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.

1. Introduction

Technical debt (TD) is a metaphor in software engineering that describes the long-term effects that originate from alternative 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 root causes, map identification strategies, and provided metrics for TD. However, even with many studies addressing how to identify and assess 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 relevant for empirical study because these developer-written labels are observable and often correlate with later maintenance effort or system degradation. Early exploratory work established methods for mining SATD data from source comments and demonstrated its prevalence across projects. Large-scale studies and recent work focused on machine learning (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]

To perform this analysis, Kaggle was identified as being able to provide the most fitting dataset. Kaggle is an online platform for data science and machine-learning use 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. Due to the fact that 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 data mining research (including SATD) in data science work. [kag 2025]

Kaggle competitions present a problem statement and a dataset. Participants develop models (often inside Notebooks) that 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 real-world 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 distribution 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. Reaserch Goals

2.1. 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)?
RQ1. How do the 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 [OBrien et al. 2022] in *23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software* to classify them.

2.2. Metrics and Data Analysis

To address the three research questions (RQs) defined in this study, a set of quantitative and qualitative metrics was established to guide the analysis of Self-Admitted Technical Debt (SATD) across Kaggle competition notebooks. These metrics were designed to capture both the prevalence of SATD and its relationship with competitive performance and domain characteristics.

For **RQ1**, which investigates the most frequent and prevalent categories of SATD, we measured the relative distribution of each SATD category across all notebooks, as well as the overall density of SATD per notebook (i.e., the number of SATD instances normalized by the total number of code comments). This enabled the identification of which types of debt are most commonly self-admitted by participants and provided an overview of the general debt landscape within competition notebooks.

For **RQ2**, which explores how SATD relates to competition outcomes, we analyzed correlations between SATD density and multiple success indicators on the Kaggle platform. Specifically, we examined the relationship between the number and type of SATD instances and participants' leaderboard positions, medals (Gold, Silver, Bronze), and community visibility metrics such as upvotes. These analyses aimed to determine whether higher-performing or more visible notebooks exhibit different patterns of technical debt compared to less successful submissions.

Finally, for **RQ3**, which investigates how SATD distribution varies across competition domains, we compared the prevalence and composition of SATD categories in three major areas of machine learning practice: Tabular, Natural Language Processing (NLP), and Computer Vision (CV). This comparison involved calculating the average SATD density per notebook within each domain and examining the proportional distribution of categories across them. The goal was to understand whether certain forms of technical debt are more characteristic of specific types of ML tasks or data modalities.

All quantitative analyses were complemented by descriptive statistics and correlation tests to identify significant patterns and relationships. This combined approach allowed for a comprehensive understanding of how different forms of self-admitted technical debt manifest in Kaggle competition notebooks, how they relate to developer success, and how they vary across distinct machine learning domains.

3. Related Work

Research on Technical Debt (TD) has evolved trying to from trying to establish a formal concept, to extensive empirical analyses that examine its impact on software quality, evolution, and maintenance. [Melo et al. 2022] conducted a systematic review on Requirements Technical Debt, identifying and measuring its manifestations throughout the development process.

An essential study by [Potdar and Shihab 2014] introduced a systematic approach for detecting Self-Admitted Technical Debt (SATD) in source code by mining developer comments that explicitly acknowledge temporary, incomplete, or suboptimal solutions. Their definition of where SATD represents explicit developer-written statements indicating known technical debt is adopted in this work to identify SATD instances within Kaggle competition notebooks.

Following research expanded on this foundation by incorporating automation and large-scale analysis [Kirda 2017] applied natural language processing (NLP) techniques using word embeddings for SATD classification, while [Di Salle et al. 2022] developed PILOT, a deep-learning framework combining text preprocessing and neural networks to detect SATD with higher accuracy. These methods enabled finer-grained detection and categorization of technical debt indicators embedded in source comments.

As machine learning (ML) systems became increasingly common, researchers began to investigate how technical debt manifests within data-driven and model-centric development processes. The fundamental work *Hidden Technical Debt in Machine Learning Systems* by [Sculley et al. 2015] outlined several unique forms of debt that emerge in ML pipelines—such as data, configuration, and model-dependency debt—highlighting how ML-specific characteristics complicate long-term software maintainability.

