Using Large Language Models to Classify Test Case Complexity with Explainability

Tiago Custódio
Universidade Federal do Amazonas
Manaus, AM, Brazil
tpc@icomp.ufam.edu.br

André Carvalho
Universidade Federal do Amazonas
Manaus, AM, Brazil
andre@icomp.ufam.edu.br

Maikon Santos
Universidade Federal do Amazonas
Manaus, AM, Brazil
maikon.santos@icomp.ufam.edu.br

Yan Soares
Universidade Federal do Amazonas
Manaus, AM, Brazil
yan.soares@icomp.ufam.edu.br

Hallyson Melo
Instituto de Desenvolvimento Tecnológico
Manaus, Amazonas, BR
hallyson.melo@indt.org.br

Nikson Ferreira
Instituto de Desenvolvimento Tecnológico
Manaus, Amazonas, BR
nikson.ferreira@indt.org.br

Rodrigo Marques
Instituto de Desenvolvimento Tecnológico
Manaus, Amazonas, BR
rodrigo.marques@indt.org.br

ABSTRACT

The classification of black-box test case complexity is a key task in software testing, enabling resource prioritization and the identification of edge scenarios. In this work, we propose a three-stage LLM-based pipeline that integrates explanation generation into the classification process, treating justifications as central to model decision-making. Our approach formulates the task as a conditional generation problem, where LLMs are guided to first produce a rationale and then a complexity label. Experimental results demonstrate that this strategy improves both predictive accuracy and explainability compared to using LLMs directly for classification. We show that LLM-generated justifications not only enhance user trust but also contribute to more consistent and explainable decisions.

KEYWORDS

Test Case Complexity, Large Language Models, Prompt Engineer

1 Introduction

Classifying the complexity of software test cases is an important task in software engineering, as it guides critical activities such as test prioritization, resource allocation, and the identification of edge-case scenarios [3], [11]. Assigning test cases to predefined complexity categories enables more efficient testing strategies, since complex test cases can be prioritized or receive additional resources, supporting practices such as parallel execution, risk-based selection, and test suite minimization.

Recent advances in Large Language Models (LLMs), such as GPT-4[1], LLaMA 2[15], and Gemma[14], have demonstrated remarkable capabilities in classification, reasoning, and text generation tasks [5, 18]. These models not only offer strong predictive power but may also generate natural language justifications that explain the reasoning behind their decisions, making them promising candidates for applications requiring interpretability [21], [8].

Despite the enthusiasm surrounding the use of LLMs in complex tasks, few studies have addressed test case complexity classification with a simultaneous focus on classification quality and explainability. This reveals an important gap in the literature, especially in high-stakes domains like software engineering, where understanding the rationale behind predictions is as crucial as the predictions themselves [8], [13]. Research in Explainable AI (XAI) emphasizes that intelligent systems should not only predict but also justify their outputs, allowing users to audit and trust automated decisions [8].

In this work, we propose a novel approach for test case complexity classification that leverages the predictive and generative capabilities of LLMs. Specifically, we are interested in the execution complexity of black box test cases, i.e., the effort, resources and level of knowledge that a Quality Assurance specialist would spend to perform the test case. We introduce a three-stage pipeline that integrates LLMs in a structured manner to enhance both classification accuracy and decision transparency. Unlike traditional methods that treat justifications as optional by-products, our method makes explanations central to the decision-making process. The research is guided by the following question: Can justifications generated by LLMs measurably and reliably improve test case complexity classification?

We formulate the task as a conditional generation problem, where the input (a test case along with structured instructions) leads to the generation of a list of features that may impact complexity, followed by the generation of a justification of why and how those features would impact it, followed by a classification into low, medium, or high complexity and a summary of the rationale for the classification. Structured prompts guide the LLM through the interpretation, reasoning, and decision phases, ensuring consistency between the rationale and the final prediction [18], [7].

We evaluated our pipeline on a real dataset of black box software test cases and compared its performance with direct LLM classification without justifications. Results show measurable gains in classification accuracy, with clear and coherent justifications.

