# An Approach for Tackling Technical Debt in Open Source Projects with Transformer Models

Program: Graduate
David Gao (`gaod1, david.gao@vanderbilt.edu`)

Dec 7, 2023

**Overview.** We propose usage of transformer models to uncover insights in regard to technical debt in open source software development. We focus on uncovering underlying issues, identifying patterns, and providing effective feedback to developers in open source projects with our solution.

**Background and Motivation.** Technical Debt is an everlasting issue for software development in almost every environment. In many software endeavors within a project, workarounds or shortcuts might be taken, resulting in extra work that needs to be done in the future to fix the issues that come up. This can hold back development teams in the development and maintenance of any codebase. Specifically, in open source projects, contributions of code can come from a wide range of developers from all over the world, and this might make the problem worse. Recent developments in deep learning with transformer models have been very able to capture large amounts of complex data, and we propose leveraging this work to approach the technical debt issue.

**Intellectual Merit.** The proposed research project holds significant intellectual merit, as it seeks to harness the power of transformer models in addressing a critical and pervasive issue in the realm of open source software development: technical debt. By leveraging fine-tuned transformer models, our research endeavors provide a novel and highly effective approach to understanding and mitigating technical debt in open source software. The intellectual merit of our project can be summarized in the following points:

- **Complex Pattern Identification:** Transformer models will empower our solution to identify various forms of technical debt within code. This capacity for nuanced pattern identification is a substantial intellectual contribution to the field.
- **Effective Feedback Loop:** My research tries to fill an outstanding need in the software development lifecycle by providing developers with feedback that they can employ to improve their software. This will facilitate more effective code contribution and expedite the remediation of technical debt across open source projects, leading to higher quality code.
- **Empirical Insights:** By applying the transformer model to various open source projects, this paper will reveal insights into the prevalence and impact of technical debt across different repositories which can serve as a guide for best practices across the open source community.

**Broader Impact.** The broader impact of this study is immediate and can grow as it is integrated into the software development practices everywhere. By providing this insight into technical debt, my research can improve the sustainability of open source projects, ensuring that the benefits of open source software continues to reach users everywhere and improve lives. As we understand technical debt more deeply, software practices can be adapted to minimize its impact at all levels of development, resulting in better code. This can be applied to software development life cycles beyond open source projects and can revolutionize the next generation of software engineering.

## 1 Introduction

Managing technical debt has been a big focus for organizations since the concept was termed by Ward Cunningham in 1992 [4, 5]. Over the years, as more software has been built, the amount of technical debt has increased as well, and is projected to continue growing [25]. Technical debt can build up over time, making future changes harder and clogging up projects. The main idea is that the best long term solution to a problem is put on hold in exchange for a faster short term solution [3, 14]. If it is ignored or not managed well, the debt can add up and limit the development of a project, and can even lead to unusable software in the future [3].

In open source software development, managing technical debt is even more important. Open source projects often involve a diverse group of contributors, and they all might come from different backgrounds [10]. Without the structure of a centralized organization, it is even easier for the debt to accumulate [10, 30]. Furthermore, since open source software has grown in popularity, any issues that come from this technical debt can have a big impact [8]. Therefore, open source software projects must try to limit the negative impacts of technical debt on their projects to make high quality software [1].

While there have been many studies on strategies for mitigating technical debt with various approaches, most of them focus on self-admitted technical debt [10, 20, 30] and issue trackers [22, 26]. Thus, we still don't have many modern solutions that leverage cutting edge technology to tackle this issue beyond the reach of human error and bias. I argue that a successful software solution must align with these standards:

- **Accuracy.** It is essential that technical debt is accurately identified and classified for it to have any benefit to developers.
- **Interpretability.** Tools must provide an interface that decreases the time developers need to take to understand technical debt issues, so that they can be addressed.

