Journal of Universal Computer Science, vol. 32, no. 7 (2026), 985-1015
submitted: 10/6/2025, accepted: 24/11/2025, appeared: 28/7/2026 CC BY 4.0

Predictive Modeling for Software Project Management - A Case Study of TensorFlow

Dung Hai Dinh
(Business Information Systems, Vietnamese-German University, Binh Duong, Vietnam
https://orcid.org/0000-0003-2033-9143, dung.dh@vgu.edu.vn)

Quan Nguyen Minh Tran
(Business Information Systems, Vietnamese-German University, Binh Duong, Vietnam
20221002@student.vgu.edu.vn)

Quang Huan Dong*
(Business Information Systems, Vietnamese-German University, Binh Duong, Vietnam
https://orcid.org/0000-0002-3085-3464, huan.dq@vgu.edu.vn)

Nicole Ondrusch
(Software Engineering and Management, University of Applied Sciences Heilbronn, Heilbronn, Germany
https://orcid.org/0009-0007-8824-6919, nicole.ondrusch@hs-heilbronn.de)

The symbol * denotes the corresponding author.

Abstract: This study explores the application of predictive modeling to the TensorFlow project, a prominent open-source machine learning library. Our research aims to develop and validate predictive models that can accurately forecast issue closures, thereby enhancing project management decisions. We systematically collect data from the TensorFlow GitHub repository through data crawling using a self-built python script, focusing on key metrics such as commit history, issue tracking, and contributor metrics. The data comprises 120,000 commit messages and 25,053 issue tickets. We apply descriptive analytics to identify insights into the project's development dynamics and implement nineteen models using various machine learning techniques, including linear regression, ridge regression, and ensemble methods. The final model, based on Lasso regression, achieves high accuracy with the lowest Mean Squared Error (MSE), the Average Cross-Validation MSE and highest R-squared value compared to other models. The final model also utilizes advanced feature engineering techniques, such as lagged features and cyclical time transformations, which help enhance the model's performance. Applied time-aware validation shows the need to further refine the evaluation design to enhance temporal modeling.

Keywords: Predictive modeling, Software project management, Issue closure forecasting, TensorFlow, Machine learning
Categories: D.2.7, D.2.8, D.2.9, I.2.6, K.6.3
DOI: 10.3897/jucs.161668

1 Introduction

In the evolving landscape of technology, software development plays a pivotal role in driving innovation and efficiency across industries. Effective project management and,

particularly, effective test management for software development is crucial in navigating the complexities of software projects, where teams must adapt to rapidly changing requirements and ensure timely delivery of high-quality software.

Despite the critical importance of software test management, traditional methods in test management – as in software project management in general - often rely on manual or simplistic algorithmic evaluations. These methods may struggle to predict and adapt to the dynamic nature of the software development process, leading to delays and suboptimal outcomes due to outdated models, inconsistencies, and limitations in both reactive and proactive adaptation strategies [Rudy, 2012]. However, agile methods, while more adaptable, still face issues like incomplete iterations and high computation costs for managing adaptation rules [Liu et al., 2018].

This paper applies predictive models to analyze and forecast key project metrics—such as issue resolution times, commit frequencies, and contribution patterns—using the TensorFlow open-source project as a case study. By leveraging GitHub data, we evaluate the effectiveness of various modeling techniques in predicting issue closures, aiming to improve test management practices. While focused on TensorFlow, our broader goal is to demonstrate how these methods can generalize to other aspects of project dynamics. Accurate forecasts support better resource allocation, sprint planning, and early bottleneck detection, offering actionable insights for managing large-scale software projects. The following points highlight the key contributions of this work:

- Enhanced Predictive Modeling Framework: a robust predictive modeling framework is developed that incorporates advanced feature engineering techniques, such as lagged features and cyclical time transformations. This framework improves the accuracy and interpretability of predictive models, making them more applicable to real-world software project management scenarios.
- Application to Large-Scale Open-Source Projects: a critical gap in the literature is addressed by applying predictive modeling techniques to a large-scale, open-source project. TensorFlow, with its complex development environment and extensive contributor base, provides a challenging and relevant case study. The findings demonstrate the scalability and effectiveness of predictive models in such dynamic environments.
- Practical Insights for Project Management: The predictive insights generated from the developed models offer practical applications for sprint planning, resource allocation, process efficiency, and performance monitoring. Project managers can use these insights to make data-driven decisions, optimizing resource use, and improving project outcomes.
- Contribution to Methodological Advancements: The importance of advanced feature engineering and model selection in predictive analytics is highlighted. The iterative development and validation of models, including the use of Lasso regression and ensemble methods, contribute to methodological advancements in the field. These insights can inform future research and practical applications in software project management.

The remaining of the paper is organized as follows: Section 2 reviews the literature on software project management and predictive modeling. Section 3 details the methodology employed, including data collection, descriptive analysis, and model

development. Section 4 presents the results and discusses their implications for project management. Section 5 concludes with the key findings and potential areas for future research.

2 Related Work

This section reviews prior research on predictive modeling in software project management, outlining commonly used techniques and highlighting a key gap in current understanding that motivates the need for additional study.

2.1 Predictive Modeling in Software Project Management

Predictive modeling plays a key role in software project management by improving outcome forecasting and operational efficiency. Model effectiveness varies by task: linear regression is commonly used for predicting issue resolution and workload due to its simplicity and accuracy with continuous data [Schützko and Timinger, 2023], while Random Forest and GBMR excel in complex tasks like resource allocation. GBMR offers high accuracy and data-type handling but can be computationally expensive and prone to overfitting without proper tuning [Rai et al., 2021, Tosun et al., 2020].

In agile settings, predictive models support sprint planning and productivity by forecasting issue closure rates, while also helping anticipate delays and budget overruns to improve project execution [Schützko and Timinger, 2023, Christenson Jr. and Goldstein, 2022, Ghasemaghaei et al., 2018].

2.2 Methods in Predictive Modeling

Predictive models in software project management often use machine learning techniques such as linear regression, ridge regression, and ensemble methods like Random Forest and GBMR [Yang et al., 2020, Ramírez-Mora et al., 2020]. Linear models are valued for their simplicity and interpretability, making them suitable for estimating metrics like issue resolution time when variable relationships are straightforward. Ridge regression adds regularization to manage multicollinearity, enhancing stability and accuracy [Giacomelli et al., 2018, Rai et al., 2021, Zhang & Yang, 2017].

Ensemble methods, though less interpretable and more resource-intensive, are effective for capturing complex, non-linear relationships [Wang and Ye, 2020]. Random Forest reduces overfitting by averaging decision trees, while GBMR improves accuracy by sequentially correcting errors [Shukla and Kumar, 2023, Varshini and Kumari, 2023]. Techniques like stacking and bagging further enhance performance and robustness in managing complex projects [Haris et al., 2023, Rokach, 2010].

Advanced feature engineering boosts model effectiveness by transforming raw data into informative inputs. Lagged variables help capture temporal dependencies; cyclical time features model periodic behaviors; and Natural Language Processing (NLP) techniques extract insights from unstructured data like commit messages and issue descriptions, improving model precision [Malhotra and Lata, 2020, Elish et al., 2015, Idri et al., 2016, Hosni et al., 2021].

Software repository mining supports these models by analyzing data from version control and issue tracking systems. Studies on commit references and developer

interactions offer empirical insights into development dynamics, aiding project managers in making data-driven decisions [Xiao et al., 2021, Hata et al., 2023].

2.3 Current Research Gap

While predictive modeling has proven useful in software project management, research has largely focused on small, controlled environments. Limited attention has been given to large-scale, open-source projects like TensorFlow, which involve complex, dynamic development and diverse contributors [Toma and Wei, 2023]. This study addresses that gap by using TensorFlow as a case study to evaluate the potential and limitations of predictive modeling in such contexts.

Further, the impact of advanced feature engineering—such as NLP on commit messages and lagged variables for temporal patterns—remains underexplored [Yang et al., 2020, Schuh et al., 2017, Malhotra and Lata, 2020]. These techniques could significantly enhance model accuracy and insight generation.

Interpretability is another challenge. While advanced models offer high accuracy, they often lack transparency, limiting their practical use by project managers [Schützko and Timinger, 2023]. Additionally, existing models often fail to adapt to evolving project conditions, leading to accuracy degradation over time [Guillaume-Joseph and Wasek, 2015, Kim et al., 2016, Yang et al., 2020]. To address these gaps, our study explores adaptive, interpretable predictive modeling within TensorFlow to support effective project management in dynamic, large-scale software environments.

3 Methodology

The methodology comprises three main components: data collection, descriptive analysis, and predictive modeling.

The data collection phase involves gathering publicly available data from the TensorFlow GitHub repository [TensorFlow Developers, 2015]. The TensorFlow project on GitHub serves as the case study for data collection efforts. This project is a prominent open-source machine learning library, and its active development and large contributor base make it an ideal subject for analyzing software project management through predictive modeling. This phase is essential for constructing a comprehensive dataset that precisely captures the development dynamics of TensorFlow. The dataset includes commit and issue metadata such as commit SHA, contributor username, timestamps, and issue details. Contributor names are retained as they are publicly visible on GitHub and directly associated with their commits, consistent with common practice in software repository mining research. Data have been omitted, anonymized or aggregated where appropriate. Nevertheless, the analysis focuses on collective patterns rather than individual behaviors. To enhance reproducibility, the code, data divisions, and hyperparameter details are available at [Dinh et al., 2025].

The descriptive analysis phase provides an essential overview of the TensorFlow project's development activities. This analysis helps in understanding the project's dynamics, including commit history, issue tracking, and contributor engagement. These insights form the foundation for subsequent predictive modeling.