Our main contributions can be summarized as follows: (1) **Proposal of an explainable pipeline**: a hierarchical framework that integrates LLMs into the task of complexity classification, placing natural language justifications at the core of the decision-making process. (2) **Balance between accuracy and interpretability**: empirical evidence that LLM-generated justifications can improve prediction quality while supporting user understanding. (3) **Structured prompting strategy**: a conditional generation process that aligns reasoning and classification by guiding the model through sequential reasoning steps.

2 Related Work

Text classification is a well-known task in natural language processing research [16]. Recently, many works have depploved LLMs successfully to enhance this task, with and without fine tuning Regarding approaches which use fine-tuning methods, Zhang et al[21] proposed a framework which explores the use of LLMs in the context of text classification. This framework was built in two stages: first, an ensemble of *base learners*, each trained using misclassified documents of the previous learner, aiming to enhance the results of the next learner; the second stage, called *recurrent learners*, incorporates the percentage error of the previous learner and misclassified class in the *prompt* of the current learner. All this process may suffer from overfitting due to oversampling the same data to learners, but the authors made experiments showing that there was no overfitting. Another work that made use of oversampling as an approach to text classification is [4] that investigates the performance of the LLM by resampling techniques.

Some approaches try to avoid oversampling data or do not make use of it. In Wang et al [17] the authors proposed a framework that aggregates data from different sources, inserting it in a huge pipeline. The authors do not train or fine-tune the LLM, instead using a few-shot strategy to classify and a human expert in the pipeline to change the prompt to improve results. Wu et al[20] present the use of an LLM to extract the embeddings and train a classifier through classic machine learning techniques. A classification self-regularization framework was proposed by [19] to extract and constrain unintended features in LLM latent space. To identify the unintended features in latent space, the authors made use of the LLMs own features explanations.

Some approaches deal with low data availability and oversampling by using data augmentation techniques. Moller et al [12] investigates the improvement of LLM performance in text classification through data augmentation to classify complex tasks with few available data.

In our work, due to the gap of data availability, in section 4.1 we provide more details about our dataset. To avoid data oversampling, which results may lead to overfitting, we made use of data augmentation. The data augmentation was realized through LLMs to generate synthetic training examples. Furthermore, to improve our dataset, we made the explainability strategy [22] present throughout our architecture.

3 Proposed Method

In this section, we describe the test case complexity classification Task and our proposed architecture.

3.1 Complexity

Carrying out test case complexity classification is important in the process of prioritization and assigning of Quality Assurance (QA) specialists to perform the test case, based on their experience and knowledge of the tested software.

The complexity in our context describes the effort a specialist may spend to execute a Test Case. This effort can be viewed as the time spent to complete some of the steps present in the Test Case, the knowledge required by the specialist, and/or the autonomy in decision making.

In our tested scenario, as is common in the literature, Test Case Complexity is divided into 3 levels, Low, Medium, or High. To do so, in this work, we consider four aspects regarding the test case: Ambiguity, Domain Expertise, Logical Paths, and Problem-Solving.

The **Ambiguity** [9] regards how clear and accurate the instructions are in the Test Case. More ambiguity requires implicit knowledge from the QA analyst if the instructions are vague and not precise, which can make the execution of the Test Case lead more to interpretations than just following instructions. **Domain Expertise** [9] is the knowledge of the product and the business rules necessary to execute the Test Case. In this category, the requirements of the QA analyst correspond to how familiar and specialized they are with the complex domains of the Test Case. **Logical Paths** is related to programming logic applied to graphs. This category requires, from the QA analyst, programming logic to perform the control of the flow graph, logic for multi-conditional statements, and critical sense for analysis and to prevent mistakes and errors in the flow graph. **Problem Solving** [9] is related to the QA analyst's autonomy to treat unexpected behaviors when performing a Test Case. It requires debugging, solving complex issues, and deep analysis to prevent similar future errors. The table 1 shows the complexity requirements for each category in our context.

