MAE**	**MAPE**
eBay	1.000	0.517	0.913	0.335
Evernote	1.053	0.532	1.160	0.388
Facebook	1.092	0.331	0.965	0.285
Netflix	1.255	0.446	1.061	0.359
PhotoEditor	0.957	0.443	0.929	0.301
Spotify	1.034	0.344	0.840	0.244
Twitter	0.943	0.267	0.971	0.253
Whatsapp	1.133	0.378	1.120	0.357

Figure 6 illustrates a risk matrix constructed automatically with the proposed approach for Netflix app. Note that reviews and app features categorized as critical impact and with a high likelihood of occurrence are frequent complaints with a strong negative sentiment. This occurs because the model identifies that such complaints have a greater severity on the application's reputation, meaning they directly affect the app's ratings in the store.

In summary, the experimental results suggest that open and accessible Large Language Models (LLMs) can play an important role in developing automated tools for analyzing mobile application reviews, facilitating risk identification, as well as contributing to monitoring and prioritizing software maintenance tasks.

4.10 Findings to Research Questions

Returning to the initial primary research question: *"How do we learn risk factors from app reviews using Large Language Models and prioritize app reviews and anticipate to mitigate risks?"*. We addressed it by formulating two specific questions (see Section 4.1) and answering them based on the results obtained from the conducted experiment.

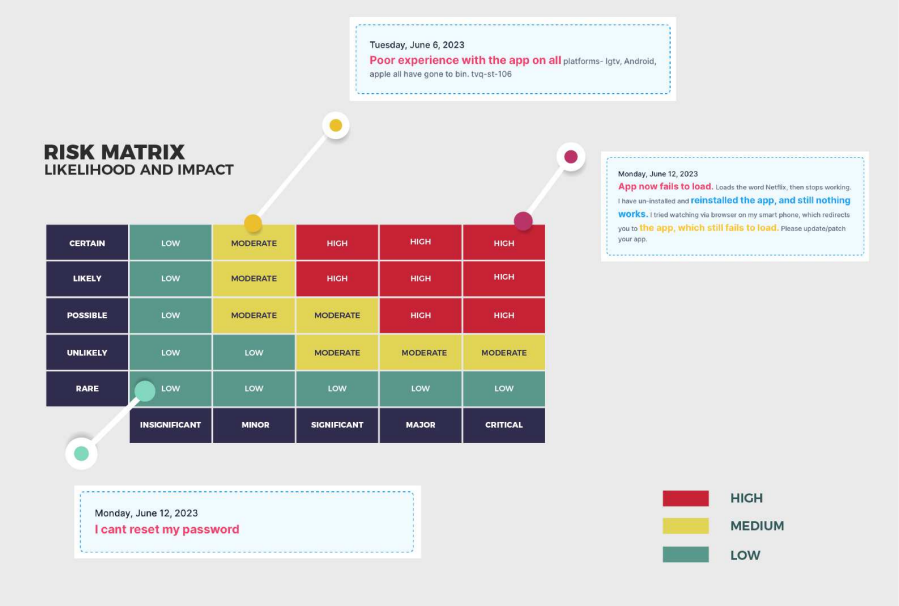

Fig. 6 Risk matrix constructed automatically with the proposed approach for Netflix app

We present objective answers to each research question below (RQ1 and RQ2), highlighting the main findings:

- **(RQ1) Feature extraction with limited labeled data.** We introduced the analysis and classification of app reviews using Large Language Models (LLMs) with limited computational resources and data privacy constraints. We incorporated a dynamic and automatic prompt generation technique to extract specific app characteristics mentioned in the reviews. The experimental results showed that our OPT-based proposed approach is superior to rule-based and has advantages over supervised methods that require a significant amount of labeled data. The findings demonstrated that our proposal is competitive compared to large proprietary language models like GPT-3.5. Despite potential areas for improvement, the findings suggest significant progress in extracting features using LLMs with limited labeled data.
- **(RQ2) Automated risk matrix construction.** We developed and evaluated instructions to classify the severity or impact of risks mentioned in app reviews. These instructions served as prompts to guide our Large Language Model (LLM) in categorizing the risks into five levels: negligible, minor, moderate, major, and critical. The experimental results indicated that our proposed approach, using LLMs, effectively identified the severity or impact of risks mentioned in the reviews and organized them into a risk matrix. Our research reveals a significant advancement in the field of software product maintenance and evolution using LLMs.

5 Threats to Validity

5.1 Internal Validity

Potential threats to internal validity include the quality and representativeness of the knowledge base of app reviews used for prompt construction. If the knowledge base is biased or lacks diversity, it may impact the effectiveness of the method. Additionally, the classification of risk severity based on the app rating may introduce subjectivity and potential errors. To mitigate these threats, we utilized review datasets from apps in different domains. We also employed a cross-domain validation strategy, where we used datasets to generate prompts from reviews of apps that are different from the target app being analyzed.

5.2 Construct Validity

Construct validity could be threatened if the classification of risk severity based on the app rating does not truly reflect the impact of the risks. In this study, our assumption is that one of the main risk factors for the app is actions that impact its reputation (average overall rating) in the app store and, therefore, increase the chances of being uninstalled or not even installed by users. To enhance construct validity, further research should focus on evaluating other types of risks associated with apps, such as security and malfunctions that affect users' smartphones.

5.3 External Validity

The external validity may be limited by the specific LLM used (OPT model) and the dataset of app reviews. Different LLM architectures or datasets from other domains may yield different results. The effectiveness of the approach may also vary depending on the characteristics of the target app. To improve external validity, conducting replication studies using different LLMs and diverse datasets from various app domains would be valuable. User studies or obtaining feedback from software engineering professionals would also help validate the practical usefulness and effectiveness of the risk matrices generated from app reviews.

