Exploring Pull Requests and Project Sustainability In Apache Incubator Projects

Raiyan Jahangir (922940219), Humaira Fasih Ahmed (924170595), Jun Wu (919841826), Zane Reis (924176625), Ahnaf Faisal (924160657)

1 Introduction

Software projects often involve distributed collaboration [1]. To avoid conflicts, developers make their own version of the project and work independently [2]. When they are done, they create a Pull Request (PR) to request merging their code with the main codebase [3]. These PRs are reviewed by the project leads and maintainers, and if deemed satisfactory, are merged into the main codebase [4]. In addition, maintainers and other developers can comment on the PR to discuss potential improvements and clarify details. As such, the entire PR submission, review, and commenting process is crucial; it helps regulate the quality of the proposed changes and ensures contributions are integrated smoothly into the main codebase. However, it is not yet clear how PR acceptance/rejection rate, PRs by core developers, communication latency, and other PR factors influence a project's journey to sustainability.

In our study, we evaluate the sustainability of projects incubated at the Apache Software Foundation (ASF) by examining their pull request process. The ASF Incubator (ASFI) helps early-stage open-source software (OSS) projects by providing mentorship and structure, categorizing them as graduated (sustainable) or retired (unsustainable) depending on their progress [5]. Specifically, our goal for this work is to empirically analyze how PR-based metrics, such as the number and status of PRs, count of active/core developers raising PRs, and pattern of communication in PR comments relate to project sustainability. For core developers, we try to understand how the characteristics of their PRs differ from non-core developers' PRs. We also explore the extent to which Deep Learning models can be used to predict project sustainability using only PR-based features.

Our work draws on structural contingency theory — which says that no single organizational structure fits all contexts and that effective performance depends on a mix of factors such as people, strategy, and culture [6, 7]. Thus, the way PR-based processes operate are dependent on each project's unique context. In ASFI projects, factors like communication patterns, review practices, and code input from core and non-core developers all interact to influence sustainability.

2 Background

2.1 Literature Review

Several studies have examined the sustainability of OSS projects within the ASFI based on some factors. Stănciulescu et al. [7] showed that sustainable projects tend to attract more contributors, have higher commit activity, and maintain robust testing practices, while unsustainable projects have low engagement and contributor retention. Yin et al. [6] implemented an LSTM-based model, which achieved high accuracy in forecasting OSS sustainability based on socio-technical factors such as governance, email communication, and code quality. The model has associated sustainable projects with small groups of centralized committers and large groups of distributed communicators. Yin et al. [8] explored how the ASFI projects are influenced by self-governance and sudden changes. The study also showed that sustainable projects demonstrate efficient feedback between governance and technical structures, while unsustainable projects do not. However, these studies primarily focused on important but broad project metrics, such as the number of contributors, source lines of code (SLOC), commit type/activity, code quality, or emails, rather than the **specific role of pull request dynamics** in OSS sustainability.

While we recognize that the study by Yin et al. [6] involves analysis of developer communications as part of social networks modeled from the ASFI mailing lists, the authors mainly consider email communi-

cations, whereas our work explores communication trends in **PR comments** and how that communication relates to project sustainability. Furthermore, Dhasmana et al. [9] have found there is no relationship between sustainable projects and the number of duplicate pull requests, pull request frequency, and commit size, but they did not explore the **core developer involvement in pull requests**, or pre-merge **discussions in comments**. We believe these factors could help understand how developer type affects the PR acceptance and review process, and if PR discussions are connected to project sustainability.

2.2 Research Questions

The specific research questions we aim to explore are:

- How do pull request activities differ between sustainable and unsustainable ASFI projects? How do PRs by core developers differ in sustainable vs. unsustainable projects?
- Can we predict sustainability based on PR-related features? How does each feature impact the model outcome?

3 Methodology

3.1 Data Source

We choose a publicly available ASFI dataset prepared by Yin et al. [5]. The dataset is structured across several files which keep track of github projects, commits made to dataset, list of developers and contributors and mailing lists. This dataset is uniquely suited to our study because it provides extrinsic labeling for sustainability classifying projects as either "graduated" , "retired", or "in_incubation". There were a total of 312 projects among which 204 were graduated, 60 were retired and 48 were in incubation.