However, an important rule in the classification of the Test Case complexity is that we always consider first the highest category present to classify the complexity, that is, if we have one category classified as High, the complexity of the whole test case is High, even if the other 3 categories are Medium or Low. Similarly, if the highest category is Medium, it will be regarded as Medium even if it is Low in all categories.

3.2 3-level LLM-Based Complexity Classification

We propose an architecture to enhance the use of LLM models to classify the Test Case complexity. In our architecture, illustrated in Figure 1 the LLM model performs 3 different functions and is divided into 3 levels. In the Features Extractor Level, the LLM needs to learn how to extract the main features of the Test Case that might impact complexity, based on fine tuning using an annotated Test Case dataset. In the next Level, Justification Level, the LLM generates the justification regarding how the features extracted in the previous level might impact the Test Case Complexity. And finally, the LLM model in Classifier Level is responsible for learning

Complexity	Ambiguity	Domain Expertise	Logical Paths	Problem Solving
High	Instructions are vague and requires implicit knowledge.	Specialized knowledge and complex business logic.	Programming logic to perform graph control, multi-conditional statements and analyze any path that is not explicit defined.	Solve and debugging complex issues and analysis to prevent errors in the future.
Medium	Instructions are clear and require some interpretation.	Familiarity with specific system modules and basic to moderate business logic.	Programming logic to perform graph control, simple multi-conditional statements.	Solve and debugging minor issues. May handle analysis.
Low	Instructions are precise without interpretation.	Common knowledge domain without any specialized system knowledge.	Programming to handle simple conditional statements.	No debugging.

Table 1: Complexity Categories and the criterias for high, medium and low labels for each category.

and classifying the complexity of a Test Case and issuing a summary of the reasons behind such classification.

The LLM performs a different role for each Level resulting in the complexity classification. To perform this final task, we constructed different Prompts for each Level in our architecture. The construction of our prompts are divided into three steps: System Prompt, User Prompt, Input.

The System Prompt, see Table 2, explains the role that the LLM should take for the specific level and how it behaves itself in broad terms. The User prompt, in the Table 3, describes the task at hand at that specific level, including restrictions and output format, and the Input is the test case to be classified along with information extracted from the previous levels.

3.3 Feature Extractor Level

The *Feature Extractor Level* was designed to generate the features of the Test Case. The features is a list with the main features that classify the Test Case complexity. These main features were annotated by specialists with the necessary knowledge to classify the complexity, given an arbitrary Test Case, and the understanding of what makes a Test Case more complex than others.

In our studies and experiments, we concluded that the main features increase the LLM model quality. However, this is other information that our LLM model must learn to generate because, thinking on test or production data, this information will not be provided. As we can see in Figure 1, in the Features Extractor Level the provided dataset has 4 fields: Summary, Steps, Initial Setup and Expected Results, see section 4.1 for more details of each field.

Therefore, in the Features Extractor Level of our architecture, we realize a fine-tuning of our LLM model. To perform the fine-tuning, we made use of the Quantized Low-Rank Adaptation (QLoRA) technique, which uses adapters on training. It is important to note that these adapters will be our "new weights", in other words, we do not change the original LLM model parameters in any Level of our architecture. In the end of the Features Extractor Level, our model generates the features, which enhances our raw dataset to *New Dataset - 1*. The Table 3 shows the prompt used to guide our LLM model to generate the features of a Test Case.

3.4 Prompt Construction Justification Level

At the Justification Level, the LLM model generates the Test Case justification, which synthesizes the Test Case itself. The main difference between the human-extracted features (from the Feature Extractor) and the justifications is that the former are created by human specialists, while the latter are generated by the LLM.

We consider the justifications generated by the LLM as "AI-features", meaning they represent the main features of the Test Case from the LLM's perspective at this Justification Level.

Initially, the idea was for the LLM to synthesize the entire Test Case. However, after experiments and analyzing the results, we concluded that the Test Case justification, synthesized by the LLM, improved the results when used in conjunction with the complete Test Case data as input.