6 Final Remarks

We introduced the analysis and classification of app reviews through a risk matrix using Large Language Models (LLMs), specifically open-access methods suitable for scenarios with limited computational resources and data privacy constraints, in contrast to proprietary models like GPT.

Our proposed approach incorporates a dynamic and automatic prompt generation technique, enabling the extraction of specific application characteristics mentioned by users in the reviews. We also developed and evaluated instructions to classify the severity or impact of the risks mentioned in the reviews. These instructions serve as prompts to guide the LLM in classifying the risks into five levels: negligible, minor, moderate, major, and critical. This standardizes the risk assessment and facilitates the automatic construction of the risk matrix. The experimental results provide evidence that the proposal, through Open Pre-trained Transformers (OPT), is competitive compared to large proprietary language models such as GPT-3.5. Although there is room for improvement regarding the risk matrices generated by supervised reference models, our results indicate a significant advancement in software product maintenance and evolution.

Direction for future work involves the development of tools to support decision-making, visualization, and monitoring associated with the risk matrices. These tools would provide actionable insights and facilitate the interpretation of the risk assessment results. Additionally, exploring techniques to enhance the accuracy and granularity of the risk classification levels could further improve the effectiveness of the risk matrix approach. Furthermore, investigating the integration of external data sources, such as social media and user forums, could provide additional context and insights into app-related risks. The development of part of these tools is addressed and better discussed in the next paper.

Acknowledgements This study was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) [process number 426663/2018-7], Federal University of Mato Grosso do Sul (UFMS), São Paulo Research Foundation (FAPESP) [process number 2019/25010-5 and 2019/07665-4], Artificial Intelligence (C4AI-USP), and from the IBM Corporation.

Availability of data and materials

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Authors' Contributions

V.L., J.B., and R.M. designed the experiment. V.L. and R.M performed the experiments and analyzed the data. V.L. and R.M. wrote the code. V.L. prepared the figures and tables. V.L., J.B., and R.M. drafted the work and revised it critically for important content.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Araujo, A. and Marcacini, R. M. (2021). Re-bert: Automatic extraction of software requirements from app reviews using bert language model. In *The 36th ACM/SIGAPP Symposium On Applied Computing*.

Araujo, A. F., Gôlo, M. P. S., and Marcacini, R. M. (2021). Opinion mining for app reviews: an analysis of textual representation and predictive models. *Automated Software Engineering*, 29(1):5.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Chaouch, S., Mejri, A., and Ghannouchi, S. A. (2019). A framework for risk management in scrum development process. *Procedia Computer Science*, 164:187–192.

Dabrowski, J., Letier, E., Perini, A., and Susi, A. (2020). Mining user opinions to support requirement engineering: An empirical study. In Dustdar, S., Yu, E., Salinesi, C., Rieu, D., and Pant, V., editors, *Advanced Information Systems Engineering*, pages 401–416, Cham. Springer International Publishing.

Dragoni, M., Federici, M., and Rexha, A. (2019). An unsupervised aspect extraction strategy for monitoring real-time reviews stream. *Information Processing and Management*, 56(3):1103–1118.

Genc-Nayebi, N. and Abran, A. (2017). A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software*, 125:207–219.

Guzman, E. and Maalej, W. (2014). How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)*, pages 153–162. IEEE.

Hammad, M. and Inayat, I. (2018). Integrating risk management in scrum framework. In *2018 International Conference on Frontiers of Information Technology (FIT)*, pages 158–163. IEEE.

Hammad, M., Inayat, I., and Zahid, M. (2019). Risk management in agile software development: A survey. In *2019 International Conference on Frontiers of Information Technology (FIT)*, pages 162–1624. IEEE.

Ionita, D., van der Velden, C., Ikkink, H.-J. K., Neven, E., Daneva, M., and Kuipers, M. (2019). Towards risk-driven security requirements management in agile software development. In *Information Systems Engineering in Responsible Information Systems: CAiSE Forum 2019, Rome, Italy, June 3–7, 2019, Proceedings 31*, pages 133–144. Springer.

Iyer, S., Lin, X. V., Pasunuru, R., Mihaylov, T., Simig, D., Yu, P., Shuster, K., Wang, T., Liu, Q., Koura, P. S., et al. (2022). Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*.

Johann, T., Stanik, C., Maalej, W., et al. (2017). Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE*

25th International Requirements Engineering Conference (RE), pages 21–30. IEEE.

Logan IV, R., Balažević, I., Wallace, E., Petroni, F., Singh, S., and Riedel, S. (2022). Cutting down on prompts and parameters: Simple few-shot learning with language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2824–2835.

Noei, E., Zhang, F., and Zou, Y. (2021). Too many user-reviews! what should app developers look at first? *IEEE Transactions on Software Engineering*, 47(2):367–378.

Paltrinieri, N., Comfort, L., and Reniers, G. (2019). Learning about risk: Machine learning for risk assessment. *Safety science*, 118:475–486.

Pilliang, M., Munawar, Hadi, M. A., Firmansyah, G., and Tjahjono, B. (2022). Predicting risk matrix in software development projects using bert and k-means. In *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 137–142.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3973–3983.

Ross, S. I., Martinez, F., Houde, S., Muller, M., and Weisz, J. D. (2023). The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 491–514.

Strobelt, H., Webson, A., Sanh, V., Hoover, B., Beyer, J., Pfister, H., and Rush, A. M. (2022). Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. *IEEE transactions on visualization and computer graphics*, 29(1):1146–1156.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Norwell, MA, USA.

Xiaosong, L., Shushi, L., Wenjun, C., and Songjiang, F. (2009). The application of risk matrix to software project risk management. In *2009 International Forum on Information Technology and Applications*, volume 2, pages 480–483. IEEE.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. (2022). Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.