3.2 Data Processing

We initially hoped that this ASFI dataset contained structured data for PRs, but we found that this was not the case. We considered processing the mailing list data to capture PRs, but we determined that collecting the data directly from GitHub's API would be more robust. To facilitate this, we developed a pipeline to automatically scrape data from GitHub, remove irrelevant fields, and add additional fields to enable easy integration with the ASFI dataset. Fortunately, GitHub labels users who are bots, allowing us to easily exclude PRs and comments from these users. Developing this pipeline revealed a flaw in the ASFI dataset: the GitHub URLs for 102 out of the 312 were incorrect, each returning a 404 error. A majority of these projects had been retired. Given that retired projects represented a majority of our data, this significantly impacted our ability to develop an accurate model for predicting project sustainability. To address this, we manually verified the 102 erroneous entries, identifying valid URLs whenever possible. As a result of this effort, we reduced the number of incorrect URLs from 102 to 43. These 43 remaining projects did not use GitHub and exclusively used Apache Subversion for source version control. Overall, we ended up with 180 graduated and 26 retired projects that we successfully obtained pull request data from.
For our model, we aggregated data by month for each project. PRs and comments from bots were removed. The selected features are as follows:

listid is the project ID as found in the ASFI dataset. This field is identical to the one in lists_2019_8.csv. This field was kept to allow for easy integration with the ASFI dataset.

repo is the name of the GitHub repo as it appears in the project's GitHub URL. This is sometimes different from the project's name. Use "listid" if you wish to use this data with the ASFI dataset.

status This is the project's status as indicated in the ASFI dataset. This field is populated using data from lists_2019_8.csv.

month contains a year and a month. This is the year and month the following fields were calculated for.

avg_response_time is the average time between comments on a PR per month for a project.

avg_first_response_time is the average time it takes for a PR to receive its first comment per month for a project.

active_devs is the number of developers who have either created a PR or commented per month for a project.

total_active_devs is the total number of developers who have either created a PR or commented across all months for a project.

accepted_prs is the number of previously open PRs that were merged into the repository per month for a project.

avg_time_to_acceptance is the average time for a PR to be merged per month for a project.

rejected_prs is the number of previously open PRs that were closed without merging per month for a project.

avg_time_to_rejection is the time for a PR to be closed without merging per month for a project.

unresolved_prs is the number of open PRs that were neither accepted nor rejected by the end of the month per month for a project.

avg_thread_length is the average number of comments per open PR per month for a project. PRs that were open at any time during this month were counted.

new_prs is the number of newly created PRs per month for a project.

new_comments is the number of newly created comments per month for a project.

3.3 RQ1: PR Activity Analysis

Ratio of Merged to Closed PRs: We calculate this ratio for graduated and retired projects by dividing the number of merged PRs by the number of closed PRs. Using GitHub's API, we identify PR states through the ***state*** field and the ***merged_at*** timestamp, thereby distinguishing accepted/merged, open/unresolved, and rejected/closed PRs. We used violin and scatter plots to visualize the significance of this metric to project sustainability.

Communication Latency in PR Discussions: We measure latency using three sub-metrics: first response time (from PR creation to first comment), PR resolution time (from creation to closure), and average response time (time between comments). For each, we compute the mean and median values—using the median to account for skewed distributions—and employ box plots to display medians, quartiles, and outliers. These plots help show whether graduated projects respond faster than retired ones.

Average No. of Accepted/Closed/Unresolved PRs per Project: Using the same method as the merge-to-closed PR ratio, we measure accepted, closed and unresolved PRs per project. Since these metrics can differ heavily due to project size, we normalize them by **total_active_developers**. We used box- and violin-plots to find the potential relation of these metrics to project sustainability.

Core Developers: The second part of RQ1 aims to analyze the relationship between PRs made by core developers in sustainable vs. unsustainable projects. A **core developer** is defined as someone who has **more commits than the average commits** in a particular project and is **part of k people whose sum of commits comprise 90% of the total commits made in the project.** For this, we use the *project_id*, *project_name*, and *project_status* fields from the **lists** table, the commits data (who did the commit and in which project) from the **commits** table, and the developer data from *alias_id* and *name* fields from **alias** table. We map and transform the data to get total no. of people and commits per project, from which we compute the total no. of commits per person per project. From these, we find average commits per project. Then based on the definition of core developer, we determine the first `k` people with the most commits as core developers.

Mapping Core Developers' PRs to Project Sustainability: For this, we use the PR data scraped

with Github API. The PR data has 252 projects from among 312 in the ASFI dataset. Among 252, 180 are "graduated", 26 are "retired", and 46 are "in_incubation". We remove all projects that are "in_incubation" for our analysis. From the rest, we consider "graduated" as "Sustainable" and "retired" as "Unsustainable". We do not consider cases where the PR *status* was `'Open'`, since they are unresolved. We map the project names in PR data to ASFI (since PR data uses direct Github repo names). Finally, we get all the necessary information in a single table to analyze core developer PR characteristics. We got a total of 369934 PRs from 206 projects. We analyse relevant metrics to determine the extent to which core developer PRs differ between sustainable and unsustainable projects in Section 4 (Results).

3.4 RQ2: PR-based Project Sustainability Prediction

For RQ2, we trained 4 recurrent neural networks (RNN) [10] namely LSTM [11], Bi-LSTM [12], GRU [13], and Bi-GRU [14]. We chose these models because RNNs are well-suited for time-series data, and predicting project sustainability using PRs inherently requires longitudinal data. Furthermore, we structured our PR data over monthly intervals, which justifies the use of the aforementioned models more.