The System Prompt also assigns a role to the LLM, as in the Feature Extraction Level. The User Prompt instructs the LLM to output a list of features, with a maximum of 5 words per feature. This limit was set to prevent LLM hallucination and repetitive outputs, ensuring a concise and diverse set of features.

The diversity of the justification set is crucial for extracting non-similar justifications for the same Test Case, assigning unique justifications that can be used to classify Test Case complexity. In other words, Test Cases with similar justifications can be considered to have the same or very similar complexity.

3.5 Prompt Construction Classifier Level

The Classifier Level was designed to classify the Test Case complexity. In this Level, all data produced in previous Levels is used as input to our LLM model to realize the fine-tuning process.

The fine-tuning process in this Level is similar to that realized on the Feature Extraction Level. The training on this Level was done without any other trained weights generated on previous Levels, in other words, the fine-tuning made in Classifier Level used the same LLM model but without the weights generated in the Feature Extraction Level. This process was realized due to the fact that all Test Case information must be lent by the dataset and learned by our LLM model in each Level. Therefore, the last Level of our method was created to focus the learning on classifying the Test Case complexity.

4 Experiments

In this section, the experiments were conducted to evaluate the proposed method. It is important to add that our dataset is private and has sensitive data, and thus some of the details might be ommited due to compliance. Due to the same reason, the LLMs used were run locally, to avoid uploading confidential information to cloud-based servers due to compliance.

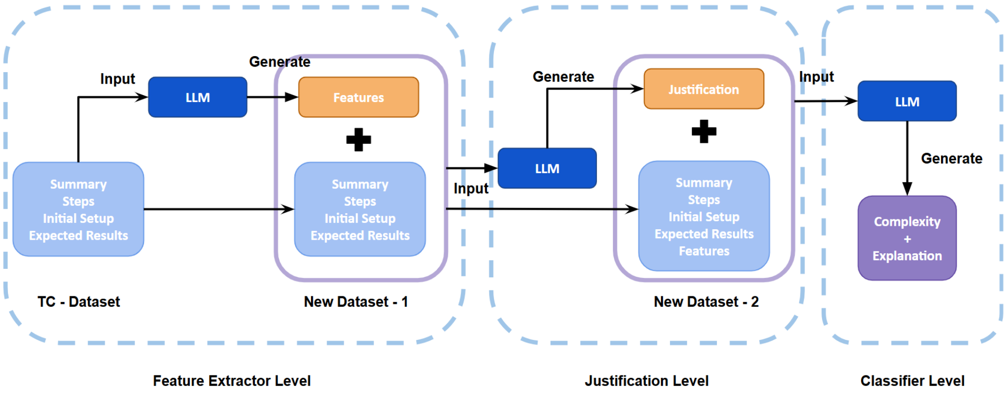


Figure 1: The LLM pipeline of our proposed method, containing 3 levels: Feature Extractor, Justification and Classifier.

Feature Extractor Level and Justification Level	Classifier Level
You are an expert in evaluating the Test Case's complexity which is used to test Android systems. You know the step-by-step process of all test cases and the logical flow of each one. You know all the features to measure the complexity of the Test Case. You are also very familiar with test cases, due to your extensive experience and knowledge in evaluating the difficulty of test cases precisely.	*You are an expert in evaluate the Test Case's complexity which is used to test Android systems. You know the step-by-step process of all test cases and the logical flow of each one. You are also very familiar with test cases, due to your extensive experience and knowledge in evaluating the difficult of a test cases precisely. The difficult/complexity of a Test Case can be: High, Medium or Low.*