My approach comes from a few key insights. At the forefront is the utilization of transformer models [28]. Transformer architectures, since their introduction, have revolutionized the field of machine learning, especially in the domain of natural language processing. They offer an exceptional ability to capture and understand context and sequences due to their self-attention mechanism [28]. This allows them to grasp small details in complex data sets and generate meaningful results.

The true power of transformers is their ability to be applied to many downstream jobs efficiently. They can be fine-tuned for many tasks, ranging from sentiment analysis to code prediction [27]. This adaptability amplifies their value, especially when tackling problems that are large scale and complex.

These capabilities are ideally suited for the issues of technical debt analysis, which have previously been restricted by the subjectiveness of the problem. However, this somewhat abstract task is extremely compatible with the flexible nature of transformer models, since they have demonstrated considerable proficiency in understanding complex sequences of data and generating coherent responses.

In leveraging these insights, my approach aims to harness the full potential of transformer models and to utilize it in a novel manner.

**Our first proposed component is to identify patterns of technical debt from commit messages and code augmentation using transformer based models.** We believe that leveraging transformer-based models will offer a cutting-edge approach to this. By parsing and interpreting commit messages from developers, we aim to uncover subtle indicators of technical debt. Additionally, by observing code changes, we can further pinpoint areas where technical debt may begin to build up, unnoticed. The transformer models' capability to understand context and sequence makes them an ideal candidate for this task, ensuring a thorough and accurate analysis.

**Our second proposed component is to investigate the most effective ways of integrating technical debt feedback into the software development process.** It's imperative to not only identify technical debt but also to find ways to make this information actionable for development teams. We aim to implement a user-friendly feedback mechanism which can give developers insight into the situation at hand. The goal is to ensure that developers receive relevant information about technical debt in a manner that can easily be assimilated into their workflows.

**Our third proposed component investigates the state of technical debt in active open source repositories.** By analyzing a wide range of repositories, we hope to get a broad understanding of how technical debt manifests in different types of open source projects. Open source repositories can provide a lot of understanding, given their transparent nature and diverse contributor base. Insights gleaned from this study could offer invaluable benchmarks and best practices for both open source and industry software projects.

The overall thesis statement of the proposed research is:

> *Implementing transformer models to analyze open source software can significantly enhance the identification and understanding of technical debt patterns, leading to more effective feedback to developers and resulting in improved software quality.*

The expected contributions of the proposed solution are the following:

1. A fine-tuned transformer model for classification of technical debt given the lines of code added from a code differential.
2. A fine-tuned transformer model for sequence to sequence generation of descriptive statements given lines of code previously identified as technical debt.
3. A comparison of technical debt analyses across a few open source repositories.

The remainder of this proposal is as follows. In Section 2, we discuss background material and related research efforts. Next, we describe our proposed research and associated technical approaches in Section 3, and in Section 4, we detail our evaluation plan. Finally, in Section 5 we present results of preliminary experiments then finish with a discussion of concluding thoughts in Section 6.

## 2 Background and Related Work

### 2.1 Transformers

Our first research component makes use of a fine-tuned transformer model. Since their inception, transformer architectures have revolutionized the domain of natural language processing (NLP) and many other tasks. The transformer's unique structure, which leverages self-attention mechanisms, allows it to capture context and sequence information very well [28]. This research will focus on the use of encoder transformer models similar to BERT [6]. Specifically, there has been work done to train a model which understands code very well with the same architecture [7]. Models derived from BERT have pre-trained representations that are very good, but we will leverage the ability to fine-tune this model so it can perform well on downstream tasks such as classification among many others [24]. These models have demonstrated groundbreaking performance in complex NLP tasks, and they have outperformed precursors by large margins. With these vast improvements, we hypothesize that transformer models can be applied in the area of code analysis, and can outperform even the most popular modern approaches.

### 2.2 Open Source Software Development

A big focus in all the research components is the way in which we seek to uncover insights into the technical debt status of open source projects. Many open source projects are contained within some sort of code repository, which now utilizes a distributed version control system which allows developers all over the world to contribute to a project while it is overseen by a core group of developers [21]. A typical "commit" to a repository will typically contain a list of changes that were made in the codebase and a corresponding message. There also are issue trackers in some projects, which are platforms where developers and users can report and track all sorts of issues that are relevant to the project.