The predictive modeling phase utilizes the collected data to develop models capable of forecasting future project trends. This involves applying various machine

learning techniques to identify patterns and predict outcomes such as the number of issues likely to be resolved in subsequent months. The predictive models are iteratively refined through feature engineering and model selection to achieve the highest possible accuracy and robustness.

Together, these components form a robust framework for analyzing and predicting the development trajectory of TensorFlow, providing valuable insights into effective software project management.

3.1 Data Collection

Data collection involves identifying key metrics, using suitable tools for extraction, and ensuring accuracy and completeness. To analyze TensorFlow's development, three key metric categories were used. First, Commit History captures development patterns, contributor collaboration, and update frequency. Metrics like commit count, frequency, and messages offer insights into project dynamics and cycles [Koch and Neumann, 2008, Allamanis and Sutton, 2013]. Second, Issue Tracking evaluates resolution time, recurring challenges, and team responsiveness, helping assess project management efficiency [Murgia et al., 2009, Saini et al., 2022]. Third, Contributor Metrics assess involvement based on commits, lines of code, and their impact, revealing individual contributors' influence [De Bassi et al., 2018, Ndenga et al., 2015].

The data collection framework includes involves six following steps:

- Sources Identification: Relevant data sources are selected from the TensorFlow GitHub repository, including commit history, issue tracking, and contributor metrics [Emanuel et al., 2013].
- Data Extraction: The GitHub API is utilized to systematically collect data. A dedicated script was developed to automate this process.
- Authentication: The script uses a GitHub Personal Access Token (PAT) for authentication, allowing efficient and higher rate-limit access compared to unauthenticated requests [Paulson et al., 2004].
- Rate Limiting and Pagination: GitHub's API imposes rate limits to ensure fair usage. The script manages these limits by incorporating rate limiting handling and pagination to fetch data across multiple pages [Foucault et al., 2015].
- Data Fetching: The core functionality involves fetching data from GitHub's API endpoints. The script focuses on collecting commits and issues from the TensorFlow repository due to its prominence and activity level [Tomas et al., 2013].
- Efficiency and Scalability: To manage large datasets, the script limits the number of items fetched and implements a delay between requests to comply with GitHub's rate limiting policies [Emanuel et al., 2011].

This approach follows the methods used in various studies that have demonstrated the effectiveness of these metrics in evaluating open-source software projects [Seker et al., 2021, Norick et al., 2010, Sunindyo et al., 2012, Bakar, 2015]. After data collection, preprocessing is essential to ensure the data is suitable for analysis. First, Data Cleaning involves tidying up commit messages and issue titles by removing newline and carriage return characters, which can disrupt analysis. Second, Data Transformation includes converting timestamps into a standardized datetime format to facilitate time-series analyses and the chronological ordering of events. Third, Feature Engineering entails extracting useful information from raw data, such as categorizing commit messages

based on their content or tagging issues with labels indicating their nature (e.g., bug, enhancement, documentation).

Based on this multi-step process, a data dictionary was developed to define the extracted attributes, as shown in Table 1.

Feature	Description	Type
date	The date of the commit or issue event	DateTime
month_year	The month and year extracted from the date	Period
contributor	The identifier for the contributor making a commit	String
created_at	The date an issue was created	DateTime
closed_at	The date an issue was closed	DateTime
labels_count	The number of labels assigned to an issue	Integer

Table 1: Data dictionary

By following this detailed data collection and preprocessing methodology, we ensure that the data used in our analysis is accurate, comprehensive, thereby suitable for predictive modeling, forming a robust foundation for subsequent steps in our study.

3.2 Descriptive Analysis

The descriptive analysis phase involves summarizing and visualizing the collected data to provide an overview of TensorFlow's development activities. This includes analyzing commit history, issue tracking, and contributor metrics to understand the project's dynamics and identify patterns.

- Commit History: This analysis includes examining the frequency of commits, identifying periods of high and low activity, and correlating these with project milestones such as major releases or sprints. Visualization tools such as time-series plots are used to depict these patterns.
- Issue Tracking: This involves calculating the average and median issue resolution times, analyzing the distribution of issue resolution times, and identifying factors that influence resolution efficiency. Histograms and other visual tools are used to illustrate these findings.
- Contributor Metrics: This entails analyzing the number of commits by each contributor, identifying key contributors, and understanding their impact on the project's development. The distribution of contributions among contributors is visualized to highlight collaboration dynamics.

3.3 Predictive Modelling

The predictive modeling phase utilizes the collected and analyzed data to develop models capable of forecasting future project trends. This involves applying various machine learning techniques to identify patterns and predict outcomes such as the number of issues likely to be resolved in subsequent months. The predictive models are iteratively refined through feature engineering and model selection to achieve the highest possible accuracy and robustness. Following the methods discussed in various studies on predictive modeling in software development, such as [Malhotra, 2017] or [Hao and Zhang, 2011], the effectiveness of these models is assessed using performance

metrics that evaluate the accuracy, consistency, and explanatory power. The performance metrics employed are Mean Squared Error (MSE), Average Cross-Validation MSE (Average CV MSE), and Coefficient of Determination R-squared (R^2).

- MSE: This metric measures the average squared difference between the actual and predicted values. In our models, MSE is expressed in "squared number of issues," providing a direct measure of prediction accuracy [Shmueli et al., 2016].
- Average CV MSE: This metric averages the MSE across multiple cross-validation folds, ensuring the model's consistency and generalizability. It is also expressed in "squared number of issues".
- R^2: A unitless metric that quantifies the proportion of variance in the target variable explained by the model. An R^2 value close to 1 indicates a high level of explanatory power [Balsamo et al., 2004].

In the following, we present the development and evaluation of various predictive models through two experiments, focusing on enhancing the prediction of issue closure rates. The use of these metrics and the associated methods aligns with best practices in predictive modeling for software engineering.

3.3.1 Experiment 1 - Evaluating Predictive Models in Project Management

Experiment 1 initiated a comprehensive exploration of predictive modeling within the TensorFlow project management context. The experiment evaluated a series of ten distinct models, each designed with unique features and techniques aimed at enhancing the prediction of issue closure rates and other project dynamics. Below is an expanded description of each model and their respective outcomes, supported by a detailed table summarizing their performance metrics.

- Baseline Linear Regression Model (M1): This model used the issue close date as the sole input feature, establishing a baseline for comparison with subsequent models.
- Enhanced Linear Regression Model (M2): Improved upon M1 by incorporating both the creation and close dates of issues, aiming to capture a broader spectrum of the issue lifecycle.
- Ridge Regression Model (M3): Applied ridge regression to address potential overfitting by introducing L2 regularization, refined through extensive hyperparameter tuning.
- Advanced Machine Learning Techniques (M4 to M8): These models explored a range of machine learning techniques including Random Forest (M4), GBMR (GBM, M5), XGBoost (M6), LightGBM (LGBM, M7), and Support Vector Regression (SVR, M8). Each model aimed to leverage ensemble learning methods or advanced regression techniques to improve predictive accuracy.
- Feature Engineering models (M9 and M10): Both models incorporated advanced feature engineering, analyzing textual data from commit messages and issue labels to enhance model input with qualitative data insights.

3.3.2 Experiment 2 - Enhanced Modeling with Advanced Feature Engineering

Experiment 2 built on insights from Experiment 1, focusing on advanced feature engineering to better capture TensorFlow's project dynamics. Key enhancements included Lagged Features from prior months, Cyclical Time Transformations, and Rich Contextual Data on contributors and issue labels, all aimed at improving the model's understanding of development patterns and issue resolution.

First, Lagged Features captures the influence of past activity on current outcomes. For instance, the previous month's closed issues (prev_monthly_closed) helped the model recognize continuity in issue resolution efforts. Similarly, the previous month's contributors (prev_contributors) indicated the number of active developers in the prior month, offering insight into team capacity and its potential impact on productivity. The previous month's average labels (prev_avg_labels) measured issue complexity by tracking how issues were categorized, contributing to a more nuanced representation of workload and focus.

Second, to account for temporal patterns, Cyclical Time Transformations incorporated sine and cosine transformations of the month variable, capturing the cyclical nature of software development activities. The Month Sine (month_sin) and Month Cosine (month_cos) features were derived from the month number to represent the cyclical trends in the data. The sine and cosine transformations helped the model understand seasonal variations, ensuring months close to each other in the calendar year had similar feature values.

Third, in addition to time-based and historical features, Rich Contextual Data was introduced to further enhance the model. The Number of Unique Contributors (contributors_per_month) feature served as a direct measure of team size and engagement, reflecting variations in development effort. Meanwhile, the Average Labels Per Issue (avg_labels_count) feature acted as a proxy for issue complexity and categorization trends. Together, these features allowed the model to better interpret project dynamics and make more informed predictions.

Based on these feature engineering techniques, an updated data dictionary (which is described in Table 2) will be used in experiment 2 listed, as follows, with the importance of new features also mentioned.

Feature	Description	Type	Importance
date	The date of the commit or issue event	DateTime	Original feature
month_year	The month and year extracted from the date	Period	Original feature
contributor	The identifier for the contributor making a commit	String	Original feature
created_at	The date an issue was created	DateTime	Original feature
closed_at	The date an issue was closed	DateTime	Original feature

labels_count	The number of labels assigned to an issue	Integer	Original feature
monthly_created_issues	The count of issues created in a particular month	Integer	Helps capture the influx of new issues that may affect the number of issues closed in the subsequent month
prev_monthly_closed	The number of issues closed in the previous month	Integer	Provides insight into the momentum of issue resolution, indicating whether the project is in a high-activity phase or not
prev_contributors	The number of unique contributors in the previous month	Integer	Correlates the available human resources to the number of issues closed
prev_avg_labels	The average number of labels per issue in the previous month	Float	Serves as a proxy for the complexity or categorization of issues, impacting the closure rate
quarter	The quarter of the year (1-4)	Integer (1-4)	Captures seasonal effects in issue closure rates
year	The year of the event	Integer	Accounts for long-term trends and changes in the project dynamics over the years
month_sin	Sine transformation of the month to capture cyclic patterns	Float	Helps model the seasonality in the issue closure rates effectively
month_cos	Cosine transformation of the month to capture cyclic patterns	Float	Complements the sine transformation to fully capture the cyclic nature of the monthly data

Table 2: Updated data dictionary

The dictionary was first used with M1 to take the role as the baseline, using the same algorithm from the best model in Experiment 1 and new feature engineering techniques.