Table 2: System Prompt for each Level from our architecture

Feature Extractor Level	Prompt Construction Justification Level	Prompt Construction Classifier Level
Extract the main features of the Test Case. To extract the features of the Test Case, you must strictly follow the following format: Main Features: 1. <Feature 1 be succinct> 2. <Feature 2 be succinct> ... IMPORTANT: The features describes the main components of the Test Case. Pay attention on context. Describe the features of the Test Case. Input: Test Case Data.	*To extract the features of a test case, you must strictly follow the following format: **Features:** 1. <Feature 1 be succinct, maximum of 5 words.> 2. <Feature 2 be succinct, maximum of 5 words.> ... IMPORTANT: Use only the provided format. Keep your answers concise and brief. Pay attention on context. Each Feature must have a maximum of 5 words. Extract the Features of the Test Case. Input: New Dataset - 1*	*Classify the difficult of the respect test case with only one word (Low, Medium or High). To classify the complexity of a test case, you must strictly follow the following format: <The Complexity must be High, Medium or Low> IMPORTANT: The difficult describes how much effort is necessary to realize a test case. The difficult can be considered as Low, Medium and High. Always consider the highest feature to classify the test case. The order of the higher difficult to lower is: High, Medium and Low. Pay attention on context. Classify the difficult of the Test Case. Input: New Dataset - 2*

Table 3: User Prompt for each Level from our architecture

4.1 Dataset

Our data set consists of 354 test cases. Each Test Case has six columns: (1) **Summary**: is a brief Test Case description; (2) Initial Setup: contains all initial configurations to realize the test; (3) **Steps**: describes each step to perform the test; (4) **Expected Results**: the results that are expected after performing the test; (5) **Features**: is a list of features that justify the labeled complexity. This list was created by a team of specialists with the knowledge to classify the Test Case complexity; (6) **Complexity**: describes how much effort a specialist will apply to perform the test.

All experiments were performed with these data; in other words, no additional external data was used to train and/or generate new data samples, except for the data that our own model produced. Both for fine-tuning and generation Levels, our data was split into 283 TCs to train and 71 TCs to test.

4.2 Features Generation

In the Feature Extractor Level, we focused on the fine-tuning of our LLM to learn the features of the Test Case. These features were provided by a team of Android Test Case specialists. As explained before, in this Level we augmented our dataset from 283 TCs to 5287 TCs. The Large Language Model used in the Feature Extractor Level was Llama 3.1 8 billion parameters [2].To perform data augmentation, we used of the same LLM model, Llama 3.1, to generate new examples based on the original training examples.

To perform the fine-tuning, we used the efficient fine-tuning method Quantized Low-Rank Adapters [6], QLoRA, which provides the training of adapters without updating the LLM original weights. Furthermore, the QLoRA reduces the usage of GPU, which enables the fine-tuning of bigger models with the same resources. In this work, we focused on local LLM finetuning due to the confidential nature of the test cases and their complexity labels.

The data used to perform the fine-tuning in this Level is the original dataset, *TC - Dataset*, section 4.1 for more details.

The features generated in the Features Extractor Level will be used to enhance our dataset, *New Dataset - 1* with the features extracted from the LLM. With this new information available, we expect the LLM model to use it as main features in the following Levels, Justification and the Classifier.

4.3 Justification Generation

The Justification Level, we made the LLM generate the justification of the features that it considers more important in the Test Case. In the previous Level, Feature Extractor Level, the LLM learns the Test Case's main features from the view of the specialists. As mentioned earlier, these main features synthesize the Test Case complexity, however, in our experiments, we discovered that the features generated by the LLM, the justifications, from its own point of view, improve further the final results.

To perform Justification Generation, we made use of the Zero-shot strategy [10]. The Large Language Model used to generate the justifications is the same as the previous Level, Llama 3.1 8 billion parameters. An important detail is that the Llama model used in the Justification Level does not use the weights or adapters trained in the Feature Extractor Level, the previous Level.

The LLM receives the original augmented dataset plus the features, the *New Dataset - 1*, to then artificially generate the Test Case justification. This process was realized because we want the LLM to extract the important features on its own. The justifications were extracted from the augmented dataset, with the main features generated at the Feature Extractor Level, resulting in our new enhanced dataset, *New Dataset - 2*. This new dataset will be used by our Complexity Level to classify the Test Case.