### 2.3 Technical Debt

Technical debt was a term coined by Ward Cunningham in 1992 [5] and describes the cost of short term solutions which may shortcut best practices and come back to require extensive refactoring in the long term [4]. There have been many studies that propose various ways to measure and manage technical debt.

A large number of works focus on self admitted technical debt which is when developers submit code that falls under the technical debt definition on purpose, usually with the intention of coming back to fix it [19]. This is typically declared in commit messages, comments, issue sections, and pull requests [13]. Using this information, studies have sought to identify [10, 13, 20], quantify [18], prioritize [2], and manage [9, 17] technical debt.

Some of these approaches have utilized deep learning and artificial intelligence and found success [17, 20]. Even more recently, transformers have started to be used to tackle these problems by classifying technical debt from natural language written by developers [22].

## 3 Proposed Research

Our goal is to provide a software solution to identify technical debt and give meaningful feedback to developers. We propose the following three research components to achieve this task:

1. A fine-tuned transformer model for classification of technical debt given the lines of code added from a code differential.
2. A fine-tuned transformer model for sequence to sequence generation of descriptive statements given lines of code previously identified as technical debt.
3. A comparison of technical debt analyses across a few open source repositories.

The remainder of this section overviews each component and its corresponding approach.

### 3.1 Fine-Tuned CodeBERTa Classification Model

The goal of this research component is to train a model which can classify commits containing technical debt from the lines added in a git commit. To do this, we will leverage the power of transformer models and apply them to the specific tasks that we defined earlier. We propose taking the CodeBERTa model [11], a variant of the popular RoBERTa model specifically pre-trained on a diverse range of code, and fine-tuning it using the Technical Debt Dataset [12]. This dataset, meticulously annotated, provides many examples where technical debt has been identified along with their corresponding commits and code changes.

Through this fine tuning process, we aim to have a model which has a deep understanding of the linguistic and structural patterns that characterize technical debt. Using this, the model will be trained for a binary classification task, classifying a code sequence as technical debt or not technical debt.

### 3.2 Fine-Tuned CodeT5 Sequence-to-Sequence Model

With a subsection of the Technical Debt Dataset, we will train a second model to describe the technical debt occurring within a given commit. We propose fine-tuning the CodeT5 model [29], a version of the T5 encoder-decoder transformer model trained on code. We fine-tune this model on extracted SonarQube [2] messages from the Technical Debt Dataset corresponding to commits that have already been previously identified as technical debt.

Using this dataset, we train a sequence to sequence generation model, creating informational feedback messages from code input from commits. This provides developers with interpretable messages which accelerates the technical debt remediation process, and aligns identified technical debt with a natural language description.

### 3.3 Empirical Analysis of Open Source Repositories

With the model and our tooling, we will analyze a few open source repositories and try to gather insights into their technical debt situation. Our objective is to draw correlations between the types of technical debt present and their impact on the project's health. This exploratory phase will involve applying the first two research components to the commit history of a project and diving into the data. With this, we can identify similarities and differences across various projects of different sizes, use cases, and popularity. This will provide valuable insights for the entire open source community and beyond.

## 4 Proposed Experiments and Metrics

In this section, we summarize the experiments and metrics that we will use to evaluate each of our research efforts.

### 4.1 Fine-Tuned CodeBERTa Classification Model

We propose the utilization of components from the Technical Debt Dataset to fine-tune a CodeBERTa model to identify technical debt in a binary classification task. Our approach must be able to carry out this classification class with high accuracy and a high F1 metric to be effective. Thus, we propose the following experiment and evaluation strategy.