- Baseline Model of Experiment 2 (M1): Built on the foundations of Experiment 1 but incorporated a refined set of features, including more nuanced temporal inputs.
- Enhanced Feature Model (M2): Utilized advanced feature engineering to represent the complexity of the software development process more accurately.
- Ridge and Lasso Models with Enhanced Features (M3 and M4): Both models incorporated regularization techniques to manage overfitting, with M4 using Lasso regression to achieve optimal performance through feature selection.
- Exploring Ensemble Methods and Automated Frameworks (M5 to M9): These models explored various machine learning techniques and automated frameworks like the GBM (M5), Random Forest (M6), Tree-based Pipeline Optimization Tool (M7 and M8) or Stacking (M9) to enhance predictive performance.

3.3.3 Time-Aware Validation and Baselines

To address potential temporal leakage and strengthen the credibility of the predictive framework, we employed a time-series aware evaluation strategy. Specifically, an expanding-window walk-forward validation was used: at step t, models are trained on all data up to month t, and predictions are made for $t+1$. This approach mirrors real-world deployment, where forecasts must be based only on historical information, thereby eliminating the possibility of future data influencing past predictions.

Alongside the original Lasso regression model, a comprehensive suite of baselines was introduced to contextualize performance:

- Naïve Heuristics: A last observation carried forward predictor (Naïve Last), and a seasonal repetition heuristic (Naïve Seasonal) that reuses the average from the same calendar month in previous years.
- Classical Statistical Models: ARIMA, SARIMAX (seasonal extension), and Holt–Winters exponential smoothing, designed to capture trend and autoregressive patterns in the data.
- Machine Learning Baselines: Random Forest, XGBoost, and LightGBM, applied in a walk-forward manner using the same engineered features as Lasso.
- Prophet: A forecasting model designed to handle time series data with strong seasonal pattern, designed to capture trend-seasonality decompositions frequently used in software project analytics.
- Original Lasso Model: The previously trained model from random-split cross-validation, here applied to a 12-month held-out set. This served as a diagnostic baseline to examine the impact of temporal leakage.

Performance was evaluated with three complementary metrics:

- MSE to penalize larger deviations,
- Mean Absolute Error (MAE) to capture average predictive error, and
- R^2 to assess explanatory power relative to naïve variance.

This expanded methodology ensured that our evaluation was both rigorous and comparable against simple heuristics and established statistical forecasting techniques.

In the following section, the findings are presented and critically discussed.

4 Result and Discussion

This section presents the findings from our data analysis and predictive modeling efforts. The results provide insights into TensorFlow's development dynamics, issue resolution patterns, and the predictive accuracy of our models. The discussion interprets these results, highlighting their implications for software project management.

4.1 Data Collection Result

The collected data is stored in a structured format, facilitating further processing and analysis. This structured data includes essential details such as commit messages, issue titles, and timestamps, encompassing 120,000 commits (see excerpt in Table 3) and 25,053 issues (see excerpt in Table 4).

No.	SHA	Contributor	Date	Message
1	bcb048c1d0c09c70629869bb0a45ba502f14e7ee	chaserileyroberts	2024-02-29 23:54:12+00:00	Introduce a HLO instruction for CollectiveBroadcast. PiperOrigin-RevId: 611622547
2	c4c8820ba3bd9b66f56b4806c884673fda293e4e	sparc1998	2024-02-29 23:43:43+00:00	Fixed a bug in MSA's recognition of loops for the memory-bound loop optimizer. [...]
[…]				
119999	a73f6277a390c2ad69ddc877139c4c17ea12325b	Concurrency Practitioner	2018-09-25 22:51:04+00:00	Merge branch 'master' of https://github.com/ConcurrencyPractitioner/tensorflow
120000	7630e9df4804a01f5dd0ab20d4c0bcfb58e45432	Concurrency Practitioner	2018-09-25 22:50:13+00:00	Fixing error

Table 3: Excerpt of commit dataset (with […] mark omissions)

No.	ID	Title	State	Labels	Created_at	Closed_at	Labels_count
1	2161173528	Any _working_ alternative to tf?	open	['type: performance']	2024-02-29 12:43:55+00:00		1
2	2160445130	cannot install tensorflow 2.14 through pip	closed	['type: build/install', […]	2024-02-29 05:53:28+00:00	2024-02-29 07:55:09+00:00	4
[…]							
25052	365131817	No module named '_tensorflow_wrap_toco'	closed	['comp:lite']	2018-09-29 13:17:45+00:00	2018-10-11 16:40:10+00:00	1
25053	365128254	AudioSpectrogram not support batch_size mode and no fbank op	closed	['stat:awaiting tensorflower']	2018-09-29 12:30:55+00:00	2019-02-23 00:06:07+00:00	1

Table 4: Excerpt of issue dataset (with […] mark omissions)

4.2 Descriptive Result

The following sub-sections—Commit History, Issue Tracking, and Contributor Metrics—provide a structured overview of the project's development timeline, issue management processes, and patterns of contributor involvement.

4.2.1 Commit History

As shown in Figure 1, the commit history of TensorFlow reveals periodic fluctuations in the number of commits per month, indicative of the typical ebbs and flows in software development cycles. Periods of high activity often correspond to sprints or major releases, while periods of lower activity may indicate phases of consolidation or planning. This pattern is consistent with the findings in several studies of open-source software projects. For instance, research indicates that commit behaviors, including frequency and size, often follow specific patterns and can be correlated with phases of active development and maintenance [Ma et al., 2013]. Additionally, studies have shown that the size and interval of commits can provide insights into the development process, reflecting periods of intensive coding or bug fixing [Ferreira et al., 2022]. Understanding these patterns helps in planning and managing software projects more effectively [Kolassa et al., 2013].

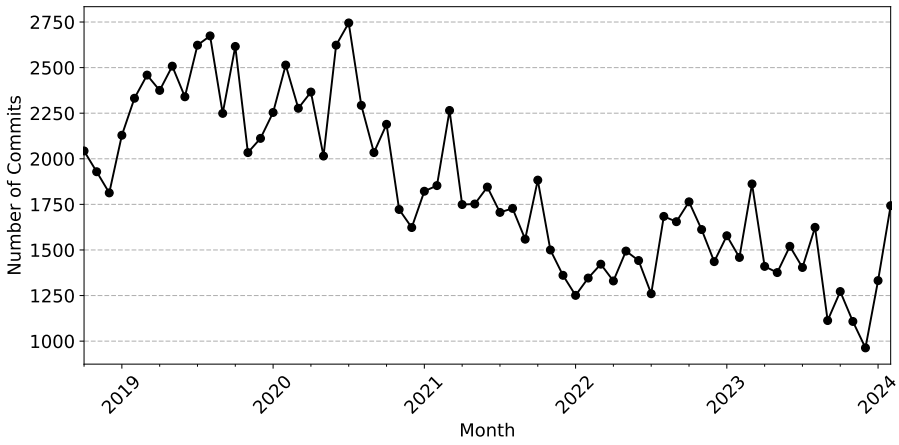

Figure 1: Commits per month

For instance, the analysis highlighted a sharp decline in commits in early 2024. Understanding whether this trend represents a seasonal variation, a strategic shift in the project, or external factors is essential for predictive modeling, as such trends can significantly impact the accuracy of forecasts.

4.2.2 Issue Tracking

Issue tracking is another critical aspect of project management that reflects the efficiency and responsiveness of the development team. Figure 2 illustrates the issue resolution time. The average issue resolution time for TensorFlow is approximately 112 days, with a median resolution time of about 17 days. This pattern is not unique to TensorFlow; it is observed across various open-source projects. Studies have shown that factors such as the complexity of the issue, the level of community engagement, and the use of effective categorization and labeling can influence resolution times [Choetkiertikul et al., 2017]. Additionally, the analysis of micro-process patterns in issue resolution can provide insights into the dynamics of the resolution process and help improve management strategies [Wang et al., 2020]. Understanding these factors can aid in developing more effective issue resolution policies and enhancing overall project management efficiency [Jahanshahi et al., 2020].

The histogram of issue resolution times, plotted on a logarithmic scale, reveals a right-skewed distribution. Most issues are resolved quickly, but there is a long tail of issues that take much longer to resolve. This pattern is vital for predictive modeling, as it highlights the need to account for outliers and complex issues in forecasting models.