4.4 Classification Level

Now, in the last Level, the Classifier Level, we made a fine-tuning of our LLM model to learn the Test Case Complexity. The Large Language Model used in this Level is the same as the previous Levels, Llama 3.1 8 billion parameters and the fine-tuning training was realized through QLoRA. To fulfill this task, we used all the data generated before, *New Dataset - 2*, to perform the fine-tuning. The complexity that our LLM model must learn can be High, Medium or Low, for more details see section 3.1. An important detail in the fine-tuning process in this Level, is that no trained weights or previous adapters were used, in other words, new adapters were trained from scratch in the Classifier Level. Therefore, all the information learned from the previous Levels is shared just by the data.

4.5 Results

To evaluate our method, we utilized traditional classification metrics: precision, recall, f1-score, and accuracy. We also present Confusion Matrices to show the its performance across each class.

An analysis of the confusion matrix (2) reveals two key findings: (1) the LLM is highly effective at differentiating between High and Low complexity classes, and (2) the Medium class (as anticipated) presents the greatest challenge for the model, which often misclassifies High or Low complexity test cases as Medium. Interestingly, when the true complexity was Medium, the LLM achieved strong performance (21 out of 24).

The results in Table 4 provide further details on the LLM's performance in classifying Test Case complexity. While the High and Low classes exhibited high precision (83% and 85% respectively), demonstrating the LLM's capability to differentiate them from the Medium class (62%), the misclassifications of Medium class instances as High (8) and Low (5), reflected in the recalls (69% and 61% respectively), highlight the model's difficulty in distinguishing the Medium class from the others, even with its highest recall (88%). Nevertheless, the similar f1-scores across all classes indicate the LLM's consistency.

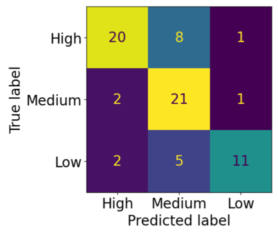
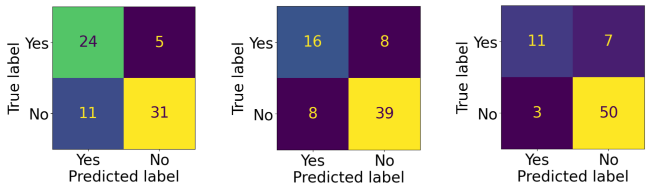


Figure 2: Confusion Matrix for the classification of Test Case Complexity by our proposed method.

Class	Precision	Recall	F1-Score
High	83%	69%	75%
Medium	62%	88%	72%
Low	85%	61%	71%

Table 4: Classification Results of our proposed model in classifying the Test Case complexity.



(a) High Classifier **(b) Medium Classifier** **(c) Low Classifier**

Figure 3: Confusion matrices for the high, medium, and low complexity classifiers.

We believe the Medium complexity was not well-defined by the dataset, potentially leading to numerous misclassifications from other classes. The main features, as labeled by specialists, may not adequately describe or delineate the complexities to effectively differentiate the Medium class from the High and Low classes.

To address the challenges in classifying complex Test Cases, we conducted an experiment: we trained three LLM models using QLoRA for binary classification (yes/no for each complexity class). Figure 3 illustrates the results, revealing the LLM's difficulty in performing a single-task complexity classification. As observed, employing three classifiers led to higher precision for High and Low classes, but lower precision for the Medium class. This approach could be an interesting alternative for use cases where identifying edge cases is paramount.

Finally, Table 5 demonstrates how the different levels of our proposed method impact the final classification results. It's evident that

Experiments	Precision	Recall	F1-Score	Accuracy	Dataset	Fine-Tuning
Classifier Only	13%	30%	18%	37%	TC - Dataset	No
Classifier Only	**79%**	53%	48%	56%	TC - Dataset	Yes
Feature Extractor + Classifier	24%	32%	22%	38%	New Dataset - 1	No
Feature Extractor + Classifier	70%	60%	61%	63%	New Dataset - 1	Yes
Feature Extractor + Justification + Classifier	29%	37%	29%	44%	New Dataset - 2	No
Feature Extractor + Justification + Classifier	77%	**73%**	**73%**	**73%**	New Dataset - 2	Yes

Table 5: Ablation of the Proposed Method, regarding both LLMs with and without fine-tuning.