The Technical Debt Dataset provides many measures of technical debt, one of which is their use of the SZZ algorithm [23] within Git repositories to create a SQL table containing the commit hash where an issue originated. Using the SZZ_FAULT_INDUCING_COMMITS table and the GIT_COMMITS_CHANGES table, which contains all information of a commit hash, including the code differential, we can query the Technical Debt Dataset to create a dataset which fits our classification task. From this query, we obtain the code differential and create a binary column which is marked with 1 if the commit hash exists within the fault inducing table and 0 if it does not. We further process this data by parsing the code differential to only include the added lines of code, removing all other irrelevant symbols from the differential. This approach was adopted based on the assumption that technical debt originates solely from additive procedures in programming.

This dataset was then separated into training, validation, and test splits and was pushed to Huggingface at https://huggingface.co/datasets/davidgaofc/techdebt. The training set consists of approximately 207,000 entries while both the validation and test sets consist of approximately 69,200 entries each.

Using this dataset, we can fine-tune a CodeBERTa model for classification since the input is pure code, and we have formatted a column for the binary classification task. To evaluate this model, we utilize all of the standard metrics used in classification tasks: accuracy, F1, precision, and recall. While all metrics are important for measuring the effectiveness of our model in practice, we especially focus on the recall, since our primary task is catching all positive cases of technical debt in the open source repositories.

## 4.2 Fine-Tuned CodeT5 Sequence-to-Sequence Model

For our second component, we propose utilizing other components from the Technical Debt Dataset to tine-tune a CodeT5 model for sequence to sequence generation so that we can generate descriptions of commits that have already been identified as technical debt. Our approach must create accurate descriptions of the issues in the code, so we evaluate the generated descriptions using the ROUGE metrics proposed by Lin [15]. These metrics measure the overlap of the machine generated response to the corresponding ”correct” reference response.

For this task, we utilize the SonarQube results compiled within the Technical Debt Dataset. SonarQube is an open source software which utilizes static analysis to detect bugs, code smells, and other software issues within code using a rule-based system. Using the SONAR_ISSUES table, we obtain the SonarQube message which describes the issue to a given commit hash and join it with the GIT_COMMITS_CHANGES table. From this query, we create a dataset which maps a code differential to the SonarQube message that corresponds to it. Again, we clean the code differential such that only the added lines of code persist. It is important to note that all entries in this dataset have already been identified as technical debt in some fashion.

This dataset was split into train and test subsets, and is available at https://huggingface.co/datasets/davidgaofc/techdebt_label. The training split consists of around 8,790 entries while the test split contains around 2,200 entries.

Using this, we can fine-tune a CodeT5 model for sequence to sequence generation since we have a mapping of code input to natural language output. To evaluate this model, we utilize the ROUGE scores, which compare our machine generated text to the reference text provided by our test dataset. Specifically, we look at the ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-L SUM scores which are different measures of token overlap. ROUGE-1 measures single token overlap,

Table 1: Classification Metrics

| Accuracy | F1 | Precision | Recall |
|---|---|---|---|
| 95.410 | 94.929 | 94.941 | 95.410 |

ROUGE-2 measures the overlap of pairs of tokens, the two ROUGE-L scores measure the longest common sub-sequence between the generated response and the reference text.

### 4.3 Empirical Analysis of Open Source Repositories

For the third research component, we will select a diverse set of open source repositories based on size and purpose. It is important to note that the Technical Debt Dataset that we trained our models on consists exclusively of Java projects, so the analysis is restricted to this category as well.

After selecting an adequate sample of open source Java repositories, we can utilize the Public Git Archive [16] to extract all commits from our sampled projects. From these commits, we must utilize the same data pre-processing steps to extract added lines of code from the code differentials. Then, we can classify all commits within the project with our fine-tuned CodeBERTa classification model, and obtain a measure of the extent of technical debt present in the project.

This is best measured by a percentage of commits containing technical debt as well as a proximity analysis to see the files where technical debt occurs more often. We can also perform analyses on the users within each project that commit the most code containing technical debt.