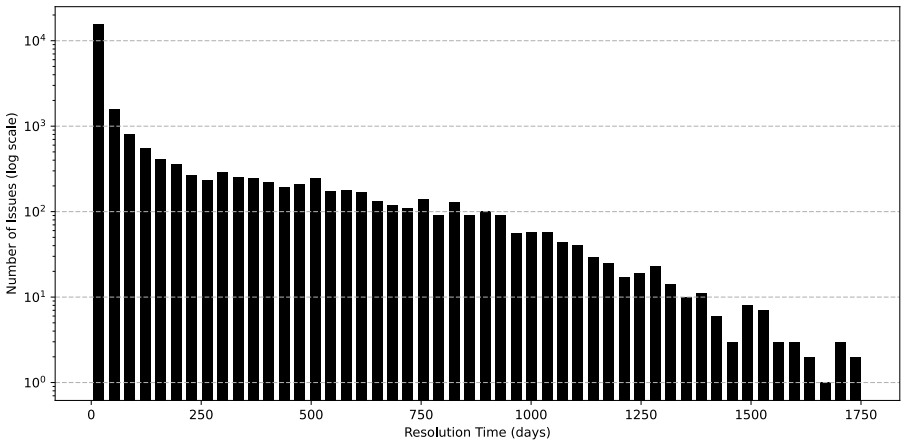

Figure 2: Issue resolution time

4.2.3 Contributor Metrics

Analyzing contributor metrics provides insights into the collaboration and contribution dynamics within the TensorFlow project. A core group of contributors is responsible for a substantial number of commits. Notably, one contributor, "tensorflower-gardener," dominates the commit history, as presented in Figure 3. This contributor might represent an automated process or a highly active individual.

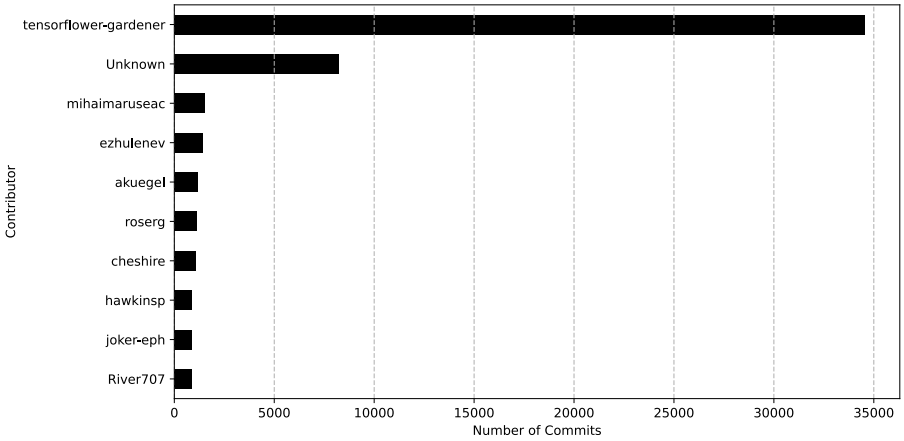

Figure 3: Top 10 contributors with outliers

Excluding this outlier, the distribution of contributions among the top contributors is even, indicating a collaborative effort, as shown in Figure 4. These insights are essential for predictive modeling, as understanding the roles and contributions of key contributors can help in forecasting future development activities.

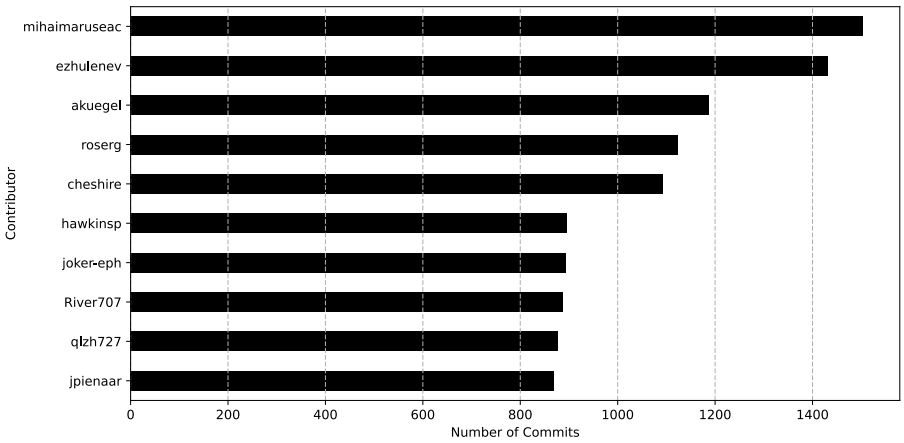

Figure 4: Top 10 contributors without outliers

4.3 Experiment Result

4.3.1 Experiment 1 Result

Experiment 1's results highlight the iterative process of model development and evaluation. Table 5 summarizes the performance of models from model M1 to M10.

Model ID	Model Type	Description	MSE	R-squared	Average CV MSE
M1	Baseline Linear Regression	Uses issue close date as the input feature.	5006.67	0.8806	6595.43
M2	Enhanced Linear Regression	Includes both issue creation and close dates as input features.	3847.42	0.9083	5476.13
M3	Ridge Regression	Incorporates L2 regularization to address model complexity.	3892.50	0.9072	5468.81
M4	Random Forest	Employs ensemble learning with decision trees.	5812.77	0.8614	8968.17
M5	Gradient Boosting Machine (GBM)	Uses boosting techniques to improve accuracy.	8233.49	0.8037	9646.06
M6	XGBoost	An optimized version of GBM for faster	9278.72	0.7788	13182.39

		performance and better accuracy.			
M7	LightGBM	Optimized for handling large datasets with faster training times.	8750.32	0.7914	9746.36
M8	Support Vector Regression (SVR)	Applies support vector machines for regression analysis.	5840.68	0.8607	4979.35
M9	GBM with Textual Features	Integrates GBM with textual analysis of commit messages and issues.	10399.38	0.7520	11801.06
M10	Ridge Regression with Text Features	Combines ridge regression with textual and categorical data analysis.	4151.38	0.9010	5658.40

Table 5: Experiment 1 model comparison of results

Experiment 1's results highlighted the iterative process of model development and evaluation. The baseline model, M1, employed a linear regression approach with the issue close date as the sole input feature. This model achieved an MSE of 5006.67 and an R^2 value of 0.8806. Building on this, the enhanced linear regression model, M2, incorporated both the creation and close dates of issues, resulting in a lower MSE of 3847.42 and a higher R^2 of 0.9083. To address potential overfitting, M3 applied ridge regression with L2 regularization, achieving an MSE of 3892.50 and an R^2 of 0.9072, indicating robustness against multicollinearity.

Advanced machine learning techniques were explored through models M4 to M8, including Random Forest, GBMR, XGBoost, LGBM, and SVR. While these models exhibited higher MSEs, reflecting varying levels of increased error, they provided valuable insights into the complex dynamics of project management. Specifically, M4 (Random Forest) had an MSE of 5812.77 and an R^2 of 0.8614, whereas M5 (GBM) had an MSE of 8233.49 and an R^2 of 0.8037. Models M6 (XGBoost) and M7 (LGBM) achieved MSEs of 9278.72 and 8750.32 with R^2 values of 0.7788 and 0.7914, respectively. The SVR model, M8, resulted in an MSE of 5840.68 and an R^2 of 0.8607.

Further enhancements were made in models M9 and M10 by incorporating advanced feature engineering, analyzing textual data from commit messages and issue labels. Model M9, which integrated GBM with textual features, showed an MSE of 10399.38 and an R^2 of 0.7520. Model M10, which combined ridge regression with textual and categorical data analysis, demonstrated a significant improvement with an MSE of 4151.38 and an R^2 of 0.9010. These results underscore the importance of combining quantitative and qualitative data analysis for enhancing predictive modeling in software project management [Hosni et al., 2021].

Experiment 1 highlights the extensive and iterative process undertaken, setting a robust foundation for subsequent experiments and modeling efforts. Each model contributed uniquely to the understanding and forecasting of project dynamics,

demonstrating the role of model selection, feature engineering, and methodological rigor in predictive analytics within software project management.

4.3.2 Experiment 2 Result

Experiment 2's results demonstrate the impact of advanced feature engineering on the model performance. Table 6 highlights the performance for each model.

Experiment 2's synthesis of advanced modeling demonstrated the crucial role of sophisticated feature engineering in enhancing predictive accuracy. The progression from M1 to M9 illustrated a narrative of continuous improvement and learning. The notable increase in R^2 values across the models, coupled with the reduction in MSE from M1 to M4, highlighted the growing alignment between predicted outcomes and actual historical data. For instance, M1 started with an MSE of 3225.19 and an R^2 of 0.9224, while M4 achieved an MSE of 1357.56 and an R^2 of 0.9673. These improvements underscore the importance of carefully engineered features in capturing the essential patterns within the data, leading to more accurate and reliable predictions. This aligns with studies showing the impact of feature engineering in predictive modeling across various domains [Khurana et al., 2017, Nargesian et al., 2017, Krause et al., 2014].

The best model, M4, employed Lasso regression, which was selected for its balance of simplicity, interpretability, and predictive accuracy. With an MSE of 1357.56 and an R^2 value of 0.9673, M4 significantly outperformed the other models in Experiment 2. This model incorporated advanced feature engineering techniques such as lagged features, unique contributors, average labels per issue, and cyclical transformations of time. The success of M4 underscores the effectiveness of combining in-depth feature engineering with strategic model selection. This is supported by research highlighting the effectiveness of automated and manual feature engineering methods in improving model performance [Oyamada, 2019].

Model ID	Model Type	Description	MSE	R-squared	Average CV MSE
M1	Baseline Model of Experiment 2	Refined features based on insights from Experiment 1.	3225.19	0.9224	4508.17
M2	Enhanced Feature Model	Advanced feature engineering capturing temporal and contextual data.	1365.16	0.9671	4792.38
M3	Ridge Regression with Advanced Features	Applied ridge regression with a focus on regularization.	5109.30	0.8771	7496.08
M4	Lasso Regression with	Utilized Lasso regression for optimal feature selection.	1357.56	0.9673	4625.11

	Advanced Features				
M5	GBM with Advanced Features	Employed Gradient Boosting Machine with sophisticated features.	10005.74	0.7593	8223.12
M6	Random Forest with Advanced Features	Used Random Forest with an enhanced set of features.	4430.95	0.8934	6144.77
M7	TPOT with Advanced Features	Automated machine learning exploring pipelines.	1495.39	0.9640	5749.97
M8	TPOT Mixed Features	Mixed feature set explored through TPOT.	1798.96	0.9567	7862.44
M9	Stacking with Advanced Features	Combined multiple models through stacking to enhance predictions.	2848.39	0.9315	6233.90

Table 6: Experiment 2 model comparison of results

In contrast, the more complex machine learning models did not achieve the same level of predictive accuracy. For instance, Model M6, which used the TPOT-based automated machine learning technique, resulted in an MSE of 1495.39 and an R^2 of 0.9640, indicating a lower performance compared to M4. Similarly, Model M5, which utilized GBM, resulted in a much higher MSE of 10005.74 and an R^2 of 0.7593. This suggests that the complexity of the model did not necessarily translate to better performance. The complex models often struggled to capture the underlying patterns as effectively as the well-engineered simpler models. This finding highlights that more sophisticated models are not always superior, and that the quality of feature engineering plays a pivotal role in the success of predictive models [Zhu et al., 2013].