Building on this perspective, the work *23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software* [OBrien et al. 2022] provided the most comprehensive taxonomy of SATD specific to ML systems to date. The authors analyzed over 2,600 machine learning repositories and identified 23 distinct SATD types and later organized them into nine high-level categories that describe the main dimensions of technical debt in ML software:

1. **Data Dependency:** Issues related to data processing, storage, and configuration.
2. **Code Dependency:** Debts arising from external libraries or module dependencies.
3. **Awareness:** Knowledge gaps or uncertainties in understanding algorithms or design decisions.
4. **Modularity:** Weak separation of components or a lack of abstraction in ML code.
5. **Configurable Options:** Incomplete or rigid hyperparameter and model configuration.
6. **Scalability:** Debts stemming from prototype or non-optimized experimental code.
7. **Readability:** Poor code clarity and maintainability.
8. **Performance:** Inefficient or suboptimal implementations affect execution time and resource use.
9. **Duplicate Code Elimination:** Redundant or repeated code requires refactoring.

This taxonomy represents a major step toward contextualizing SATD within the realities of ML development, capturing both traditional software debts and new ML-specific concerns, such as data quality, pipeline complexity, and model understandability. Because of its domain relevance and hierarchical structure, this classification framework serves as the foundation for the categorization process used 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], further explore how refactoring activities and build processes interact with the presence and evolution of SATD, emphasizing the importance of debt management in ML-oriented software engineering.

Despite the increasing number of studies, no prior research has examined SATD within the context of competitive data-science notebooks, such as those on Kaggle. These environments are intrinsically fast-paced, time-constrained, and experimentation-driven—conditions that often collect SATD.

Therefore, this study extends previous work by analyzing SATD in Kaggle competition notebooks and applying the nine-category taxonomy proposed by [OBrien et al. 2022] to investigate its patterns, prevalence, and relationships with competition outcomes, including medals, leaderboard rankings, and community engagement.

4. Research Method

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

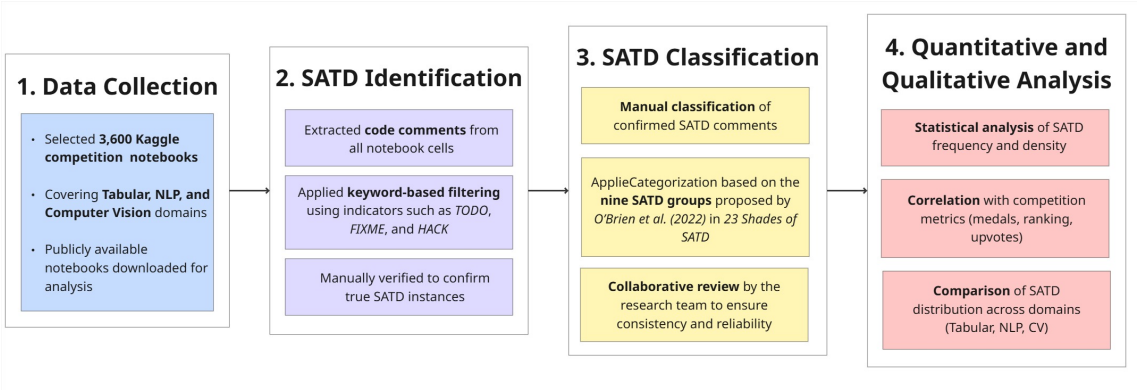

Figure 1. Study Design and Methodological Steps.

4.1. Data Collection

The dataset for this study consists 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). These categories ensure 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 an even sample across different levels of difficulty, coding practices, and problem structures, allowing for a comprehensive analysis of SATD in diverse machine learning contexts.

4.2. SATD Identification

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 information 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).

4.3. SATD Classification

After identifying all SATD instances, the classification process was conducted manually by the research team. Each extracted comment was individually analyzed and assigned to one of the nine SATD categories defined by [OBrien et al. 2022] in *"23 Shades of Self-Admitted Technical Debt: An Empirical Study on Machine Learning Software."*

The manual classification was carried out collaboratively, with all team members reviewing and discussing the categorization of comments to ensure consistency and reliability. The process relied on the semantic interpretation of each comment, taking into account its context and the technical issue it referred to.

This manual approach was chosen to guarantee contextual precision and to avoid biases that could arise from automated or AI-based classifiers. Each SATD instance was thus evaluated based on its meaning within the code, ensuring that the categorization faithfully reflected the developers' expressed technical concerns.

5. Results

6. 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.

OBrien, D., Biswas, S., Imtiaz, S., Abdalkareem, R., Shihab, E., and Rajan, H. (2022). 23 shades of self-admitted technical debt: an empirical study on machine learning software. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 734–746.

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.