Then, for the positive classifications of technical debt, we feed the cleaned code differentials into our fine-tuned CodeT5 model which creates descriptions for the data. This can be processed and clustered to reveal further insight into the specific issues afflicting a project. By utilizing other NLP models, we can generalize these commit-level descriptions into a high-level perspective to understand the state of technical debt within an open source repository.

After collecting this data across our diverse set of repositories, we can conduct an in-depth analysis of the extent and form of technical debt that occurs in all repositories in relation to the size and purpose of the repository. These correlations form the basis of our empirical analysis for our final research component.

## 5 Preliminary Results

In this section, we present preliminary results from ongoing research.

### 5.1 Fine-Tuned CodeBERTa Classification Model

We have completed the fine-tuning process for our technical debt classification model, and its performance on the test dataset is displayed in Table 1. This model was trained using the CodeBERTa-small-v1 model as a starting point and trained for a single epoch with a batch size of 30 on the training dataset specified previously. This model is available at `https://huggingface.co/davidgaofc/TechDebtClassifier`.

As illustrated in the table, the transformer model can identify technical debt from the cleaned code differential with extremely high performance across all metrics. The metric we defined as

Table 2: Sequence Generation Metrics

| ROUGE-1 | ROUGE-2 | ROUGE-L | ROUGE-L SUM |
|---|---|---|---|
| 46.710 | 39.842 | 46.308 | 46.274 |

most important in this context, recall, registers at 95.410, indicating that we can catch almost all instances of technical debt purely based on the lines of code added from a commit. This suggests that our initial hypothesis that transformers capture the many nuances of textual data is correct, and these models represent an effective approach to the issue.

### 5.2 Fine-Tuned CodeT5 Sequence-to-Sequence Model

We have completed the fine-tuning process for our second research component as well, and its performance on the designated test dataset is displayed in Table 2. This model was trained using the CodeT5-small model as a starting point and also trained for a single epoch with a batch size of 1. The model is available at `https://huggingface.co/davidgaofc/TechDebtLabeler`.

The ROUGE scores displayed in the table indicate a reasonable similarity between the generated descriptions and the reference SonarQube messages. However, we do acknowledge that these scores are averaged across the entire test dataset, so we currently lack a metric which provides a thorough measurement of the accuracy of the descriptions generated by our model.

While the results do not indicate a superiority in the ability of the transformer to generate descriptions of technical debt, we acknowledge that the ROUGE score is limited by its simple token overlap calculation, and human evaluation is needed to truly determine the accuracy of the labels generated by our sequence to sequence model.

## 6 Conclusion

Technical debt is a challenging issue that persists within all software development processes, and the problem manifests itself in a more pronounced manner in open source software projects. Managing this is an open problem, and recent developments in transformer based models provide an avenue worth exploring because of their emergent abilities to capture sequences of textual data, whether it is natural language or code. In this proposal, we present a two-model system to improve our understanding of technical debt within open source repositories using the classification and generation abilities of transformer models. We hypothesize that these models can accurately classify instances of technical debt and generate insightful descriptions of the technical debt present within Git commits.

We outline three research verticals in this proposal as follows. Our first research vertical consists of the creation of a fine-tuned CodeBERTa model to classify technical debt. Our second research vertical comprises the training of a fine-tuned CodeT5 model for generation of descriptions for technical debt. Our final research vertical is composed of an empirical analysis of open source projects using the two models proposed previously, uncovering insights into the status of technical debt across various categories of open source repositories. This study will demonstrate a novel approach to tackling technical debt using recent technological advances in transformer models and will improve our ability to understand and remedy technical debt across open source projects.

## References

[1] Aberdour, M. Achieving quality in open-source software. *IEEE software 24*, 1 (2007), 58–64.

[2] Alfayez, R., Winn, R., Alwehaibi, W., Venson, E., and Boehm, B. How sonarqube-identified technical debt is prioritized: An exploratory case study. *Information and Software Technology 156* (2023), 107147.

[3] Behutiye, W. N., Rodríguez, P., Oivo, M., and Tosun, A. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. *Information and Software Technology 82* (2017), 139–158.