Overall, experiment 2 not only improved the predictive metrics but also provided valuable insights into the importance of feature engineering and model selection. The clear superiority of the Lasso regression model (M4) with its carefully engineered features exemplifies how thoughtful, strategic approaches can lead to the development of highly effective predictive tools for software project management. This experiment serves as a testament to the fact that combining simplicity with robust feature engineering can yield powerful results in predictive analytics [Tran et al., 2014, Nesamani and Rajini, 2021].

4.4 Final Model

The final model was constructed using Lasso regression, selected for its balance of simplicity, interpretability, and predictive accuracy. This model incorporated advanced feature engineering, including lagged features, unique contributors, average labels per issue, and cyclical transformations of time.

4.4.1 Mathematical Formulation

The Lasso regression model predicts the number of closed issues in month $(i)((\widehat{y_i}))$ using the following equation:

$$\widehat{y_i} = \beta_0 + \beta_1 \cdot \text{monthly_created_issues}_i + \beta_2 \cdot \text{prev_monthly_closed}_i + \beta_3 \cdot \text{prev_contributors}_i + \beta_4 \cdot \text{prev_avg_labels}_i + \beta_5 \cdot \text{quarter}_i + \beta_6 \cdot \text{year}_i + \beta_7 \cdot \text{month_sin}_i + \beta_8 \cdot \text{month_cos}_i$$

where:
- (β_0): Intercept term.
- $(\beta_1, \beta_2, \ldots, \beta_8)$: Coefficients for each respective feature.

The coefficients (β) are estimated by minimizing the following objective function:

$$\min_{\beta_0,\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

where:
- n is the number of observations (months).
- y_i is the actual number of closed issues in month i.
- λ is the regularization parameter that controls the strength of the penalty.

4.4.2 Performance Metrics

With an MSE of 1357.56 and an R^2 value of 0.9673, the final model indicates an elevated level of accuracy, with the model explaining a substantial proportion of the variance in the number of issues closed. The final model's performance was validated through a series of predictive tests, comparing predicted values against actual closed issues over time.

4.4.3 Model Validation

Figure 5 visualizes the alignment between the predicted and actual closed issues, which is a testament to the model's precision, particularly effective in the middle portion of the time series and maintaining reasonable accuracy even during periods of high volatility.

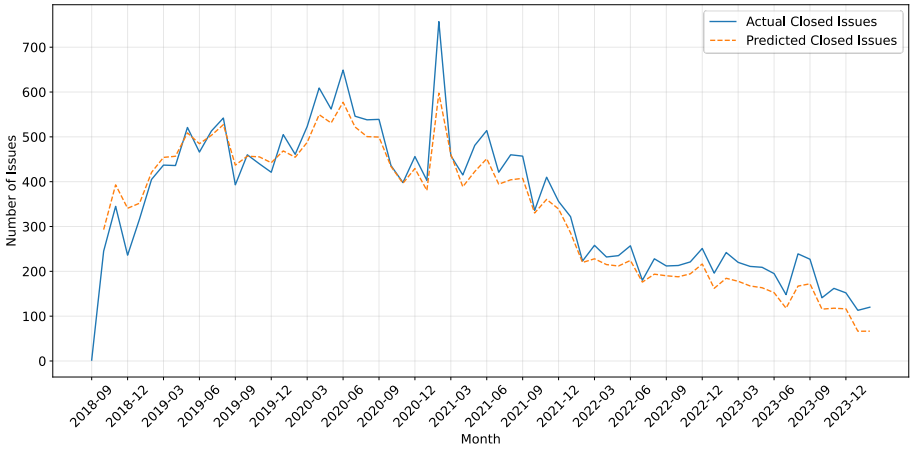

Figure 5: Actual vs. predicted closed issues over time

4.4.4 Residual Analysis

Figure 6 presents the residuals, or the differences between actual and predicted values. These are well-distributed without obvious patterns, indicating no systematic prediction errors. This is essential for a trustworthy predictive model. The importance of residual analysis in ensuring model reliability and detecting potential issues has been highlighted in various studies. For example, residual analysis is crucial for evaluating the validity of statistical models and improving their accuracy [Feng et al., 2020]. Properly distributed residuals indicate that the model's assumptions hold true and that it is not missing any critical patterns in the data. Thus, achieving well-distributed residuals is a fundamental step in confirming the robustness and effectiveness of a predictive model [Chen et al., 2020].

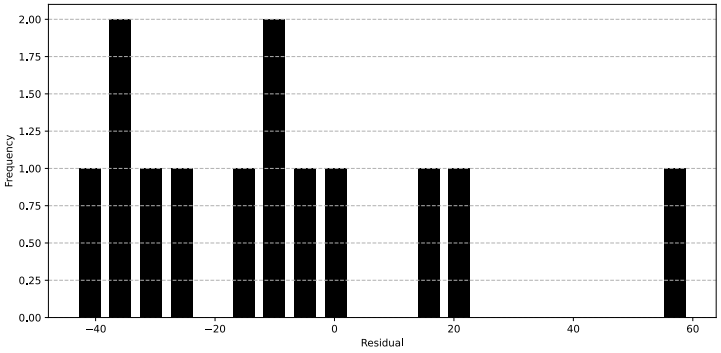

Figure 6: Residual distribution

4.4.5 Walk-forward Validation Results

The comparative evaluation results are summarized in Table 7 from highest to lowest, with model-level metrics. The table reports MSE, MAE, and R^2 across all models, highlighting the best-in-class values in italic.

Model	**MSE**	**MAE**	**R-squared**
ARIMA	6857.08	51.20	*0.6220*
Naïve Last	7792.30	54.70	0.5705
Lasso (with walk-forward)	9026.01	74.76	0.5025
Random Forest	9219.57	70.76	0.4918
Holt–Winters	10090.85	75.35	0.4438
LightGBM	10457.86	78.82	0.4236
XGBoost	11689.48	76.68	0.3557
Prophet	14820.56	91.29	0.1831
SARIMAX	21335.65	98.05	-0.1760
Naïve Seasonal	39897.17	173.84	-1.1191
Lasso (without walk-forward)	*3169.72*	*45.15*	-0.8100

Table 7: Predictive performance under walk-forward validation

The following sub-sections present a comparative error analysis and an evaluation of residuals and hold-out data, providing broader perspectives on model performance.

Comparative Error Analysis

The relative performance of models is further illustrated in Figure 7 (MSE), Figure 8 (MAE), and Figure 9 (R^2). These bar plots confirm that ARIMA's performance is the most balanced across all three metrics, combining relatively low errors with the highest R^2 value, followed closely by the naïve last-value predictor. The walk-forward Lasso model and Random Forest cluster in the mid-range, while Prophet, SARIMAX, and Naïve seasonal models underperform substantially.

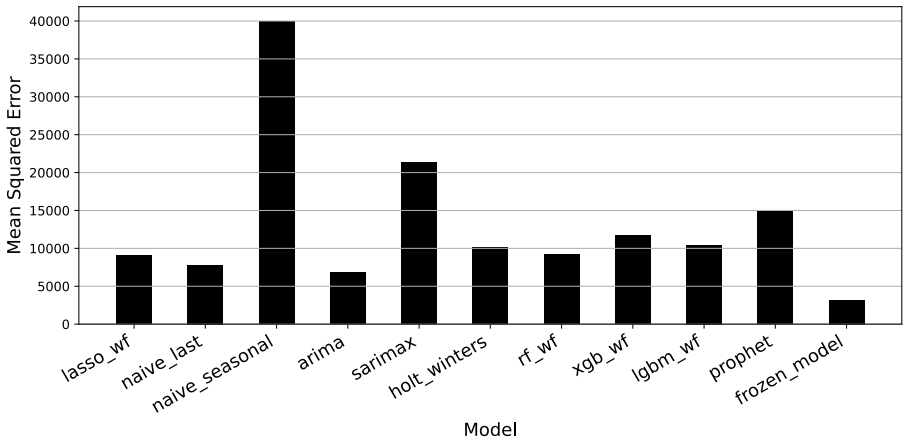

Figure 7: Walk-forward comparison using MSE (lower is better)

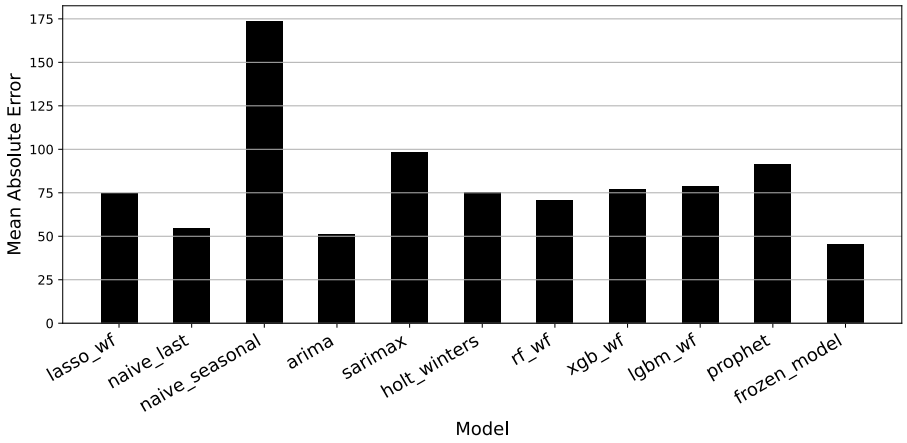

Figure 8: Walk-forward comparison using MAE (lower is better)

