Self-Admitted Technical Debt Analysis in Competition Notebooks on Kaggle

Ana Flávia de Carvalho Santos[1], **Júlia Moreira Nascimento**[1], **Juliana Parreiras Guimarães da Cunha**[1], **Lorrayne Marayze Silva de Oliveira**[1], **Pedro Henrique Marques de Oliveira**[1], **Pedro Henrique Pires Rodrigues**[1]

[1]Department of Software Engineering
Pontifical Catholic University of Minas Gerais (PUC Minas)
Av. Dom José Gaspar, 500 – Coração Eucarístico – Belo Horizonte – MG 30535-901

Abstract. *NAO PODE MAIS DE 10 LINHAS.*
VER COMO COLOCA KEYWORDS.

1. Introduction

Technical debt (TD) is a metaphor in software engineering that describes the long-term effects that originate from development decisions intended to speed delivery (for example, shortcuts in design, incomplete documentation, or superficially implemented features). These decisions can increase future maintenance effort, reduce overall software quality, and introduce fragility into evolving systems. Therefore, identifying and measuring TD is crucial for effective management and for reducing the negative impacts on software evolution and maintenance. Systematic reviews and empirical studies have documented causes, identification strategies, and metrics for TD. However, even with many studies addressing how to identify and measure technical debt, several domain-specific characteristics remain largely unexplored. [Melo et al. 2022]

A prominent and practical subtype of TD is Self-Admitted Technical Debt (SATD). In these instances, developers explicitly acknowledge imperfect or temporary solutions in the codebase via comments or annotations (e.g., TODO, FIXME, HACK). SATD is attractive for empirical study because these developer-written signals are observable and often correlate with later maintenance effort or defect proneness. Early exploratory work established methods for mining SATD from source comments and demonstrated its prevalence across projects. Large-scale studies and recent work focused on ML code have refined taxonomies and shown that SATD patterns in ML systems can differ from those in traditional software (both in type and in prevalence). [Potdar and Shihab 2014]

Kaggle is an online platform for data science and machine-learning practice that hosts public datasets, code (Notebooks), and competitions. Kaggle Notebooks are interactive, cloud-hosted Jupyter-style documents that combine code cells, outputs, and narrative (Markdown), enabling reproducible experiments and easy sharing of data-science workflows. Because notebooks integrate code, data exploration, model training, and result presentation in one place, they are widely used by practitioners to prototype, document, and publish ML experiments, making them a rich, real-world source for mining engineering practices (including SATD) in data science work. [kag 2025]

Kaggle competitions present a problem statement and a dataset. Participants develop models (often inside Notebooks), generate predictions on a test set, and submit those predictions to Kaggle. Submissions are scored automatically according to a competition metric (e.g., RMSE, AUC, F1) and compiled on a live leaderboard that ranks competitors. Additionally, authors can publish their Notebooks publicly, and other users can upvote them or fork/iterate over them. Competitions are often time-boxed and driven by rewards such as medals, prizes, or prestige, which tends to encourage rapid iteration, multiple experimental runs, and practical engineering choices that may produce SATD artifacts in the shared notebooks. [Kaggle 2025]

Furthermore, this study aims to perform a large-scale empirical analysis of SATD specifically within Kaggle competition Notebooks. We aim to (i) identify and categorize the prevalent types of SATD found in competition notebooks; (ii) measure how SATD density and categories correlate with competition success and visibility (medals, leaderboard position, and notebook upvotes); and (iii) compare SATD distribution across competition domains such as Tabular, NLP, and Computer Vision. These objectives follow directly from gaps identified in prior SATD research (both large-scale taxonomy work and ML-focused analyses) [Bhatia et al. 2023] and from the opportunity that Kaggle Notebooks provide as an observable, reproducible artifact for ML engineering studies.

2. Related Work

Research on technical debt (TD) has advanced from conceptual definitions to empirical analyses of how debt affects software evolution and maintenance. [Melo et al. 2022] conducted a systematic review on Requirements Technical Debt, identifying and measuring its manifestations across the development process.

A foundational study by [Potdar and Shihab 2014] established a systematic approach for detecting Self-Admitted Technical Debt (SATD) in source code by mining developer comments that acknowledge technical compromises or deficiencies. Their definition, in which SATD corresponds to explicit, developer-written acknowledgments of incomplete or temporary solutions, is adopted in this work to identify SATD instances within Kaggle competition notebooks.