[4] Berenguer, C., Borges, A., Freire, S., Rios, N., Ramač, R., Taušan, N., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., et al. Investigating the relationship between technical debt management and software development issues. *Journal of Software Engineering Research and Development* (2023), 3–1.

[5] Cunningham, W. The wycash portfolio management system. *ACM Sigplan Oops Messenger 4*, 2 (1992), 29–30.

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] Fuggetta, A. Open source software—-an evaluation. *Journal of Systems and software 66*, 1 (2003), 77–90.

[9] Haki, K., Rieder, A., Buchmann, L., and W. Schneider, A. Digital nudging for technical debt management at credit suisse. *European Journal of Information Systems 32*, 1 (2023), 64–80.

[10] Huang, Q., Shihab, E., Xia, X., Lo, D., and Li, S. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering 23* (2018), 418–451.

[11] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[12] Lenarduzzi, V., Saarimäki, N., and Taibi, D. The technical debt dataset. In *Proceedings of the fifteenth international conference on predictive models and data analytics in software engineering* (2019), pp. 2–11.

[13] Li, Y., Soliman, M., and Avgeriou, P. Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering 28*, 3 (2023), 1–38.

[14] Lim, E., Taksande, N., and Seaman, C. A balancing act: What software practitioners have to say about technical debt. *IEEE software 29*, 6 (2012), 22–27.

[15] Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out* (2004), pp. 74–81.

[16] Markovtsev, V., and Long, W. Public git archive: a big code dataset for all. In *Proceedings of the 15th International Conference on Mining Software Repositories* (2018), pp. 34–37.

[17] Pandi, S. B., Binta, S. A., and Kaushal, S. Artificial intelligence for technical debt management in software development. *arXiv preprint arXiv:2306.10194* (2023).

[18] Perera, J., Tempero, E., Tu, Y.-C., and Blincoe, K. Quantifying technical debt: A systematic mapping study and a conceptual model. *arXiv preprint arXiv:2303.06535* (2023).

[19] Potdar, A., and Shihab, E. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution* (2014), IEEE, pp. 91–100.

[20] Qu, Y., Bao, T., Yuan, M., and Li, L. Deep learning-based self-admitted technical debt detection empirical research. *Journal of Internet Technology 24*, 4 (2023), 975–987.

[21] Rodríguez-Bustos, C., and Aponte, J. How distributed version control systems impact open source software projects. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)* (2012), IEEE, pp. 36–39.

[22] Skryseth, D., Shivashankar, K., Pilán, I., and Martini, A. Technical debt classification in issue trackers using natural language processing based on transformers. In *2023 ACM/IEEE International Conference on Technical Debt (TechDebt)* (2023), IEEE, pp. 92–101.

[23] Śliwerski, J., Zimmermann, T., and Zeller, A. When do changes induce fixes? *ACM sigsoft software engineering notes 30*, 4 (2005), 1–5.

[24] Sun, C., Qiu, X., Xu, Y., and Huang, X. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18* (2019), Springer, pp. 194–206.

[25] Szykarski, A. Ted theodorpoulos on managing technical debt successfully, 2012.

[26] Tan, J., Feitosa, D., and Avgeriou, P. The lifecycle of technical debt that manifests in both source code and issue trackers. *Information and Software Technology 159* (2023), 107216.

[27] Tay, Y., Dehghani, M., Rao, J., Fedus, W., Abnar, S., Chung, H. W., Narang, S., Yogatama, D., Vaswani, A., and Metzler, D. Scale efficiently: Insights from pre-training and fine-tuning transformers. *arXiv preprint arXiv:2109.10686* (2021).

[28] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[29] Wang, Y., Wang, W., Joty, S., and Hoi, S. C. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[30] Zampetti, F., Fucci, G., Serebrenik, A., and Di Penta, M. Self-admitted technical debt practices: a comparison between industry and open-source. *Empirical Software Engineering 26* (2021), 1–32.