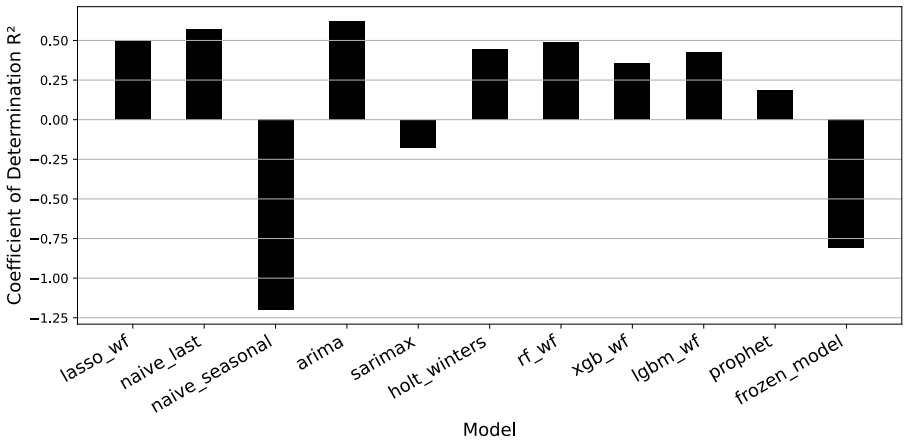

Figure 9: Walk-forward comparison using R² (higher is better)

Figure 10 presents a time-series plot comparing actual issue closures with model predictions. Both ARIMA and naïve last value closely track the observed fluctuations, whereas Lasso and ensemble models lag during periods of rapid change. Prophet, SARIMAX, and naïve seasonal diverge considerably, highlighting their limited applicability in this context.

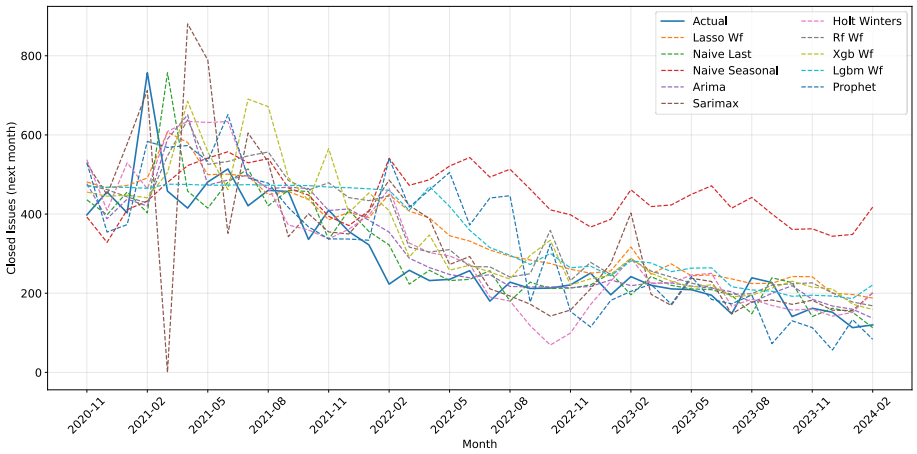

Figure 10: Walk-forward actuals vs. predictions

Residual and Hold-out Evaluation

Figure 11 depicts the residual distribution of the walk-forward Lasso and ARIMA models. The residuals show a negative skew, indicating systematic underestimation of closure volumes. By contrast, ARIMA residuals approximate a symmetric distribution, reflecting its stronger alignment with actual dynamics.

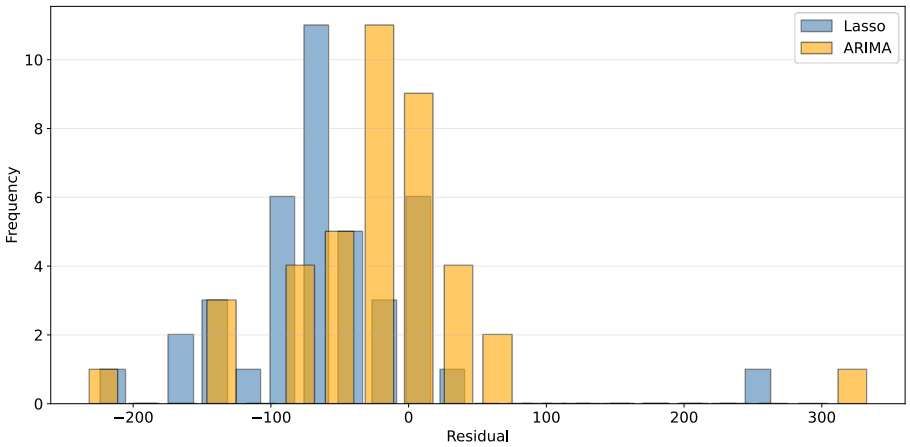

Figure 11: Walk-forward Lasso vs. ARIMA residual distribution

The frozen Lasso model, evaluated on the final 12 months (Figure 12), achieved deceptively low errors but yielded a negative R^2 (–0.81). This suggests that while the model effectively approximated the overall trend of issue closures, it may not have fully captured the underlying variability, possibly due to residual temporal dependencies in the data. Although a time-series–aware evaluation strategy was applied, future research could further refine the evaluation design to mitigate such effects and improve the modeling of temporal fluctuations.

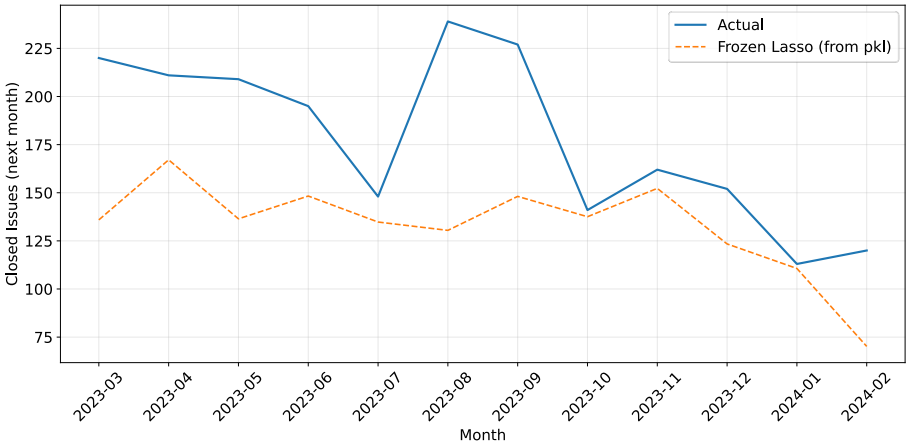

Figure 12: Frozen model on held-out last 12 months

4.4.6 Implications and Limitations of Final Model Validation

The expanded evaluation highlights important implications for both research and practice in software project management. While ARIMA achieved the strongest

predictive performance under walk-forward validation, the feature-driven Lasso framework remains competitive and provides interpretability advantages by linking issue closure rates to project signals such as contributor activity and labeling practices. This distinction is critical: ARIMA excels as a forecasting benchmark, whereas Lasso serves as an analytical instrument that can explain why changes occur, offering project managers actionable insights beyond raw prediction.

From a methodological perspective, the results emphasize the necessity of time-aware validation. The discrepancy between the optimistic frozen model and its poor held-out performance underscores the risks of temporal leakage, reinforcing the importance of rigorous evaluation design in software analytics research.

Practically, both accurate forecasts and interpretable drivers are essential for decision-making. Predictive models can inform sprint planning, resource allocation, and workload management, while feature-based analyses help managers understand how team dynamics and issue characteristics influence project outcomes. In this way, the integration of statistical baselines with interpretable machine learning strengthens the reliability and applicability of predictive modelling in software project management.

4.5 Application

The practical application of the predictive model in software project management encompasses critical areas, including trend analysis, resource forecasting, process efficiency, performance monitoring, sprint planning, quality assurance, and stakeholder management. By leveraging predictive insights, project managers can make informed decisions that enhance project outcomes and efficiency. The model presents two data sets: the actual number of closed issues over time and the predicted closures. For project managers, the ability to predict the volume of issue closures is crucial for reasons:

- Sprint Planning: Predictive analysis aids in effective sprint planning by balancing new features, bug fixes, and technical debt resolution. Research shows that predictions of sprint delivery capabilities can significantly enhance the planning and execution of iterative software development, leading to more efficient resource allocation and timing of deliverables [Hady and Kusumo, 2023].
    - Predicted < Actual Closures: This is positive, as it indicates the team is closing more issues than anticipated. The manager might harness this momentum to push for further quality initiatives or reallocate human resources to areas like feature development.
    - Predicted > Actual Closures: This outcome suggests caution. It could indicate emerging challenges within the team or the codebase, requiring deeper analysis and potentially corrective actions.
- Resource Forecasting: The predicted number of issue closures helps in forecasting the workload and planning the allocation of resources effectively. A high prediction for the upcoming months may indicate the need for temporary reinforcements or shifts in team assignments. For example, Gao et al. [Gao et al., 2023] highlight the importance of using predictive insights for resource allocation, which can significantly enhance project planning.
- Quality Assurance: Predictive models indicating a decrease in issue closures can signal the need for enhanced quality assurance initiatives. This preemptive

approach ensures that current issues are addressed promptly, maintaining exacting standards of software quality, and reducing the likelihood of future complications [Ramírez-Mora et al., 2020].