System Prompt	User Prompt	Explanation
You are an expert in evaluating Test Case's complexity to test Android systems. You know the step-by-step process of all test cases and the logical flow of each one. You are also very familiar with test cases, due to your extensive experience and knowledge in evaluating the difficult of a test cases precisely. The difficult/complexity of a Test Case can be: High, Medium or Low.	*Classify the difficult of the respect test case with only one word (Low, Medium or High). To classify the complexity of a test case, you must strictly follow the following format:* *<The Complexity must be High, Medium or Low>* *<Explanation: explain your answer>* *IMPORTANT:* *The difficult describes how much effort is necessary to realize a test case.* *The difficult can be considered as Low, Medium and High.* *Always consider the highest feature to classify the test case.* *The order of the higher difficult to lower is: High, Medium and Low.* *Use the field Main Features as main information to classify the complexity.* *After this field you consider Features, Summary, Steps, Initial Setup, Expected Results in this order.* *Pay attention on context.* *Classify the difficult of the Test Case.* *Explain the reason you classify the complexity of the Test Case.* Input: New Dataset - 2	*The test case involves basic Enable/Disable functionality of SIM Card, user interface verification, which doesn't require any complex actions or conditions. It's a simple verification of SIM Card status change.* *Complexity: Low*

Table 6: Explainability Example

fine-tuning the LLMs with QLoRA significantly affects the classification outcomes, with fine-tuned versions consistently outperforming their original counterparts. Furthermore, while incorporating the feature extractor level enhanced recall, it concurrently reduced precision. Lastly, the inclusion of the justification level resulted in a notable increase across all metrics, strongly indicating that the more processed justification indeed led to improved comprehension at the classification level.

4.6 Explainability

Our method generates two levels of explainability. In the Justification Level, the explainability is presented with the features generated by the LLM. In section 3.4, we describe the prompt and how the LLM will generate this information. This way, the LLM itself evaluates the importance of each feature and basically summarizes and explains the Test Case regarding its complexity. Evaluating these features, explanations, generated by the LLM, we can see the improvement of the model performance shown in Table 5.

In the Classifier Level, our method generates the explanation and the Test Case complexity at the same level. In Table 6, we can see an example of how the explanation works. The LLM model explains the reason for the complexity in a simple way, before the complexity. Therefore, both the LLM can use this explanation in its reasoning and human, more importantly, the explanation gives insights to QA analysts when analyzing the model results.

5 Conclusion

In this work, we proposed a method that uses data augmentation and explainability to improve Test Case Complexity classification. The results show that our method improves not only the precision of the model, but also how the explainability generated in Justification Level improves the answer of the model. Furthermore, the justification assists in the understanding of the misclassifications by the specialist and by the own LLM.

However, our results showed that, while differentiating High complexity from Low yielded good results, the Medium class is still the hardest to classify. In future work, we intend to perform a more in-depth analysis regarding how to better differentiate Medium complexity Test Cases, including interviews with QA analysts to better understand these differences.

ARTIFACT AVAILABILITY

In compliance with confidentiality policies and signed data protection agreements, the datasets utilized in this study cannot be publicly disclosed. These datasets contain sensitive information and internal processes, including mobile device validation flows, file naming conventions, and proprietary technical commands. Access to these datasets has been strictly limited to internal research and solution development purposes, in accordance with the terms set forth in confidentiality agreements (NDAs).