Subsequent studies extended this foundation through automation and large-scale analysis. [Kirda 2017] introduced NLP-based classification using word embeddings, while [Di Salle et al. 2022] proposed PILOT, a deep-learning approach for SATD detection combining text preprocessing and neural networks. These advances enabled finer-grained analysis of developer comments and improved accuracy in SATD identification.

As machine learning (ML) became integral to modern software systems, researchers began to examine how technical debt manifests in ML contexts. The seminal paper *Hidden Technical Debt in Machine Learning Systems* by [Sculley et al. 2015] described unique forms of data, configuration, and modeling debt that arise from ML pipelines and experiments.

[Bhatia et al. 2023] conducted *23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software*, providing a comprehensive taxonomy of SATD tailored for ML systems. The authors analyzed over 2,000 comments across ML repositories and identified nine main categories of SATD, encompassing aspects such as data quality, model design, reproducibility, and experimentation. This categorization revealed that SATD in ML systems differs both in nature and frequency compared to traditional software. Because of its clear structure and ML-specific insights, this taxonomy serves as the foundation for our classification in this study.

Complementary works such as *An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems* [Tang et al. 2021] and *Characterizing and Mitigating Self-Admitted Technical Debt in Build Systems* [Xiao et al. 2021] investigate the relationship between SATD, refactoring practices, and build maintenance, reinforcing the importance of SATD management in ML development.

Despite the growing literature, no previous research has examined SATD in the context of competitive data-science notebooks, such as those on Kaggle. These environments are inherently time-constrained and reward-driven, encouraging quick iterations and experimentation that may introduce or amplify SATD. This study, therefore, extends prior research by analyzing SATD in Kaggle competition notebooks, applying the nine-category taxonomy from [Bhatia et al. 2023] to investigate patterns, frequency, and correlations with competition outcomes such as medals, leaderboard ranking, and community visibility.

3. Research Method

This section presents the methodological design adopted in this study, which combines large-scale data collection, automated analysis, and manual validation to identify and characterize Self-Admitted Technical Debt (SATD) in Kaggle competition notebooks. The process was structured in three main stages: data collection, SATD identification and classification, and statistical analysis.

3.1. Data Collection

The dataset for this study was composed of 3,600 Kaggle competition notebooks, publicly available on the platform. These notebooks were selected to represent three main domains of machine learning practice — Tabular data, Computer Vision (CV), and Natural Language Processing (NLP) — ensuring diversity in data types, modeling approaches, and code organization styles.

In the Tabular domain, notebooks were collected from Titanic – Machine Learning from Disaster, considered the “hello world” of Kaggle for its simplicity and educational focus, and Santander Customer Transaction Prediction, a large-scale binary classification challenge using anonymized financial data, which reflects more mature and performance-oriented workflows.

For the Computer Vision (CV) domain, the dataset included notebooks from Digit Recognizer, based on the classic MNIST dataset and widely used for introductory image classification tasks; Cassava Leaf Disease Classification, a real-world multiclass classification problem requiring extensive data augmentation and preprocessing; and HuBMAP – Kidney Segmentation, a complex biomedical image segmentation challenge where model architecture and computational efficiency play a critical role.

Finally, the Natural Language Processing (NLP) domain comprised notebooks from Natural Language Processing with Disaster Tweets, a straightforward text classification task; Jigsaw Unintended Bias in Toxicity Classification, a more advanced problem addressing ethical and bias-related aspects in textual data; and CommonLit Readability

Prize, a regression task based on text readability prediction, which differs from the typical classification structure of most NLP challenges.

This combination of competitions provided a balanced sample across different levels of difficulty, coding practices, and problem structures, allowing for a comprehensive analysis of Self-Admitted Technical Debt (SATD) in diverse machine learning contexts.

3.2. Identification of Self-Admitted Technical Debt

SATD instances were identified following the definition proposed by [Potdar and Shihab 2014], in which SATD corresponds to explicit developer comments that indicate awareness of temporary, incomplete, or suboptimal solutions (e.g., TODO, FIXME, HACK).

The extraction process involved automated comment mining from all code cells in the notebooks. Comments containing SATD indicators were then filtered and analyzed. A hybrid identification approach was applied, combining:

- Automated detection using keyword-based heuristics and a pre-trained natural language model for contextual classification.
- Manual validation to confirm true SATD occurrences and discard false positives (e.g., comments unrelated to technical debt).