- Process Efficiency: The trend line of actual closures, when compared to predictions, can offer insights into the efficiency of the issue resolution process. A close match suggests a well-calibrated system, whereas consistent under- or over-estimation may prompt a review of the prediction model or the issue management process. Studies have developed interpretable frameworks for various forecasting domains that allow stakeholders to understand and adjust processes based on predictive insights, improving overall efficiency [Wang and Ye, 2020].
- Performance Monitoring: This analysis serves as a performance monitoring tool. By comparing actual outcomes to predictions, a project manager can gauge the team's productivity and identify potential areas for process improvement. Research shows that advanced data analysis techniques like machine learning improve the accuracy of forecasting, which can be analogous to monitoring project performance in software development [Sarangi, 2020].
- Stakeholder Management: Accurate predictions are crucial for effective stakeholder management, enabling clear communication regarding project progress and expected outcomes. This transparency helps manage expectations and supports decision-making processes, ensuring that all parties align with the project's goals and timelines [Brajer-Marczak et al., 2021].

By integrating predictive modeling insights into these areas, project managers can make data-driven decisions that improve the overall efficiency and success of software development projects. However, while predictive modeling holds significant promise, it also faces challenges such as data quality, model interpretability, and the dynamic nature of software projects [Malhotra and Lata, 2020]. These challenges necessitate ongoing research to refine predictive techniques and enhance their practical applicability in various project management contexts.

5 Conclusion

This research examined the application of predictive modeling in software project management , with TensorFlow serving as a large-scale open-source case study. By leveraging historical data from the TensorFlow GitHub repository, we designed, implemented, and evaluated predictive models aimed at forecasting issue closure dynamics and supporting project management decision-making. The key contributions and insights from this work are summarized below.

- Data Collection and Preprocessing: We systematically collected and cleaned data from the TensorFlow repository, incorporating commit history, issue tracking, and contributor activity. This structured dataset enabled both descriptive exploration and predictive modelling.
- Descriptive Analysis: Exploratory analysis revealed underlying development patterns, such as fluctuations in issue creation and resolution, contributor

involvement, and labeling practices, all of which informed subsequent feature engineering and model design.

- Predictive Modeling and Time-Aware Validation: A diverse set of models were evaluated, ranging from linear regression (Ridge, Lasso), machine learning ensembles (Random Forest, XGBoost, LightGBM), naïve heuristics to classical statistical forecasting methods (ARIMA, SARIMAX, Holt–Winters, and the proposed Lasso regression framework. Using expanding-window walk-forward validation to avoid temporal leakage, we found that ARIMA achieved the strongest predictive performance ($R^2 = 0.62$), followed closely by the naïve last-value baseline ($R^2 = 0.57$). The Lasso model achieved competitive results ($R^2 = 0.50$), comparable to Random Forest, and remained valuable for its interpretability in linking project features to issue closure outcomes. Prophet and SARIMAX underperformed, confirming the limited role of seasonal effects in this dataset.
- Application of Predictive Insights: The predictive frameworks can support multiple project management functions, including sprint planning, workload forecasting, process efficiency assessment, and performance monitoring. Importantly, the Lasso model's feature-driven structure provides explanatory insights into how factors such as contributor activity and labeling practices shape closure rates, offering managerial value beyond forecasting accuracy.

In this study demonstrates that predictive modelling can enhance software project management when evaluated with rigorous, time-aware methods. While ARIMA emerged as the strongest forecasting baseline, the Lasso regression framework retains value for interpretability and decision support. More broadly, the results highlight the need for transparent benchmarking and temporal validation to avoid overly optimistic conclusions and to ensure credible, reproducible findings in software analytics.

Future research could extend these methods to other large-scale repositories, integrate text mining or sentiment analysis of developer communications to capture team dynamics, and explore hybrid approaches that combine statistical forecasting with feature-driven machine learning to balance accuracy and interpretability.

References

[Balsamo et al., 2004] Balsamo, S., Marco, A., Inverardi, P., Simeoni, M.: "Model-based performance prediction in software development: a survey"; IEEE Transactions on Software Engineering, 30 (2004), 295-310. https://doi.org/10.1109/TSE.2004.9.

[Brajer-Marczak et al., 2021] Brajer-Marczak, R., Marciszewska, A., Nadolny, M.: "Selected Determinants of Stakeholder Influence on Project Management in Non-Profit Organizations"; Sustainability, 13, 16 (2021). https://doi.org/10.3390/su13168899.

[Chen et al., 2020] Chen, Q., Xue, B., Zhang, M.: "Improving symbolic regression based on correlation between residuals and variables"; Proc. 2020 Genetic and Evolutionary Computation Conference, 922-930 (2020). https://doi.org/10.1145/3377930.3390161.

[Choetkiertikul et al., 2017] Choetkiertikul, M., Dam, K., Tran, T., Ghose, A.: "Predicting the delay of issues with due dates in software projects." Empirical Software Engineering, 22 (2017), 1223-1263. https://doi.org/10.1007/s10664-016-9496-7.

[Christenson Jr. and Goldstein, 2022] Christenson Jr., A. P., Goldstein, W. S.: "Impact of data analytics in transforming the decision-making process"; Business & IT, Vol. XII (1), (2022), 74-82. https://doi.org/10.14311/bit.2022.01.09.

[Dinh et al., 2025] Dinh, D. H., Tran, Q. N. M., Dong, Q. H., & Ondrusch, N.: "TensorFlow Predictive Project Management [Source code]" (2025). GitHub. https://github.com/quantranvn/tf-predictive-modeling-project.

[Elish et al., 2015] Elish, M., Aljamaan, H., Ahmad, I.: "Three empirical studies on predicting software maintainability using ensemble methods"; Soft Computing, 19 (2015), 2511-2524. https://doi.org/10.1007/s00500-014-1576-2.

[Emanuel et al., 2011] Emanuel, A., Wardoyo, R., Istiyanto, J., Mustofa, K.: "Statistical Analysis on Software Metrics Affecting Modularity in Open Source Software." International Journal of Computer Science and Information Technology, 3 (2011), 105-118. https://doi.org/10.5121/IJCSIT.2011.3308.

[Emanuel et al., 2013] Emanuel, A., Wardoyo, R., Istiyanto, J., Mustofa, K.: "Modularity Index Metrics for Java-Based Open Source Software Projects"; ArXiv abs/1309.5689 (2013). https://doi.org/10.14569/IJACSA.2011.021109.

[Feng et al., 2020] Feng, C., Li, L., Sadeghpour, A.: "A comparison of residual diagnosis tools for diagnosing regression models for count data"; BMC Medical Research Methodology, 20 (2020). https://doi.org/10.1186/s12874-020-01055-2.

[Ferreira et al., 2022] Ferreira, M., Gonçalves, D., Bigonha, M., Ferreira, K.: "Characterizing Commits in Open-Source Software"; Proc. XXI Brazilian Symposium on Software Quality, 1-10 (2022). https://doi.org/10.1145/3571473.3571508.

[Foucault et al., 2015] Foucault, M., Teyton, C., Lo, D., Blanc, X., Falleri, J.: "On the usefulness of ownership metrics in open-source software projects"; Information and Software Technology, 64 (2015), 102-112. https://doi.org/10.1016/j.infsof.2015.01.013.

[Gao et al., 2023] Gao, Y., Huang, X., Zhou, X., Gao, X., Li, G., Chen, G.: "DBAugur: An Adversarial-based Trend Forecasting System for Diversified Workloads"; Proc. 2023 IEEE 39th International Conference on Data Engineering (ICDE), 27-39 (2023). https://doi.org/10.1109/ICDE55515.2023.00385.

[Hady and Kusumo, 2023] Hady, C., Kusumo, D.: "Prediction of a Sprint Delivery's Capabilities in Iterative-based Software Development"; JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika), 8, 1 (2023), 77-83. https://doi.org/10.29100/jipi.v8i1.3292.

[Hao and Zhang, 2011] Hao, Y., Zhang, Y.: "Statistical prediction modeling for software development process performance"; 2011 IEEE 3rd International Conference on Communication Software and Networks, 703-706 (2011). https://doi.org/10.1109/ICCSN.2011.6014187.

[Haris et al., 2023] Haris, M., Chua, F., Lim, A.: "An Ensemble-Based Framework to Estimate Software Project Effort"; Proc. 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), 47-52 (2023). https://doi.org/10.1109/ICSECS58457.2023.10256337.

[Hata et al., 2021] Hata, H., Novielli, N., Baltes, S., Kula, R. G., Treude, C.: "GitHub Discussions: An Exploratory Study of Early Adoption"; arXiv (Cornell University) (2021). https://doi.org/10.48550/arxiv.2102.05230.

[Hosni et al., 2021] Hosni, M., Idri, A., Abran, A.: "On the value of filter feature selection techniques in homogeneous ensembles effort estimation"; Journal of Software: Evolution and Process, 33 (2021), e2343. https://doi.org/10.1002/smr.2343.

[Idri et al., 2016] Idri, A., Hosni, M., Abran, A.: "Improved estimation of software development effort using Classical and Fuzzy Analogy ensembles"; Applied Soft Computing, 49 (2016), 990-1019. https://doi.org/10.1016/j.asoc.2016.08.012.

[Jahanshahi et al., 2020] Jahanshahi, H., Cevik, M., Navas-Sú, J., Basar, A., Torres, A.: "Wayback Machine: Capturing the evolutionary behaviour of the bug dependency graph in open-source software systems." ArXiv abs/2011.05382 (2020).