ACKNOWLEDGMENTS

This work was supported by the following Brazilian agencies: the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES-PROEX)- Finance Code 001, the National Council for Scientific and Technological Development (CNPq), and Amazonas State Research Support Foundation- FAPEAM- through the POSGRAD project 2024/2025, and by Motorola Mobility Comércio de Produtos Eletrônicos Ltda, under the terms of Federal Law No. 8.387/1991, through agreement No. 02/2021 with ICOMP/UFAM.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] AI@Meta. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[3] Antonia Bertolino. 2007. Software Testing Research: Achievements, Challenges, Dreams. In *Future of Software Engineering (FOSE '07)*. 85–103. doi:10.1109/FOSE.2007.25

[4] Nicolas Antonio Cloutier and Nathalie Japkowicz. 2023. Fine-tuned generative LLM oversampling can improve performance over traditional techniques on multiclass imbalanced text classification. In *2023 IEEE International conference on big data (BigData)*. IEEE, 5181–5186.

[5] Naihao Deng, Yikai Liu, Mingye Chen, Winston Wu, Siyang Liu, Yulong Chen, Yue Zhang, and Rada Mihalcea. 2023. EASE: An Easily-Customized Annotation System Powered by Efficiency Enhancement Mechanisms. arXiv:2305.14169 [cs.HC] https://arxiv.org/abs/2305.14169

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG] https://arxiv.org/abs/2305.14314

[7] Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, and William Cohen. 2019. Handling Divergent Reference Texts when Evaluating Table-to-Text Generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4884–4895. doi:10.18653/v1/P19-1483

[8] Finale Doshi-Velez and Been Kim. 2017. Towards A Rigorous Science of Interpretable Machine Learning. arXiv:1702.08608 [stat.ML] https://arxiv.org/abs/1702.08608

[9] Evgenia Gkintoni, Hera Antonopoulou, Andrew Sortwell, and Constantinos Halkiopoulos. 2025. Challenging Cognitive Load Theory: The Role of Educational Neuroscience and Artificial Intelligence in Redefining Learning Efficacy. *Brain Sciences* 15, 2 (2025), 203.

[10] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916 [cs.CL] https://arxiv.org/abs/2205.11916

[11] Dragan Milicev. 2007. On the Semantics of Associations and Association Ends in UML. *IEEE Transactions on Software Engineering* 33, 4 (2007), 238–251. doi:10.1109/TSE.2007.37

[12] Anders Giovanni Møller, Jacob Aarup Dalsgaard, Arianna Pera, and Luca Maria Aiello. 2023. The parrot dilemma: Human-labeled vs. LLM-augmented data in classification tasks. *arXiv preprint arXiv:2304.13861* (2023).

[13] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. doi:10.1145/2939672.2939778

[14] Gemma Team. 2025. Gemma 3 Technical Report. arXiv:2503.19786 [cs.CL] https://arxiv.org/abs/2503.19786

[15] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[16] Sowmya Vajjala and Shwetali Shimangaud. 2025. Text Classification in the LLM Era-Where do we stand? *arXiv preprint arXiv:2502.11830* (2025).

[17] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. 2024. Smart Expert System: Large Language Models as Text Classifiers. *arXiv e-prints* (2024), arXiv–2405.

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR* abs/2201.11903 (2022). arXiv:2201.11903 https://arxiv.org/abs/2201.11903

[19] Xuansheng Wu, Wenhao Yu, Xiaoming Zhai, and Ninghao Liu. 2025. Self-regularization with latent space explanations for controllable llm-based classification. *arXiv preprint arXiv:2502.14133* (2025).

[20] Yuhang Wu, Yingfei Wang, Chu Wang, and Zeyu Zheng. 2024. Large Language Model Enhanced Machine Learning Estimators for Classification. *arXiv preprint arXiv:2405.05445* (2024).

[21] Yazhou Zhang, Mengyao Wang, Chenyu Ren, Qiuchi Li, Prayag Tiwari, Benyou Wang, and Jing Qin. 2024. Pushing The Limit of LLM Capacity for Text Classification. arXiv:2402.07470 [cs.CL] https://arxiv.org/abs/2402.07470

[22] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2023. Explainability for Large Language Models: A Survey. arXiv:2309.01029 [cs.CL] https://arxiv.org/abs/2309.01029