3.3. Classification of SATD

After the identification phase, the extracted comments were automatically analyzed and classified using an AI-based approach. The classification process was conducted through an intelligent agent powered by the ChatGPT-4.0 mini language model, capable of interpreting the semantic and contextual meaning of developer comments to assign them to the most appropriate Self-Admitted Technical Debt (SATD) category.

Following data collection, all detected SATD instances were divided into four JSON files, each containing 60 technical debt comments grouped by keyword — TODO, HACK, WORKAROUND, and FIXME. Each file represented a separate batch processed by the AI agent. For each batch, the model analyzed the text of the comment, identified the underlying rationale or issue, and assigned one of the nine SATD categories defined in *23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software* [Bhatia et al. 2023].

To ensure traceability and facilitate collaborative validation, all AI-generated classifications were automatically recorded in a shared Google Sheets document, storing the original comment, the predicted category, and metadata such as the source notebook and competition domain. The authors manually reviewed a subset of these entries to confirm consistency and correct possible misclassifications, ensuring the reliability of the automated categorization process.

3.4. Metrics and Data Analysis

The analysis was guided by three Research Questions (RQs) and their associated metrics, as defined in Section 3.1:

- **RQ1: Frequency and distribution of SATD categories.**
  - **M1.1:** Percentage distribution of SATD categories.

  - **M1.2:** SATD density per notebook (instances per file).
- **RQ2: Correlation between SATD and competition success or visibility.**
  - **M2.1:** SATD density vs. leaderboard position and upvotes.
  - **M2.2:** Correlation between SATD type and competition medals (Gold, Silver, Bronze).
  - **M2.3:** Correlation between SATD quantity/type and notebook popularity (upvotes).
- **RQ3: Variation of SATD across competition domains.**
  - **M3.1:** Average SATD density by domain (Tabular, NLP, CV).
  - **M3.2:** Comparative distribution of categories across domains.
  - **M3.3:** Average SATD density per notebook within each domain.

Statistical analysis was performed to identify correlations between SATD density and competition outcomes, as well as to compare SATD distribution among the different ML domains.

3.5. Goal and Research Questions

To achieve the specific goals defined for this study, the following Research Questions (RQs) are assessed:

- **RQ1:** What are the most frequent and prevalent categories of Self-Admitted Technical Debt (SATD) in machine learning competition notebooks on Kaggle?
- **RQ2:** How does the prevalence and type of SATD correlate with competition success metrics (medals) and notebook visibility (upvotes)?
- **RQ3:** How does the distribution of SATD categories vary across different competition domains (e.g., Tabular, Natural Language Processing, and Computer Vision)?

The general goal of this study is to perform a large-scale empirical analysis of Self-Admitted Technical Debt (SATD) in Kaggle competition notebooks, in order to understand its characteristics, causes, and implications for machine learning development practices.

To this end, we adopt the definition of SATD proposed by [Potdar and Shihab 2014] for identifying SATD instances in code comments and apply the nine-category taxonomy introduced by [Bhatia et al. 2023] in *23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software* to classify them.

4. Results

5. Conclusion

References

(2025). Kaggle notebooks documentation.

Bhatia, A., Khomh, F., Adams, B., and Hassan, A. E. (2023). An empirical study of self-admitted technical debt in machine learning software. *arXiv preprint arXiv:2311.12019*.

Di Salle, A., Rota, A., Nguyen, P. T., Di Ruscio, D., Fontana, F. A., and Sala, I. (2022). Pilot: synergy between text processing and neural networks to detect self-admitted technical debt. In *Proceedings of the International Conference on Technical Debt*, pages 41–45.

Kaggle (2025). Competitions documentation. `https://www.kaggle.com/docs/competitions`. Acessado: 27 Out 2025.

Kirda, E. (2017). Unveil: a large-scale, automated approach to detecting ransomware (keynote). In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)*, pages 1–1. IEEE Computer Society.

Melo, A., Fagundes, R., Lenarduzzi, V., and Santos, W. B. (2022). Identification and measurement of requirements technical debt in software development: A systematic literature review.

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

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28.

Tang, Y., Khatchadourian, R., Bagherzadeh, M., Singh, R., Stewart, A., and Raja, A. (2021). An empirical study of refactorings and technical debt in machine learning systems. In *2021 IEEE/ACM 43rd international conference on software engineering (ICSE)*, pages 238–250. IEEE.

Xiao, T., Wang, D., McIntosh, S., Hata, H., Kula, R. G., Ishio, T., and Matsumoto, K. (2021). Characterizing and mitigating self-admitted technical debt in build systems. *IEEE Transactions on Software Engineering*, 48(10):4214–4228.