[Khurana et al., 2017] Khurana, U., Samulowitz, H., Turaga, D.: "Feature Engineering for Predictive Modeling using Reinforcement Learning"; ArXiv abs/1709.07150 (2017). https://doi.org/10.1609/aaai.v32i1.11678.

[Kim et al., 2016] Kim, J., Kim, E., Kim, J.: "Towards Conceptual Predictive Modeling for Big Data Framework"; International Journal of Software Engineering and its Applications, 10 (2016), 35-42. https://doi.org/10.14257/IJSEIA.2016.10.1.04.

[Koch and Neumann, 2008] Koch, S., Neumann, C.: "Exploring the Effects of Process Characteristics on Products Quality in Open Source Software Development"; Journal of Database Management, 19 (2008), 132-159. https://doi.org/10.4018/JDM.2008040102.

[Kolassa et al., 2013] Kolassa, C., Riehle, D., Salim, M.: "The empirical commit frequency distribution of open source projects"; Proc. 9th International Symposium on Open Collaboration, 1-8 (2013). https://doi.org/10.1145/2491055.2491073.

[Krause et al., 2014] Krause, J., Perer, A., Bertini, E.: "INFUSE: Interactive Feature Selection for Predictive Modeling of High Dimensional Data"; IEEE Transactions on Visualization and Computer Graphics, 20 (2014), 1614-1623. https://doi.org/10.1109/TVCG.2014.2346482.

[Liu et al., 2018] Liu, Y., Bai, D., Jiao, W.: "Generating Adaptation Rules of Software Systems: A Method Based on Genetic Algorithm"; Proc. 10th International Conference on Machine Learning and Computing, 347-356 (2018). https://doi.org/10.1145/3195106.3195137.

[Ma et al., 2013] Ma, Y., Wu, Y., Xu, Y.: "Dynamics of open-source software developer's commit behavior: an empirical investigation of subversion"; Proc. 29th Annual ACM Symposium on Applied Computing, 1171-1173 (2013). https://doi.org/10.1145/2554850.2555079.

[Malhotra and Lata, 2020] Malhotra, R., Lata, K.: "An empirical study on predictability of software maintainability using imbalanced data"; Software Quality Journal, 28 (2020), 1581-1614. https://doi.org/10.1007/s11219-020-09525-y.

[Malhotra, 2017] Malhotra, R.: "Software Quality Predictive Modeling: An Effective Assessment of Experimental Data"; Proc. 10th Innovations in Software Engineering Conference, 215-216 (2017). https://doi.org/10.1145/3021460.3021491.

[Nargesian et al., 2017] Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E., Turaga, D.: "Learning Feature Engineering for Classification"; Proc. IJCAI, 2529-2535 (2017). https://doi.org/10.24963/ijcai.2017/352.

[Ndenga et al., 2015] Ndenga, M., Jean, M., Ganchev, I., Franklin, W., Kimathi, D.: "Assessing Quality of Open Source Software Based on Community Metrics"; International Journal of Software Engineering and its Applications, 9 (2015), 337-348. https://doi.org/10.14257/IJSEIA.2015.9.12.30.

[Nesamani and Rajini, 2021] Nesamani, S., Rajini, S.: "Predictive Modeling for Classification of Breast Cancer Dataset Using Feature Selection Techniques"; in: Handbook of Research on Innovations and Applications of AI, IoT, and Cognitive Technologies (2021), 12. https://doi.org/10.4018/978-1-7998-6870-5.ch015.

[Norick et al., 2010] Norick, B., Krohn, J., Howard, E., Welna, B., Izurieta, C.: "Effects of the number of developers on code quality in open source software: a case study."; Proc. 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 1-1 (2010). https://doi.org/10.1145/1852786.1852864.

[Oyamada, 2019] Oyamada, M.: "Extracting Feature Engineering Knowledge from Data Science Notebooks"; Proc. 2019 IEEE International Conference on Big Data (Big Data), 6172-6173 (2019). https://doi.org/10.1109/BigData47090.2019.9006522.

[Paulson et al., 2004] Paulson, J., Succi, G., Eberlein, A.: "An empirical study of open-source and closed-source software products"; IEEE Transactions on Software Engineering, 30 (2004), 246-256. https://doi.org/10.1109/TSE.2004.1274044.

[Rai et al., 2021] Rai, P., Kumar, S., Verma, D.: "Prediction of Software Effort in the Early Stage of Software Development: A Hybrid Model"; IEEE Canadian Journal of Electrical and Computer Engineering, 44 (2021), 376-383. https://doi.org/10.1109/ICJECE.2021.3084850.

[Ramírez-Mora et al., 2020] Ramírez-Mora, S. L., Oktaba, H., Gómez-Adorno, H.: "Descriptions of issues and comments for predicting issue success in software projects"; Journal of Systems and Software, 168 (2020), 110663. https://doi.org/10.1016/j.jss.2020.110663.

[Rokach, 2010] Rokach, L.: "Ensemble-based classifiers"; Artificial Intelligence Review, 33 (2010), 1-39. https://doi.org/10.1007/s10462-009-9124-7.

[Rudy, 2012] Rudy, J.: "Runtime software adaptation: approaches and a programming tool"; Applied Computer Science, 6 (2012), 75-89.

[Saini et al., 2022] Saini, M., Arora, R., Adebayo, S.: "In-Depth Analysis and Prediction of Coupling Metrics of Open Source Software Projects"; Journal of Information Technology Research, 15 (2022), 1-16. https://doi.org/10.4018/jitr.301267.

[Sarangi, 2020] Sarangi, M.: "A Literature Review on Machine Learning Applications in Financial Forecasting"; Journal of Technology Management for Growing Economies, 11, 1 (2020), 23-27. https://doi.org/10.15415/jtmge.2020.111004.

[Schützko and Timinger, 2023] Schützko, F., Timinger, H.: "Predictive analytics for project management"; Proc. 2023 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC), 1-8 (2023). https://doi.org/10.1109/ice/itmc58018.2023.10332400.

[Seker et al., 2021] Seker, A., Diri, B., Arslan, H.: "New Developer Metrics for Open Source Software Development Challenges: An Empirical Study of Project Recommendation Systems." Applied Sciences, 11, 3 (2021), 920. https://doi.org/10.3390/APP11030920.

[Shmueli et al., 2016] Shmueli, G., Ray, S., Estrada, J., Chatla, S.: "The elephant in the room: Predictive performance of PLS models"; Journal of Business Research, 69 (2016), 4552-4564. https://doi.org/10.1016/J.JBUSRES.2016.03.049.

[Sunindyo et al., 2012] Sunindyo, W., Moser, T., Winkler, D., Dhungana, D.: "Improving Open Source Software Process Quality Based on Defect Data Mining." (2012), 84-102. https://doi.org/10.1007/978-3-642-27213-4_7.

[TensorFlow Developers, 2015] TensorFlow Developers: "TensorFlow [Computer software]" (2015). GitHub. https://github.com/tensorflow/tensorflow.

[Toma and Wei, 2023] Toma, M., Wei, O.: "Predictive Modeling in Medicine"; Encyclopedia, 3, 2 (2023), 590-601. https://doi.org/10.3390/encyclopedia3020042.

[Tomas et al., 2013] Tomas, P., Cuaresma, M., Risoto, M.: "Open source tools for measuring the Internal Quality of Java software products. A survey." Computer Standards & Interfaces, 36 (2013), 244-255. https://doi.org/10.1016/J.CSI.2013.08.006.

[Tosun et al., 2020] Tosun, A., McIntosh, S., Minku, L., Turhan, B.: "Guest Editorial: Special Issue on Predictive Models and Data Analytics in Software Engineering"; Empirical Software Engineering, 25 (2020), 1533-1535. https://doi.org/10.1007/s10664-020-09811-0.

[Tran et al., 2014] Tran, T., Luo, W., Phung, D., Gupta, S., Rana, S., Kennedy, R., Larkins, A., Venkatesh, S.: "A framework for feature extraction from hospital medical data with applications in risk prediction"; BMC Bioinformatics, 15 (2014), 1-9. https://doi.org/10.1186/s12859-014-0425-8.

[Wang and Ye, 2020] Wang, L., Ye, Z.: "An Interpretable Framework for Stock Trend Forecasting"; Journal of Physics: Conference Series, 1634 (2020), 012026. https://doi.org/10.1088/1742-6596/1634/1/012026.

[Wang et al., 2020] Wang, Y., Cao, J., Lo, D.: "Mining and Predicting Micro-Process Patterns of Issue Resolution for Open Source Software Projects." (2020), 477-482. https://doi.org/10.18293/SEKE2020-031.

[Xiao et al., 2023] Xiao, T., Baltes, S., Hata, H., Treude, C., Kula, R. G., Ishio, T., Matsumoto, K.: "18 million links in commit messages: purpose, evolution, and decay"; Empirical Software Engineering, 28, 4 (2023), 91. https://doi.org/10.1007/s10664-023-10325-8.

[Yang et al., 2020] Yang, Y., Xia, X., Lo, D., Bi, T., Grundy, J., Yang, X.: "Predictive Models in Software Engineering: Challenges and Opportunities"; ACM Transactions on Software Engineering and Methodology (TOSEM), 31, 1 (2020), 1-72. https://doi.org/10.1145/3503509.

[Zhang and Yang, 2017] Zhang, T., Yang, B.: "An exact approach to ridge regression for big data"; Computational Statistics, 32 (2017), 909-928. https://doi.org/10.1007/S00180-017-0731-5.

[Zhu et al., 2013] Zhu, Y., Zhong, E., Lu, Z., Yang, Q.: "Feature engineering for semantic place prediction"; Pervasive and Mobile Computing, 9 (2013), 772-783. https://doi.org/10.1016/j.pmcj.2013.07.004.