3.4.1 Data Preprocessing

Time-Series Structuring: Each project's PR activity is aggregated over monthly intervals using features described in Section 3.2. The dataset is sorted by project and month to ensure that the model receives the monthly data for each project in order.
Target Label: The data for each month is treated as a separate sequence with a target label - 1 for graduated and 0 for retired projects.
Normalization: We normalized the features using the field *total_active_devs* to account for the difference in project sizes. We removed the field *total_active_devs* after normalization, to prevent it from introducing any collinearity as it is already used for feature engineering. After that, we used MinMaxScaler to standardize all numerical data within a specific range. This prevents very large values from dominating the model's prediction and allows the model to consider all features equally. It also allows model to converge faster and avoid exploding gradients.
Outlier Removal: We tried two methods for outlier removal: Interquartile filtering (using the 25th and 75th percentile) to remove the lowest 25% and the highest 25% of the data. We also tried the 95th percentile. However, the test accuracy showed that the models yielded better accuracy (around 10% higher) without this step, so we decided to omit this and just rely on MinMaxScaling.
Train-Test Split: We ensured that our train-test split had equal distribution of graduated and retired projects to avoid incorrect results.
Class-weights: Since we have class imbalance, we compute class weights to avoid the model being biased towards "graduated" projects which are higher in number in our dataset.
Zero Padding: We zero-pad our data using the highest number of months observed in a project (across all projects), since different projects graduate/retire at different times from the incubator, which results in unequal monthly sequences. Even though this would cause variation in the models according to Yin et al.[6], we use this method due to time constraints of feeding one training sample at a time.[6]

3.4.2 Feature Selection/Importance

To determine which PR metrics contribute to project sustainability and are linearly independent, we used Lasso Regression [15]. Lasso assigns a weight/coefficient and a sign (+/-) to each of the PR features. If the feature does not contribute much to the project's sustainability, then it shrinks the weight of the feature toward zero. We tested using both RobustScaler and MinMaxScaler with Lasso. While RobustScaler kept more of our data trends since the outliers have some significance for our monthly data, our models learned better with MinMaxScaler. We did not use StandardScaler since none of our variables were normally distributed – which was tested using the built-in **Shapiro-Wilk** test in scikit-learn.

3.4.3 Model architecture

We determined the model architecture shown in Fig. 1 after tuning the hyperparameters several times. Note that we use the same architecture for all 4 models due to time constraints.

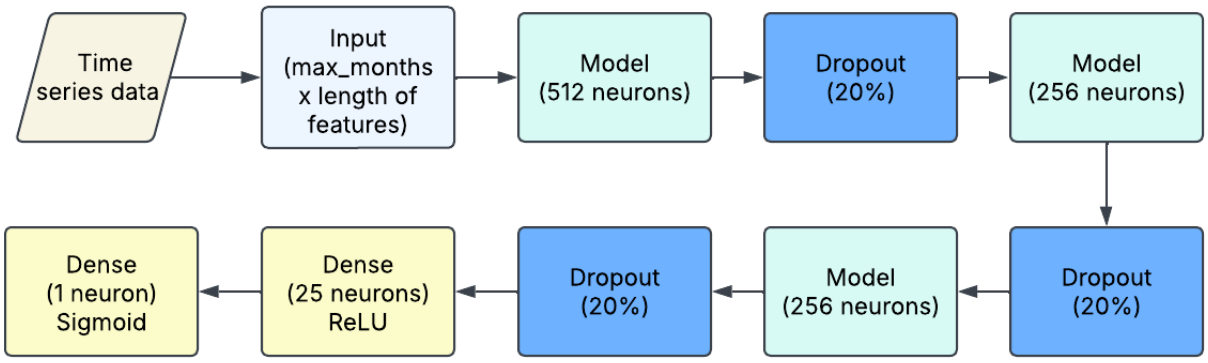


Figure 1: The architecture of the models

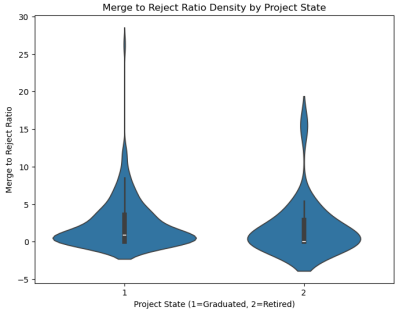
Input Layer: The first layer in the architecture is the input layer. Each model accepts one monthly time step (a sequence) of PR-related features.
Model Layer: This layer contains the RNN layer. RNNs learn long-term dependencies in time-series and sequential data and then pass the data to the next layer. We keep a total of 3 model layers in the architecture (e.g. 3 BiLSTM layers for BiLSTM model).
Dropout Layer: Dropout layer drops some neurons randomly in order to avoid overfitting. We use a dropout of 0.2, which means 20% of the neurons are dropped when the data passes through this layer. We have a dropout layer after every model layer to drop neurons.
Dense Layer: We have a dense layer after the sequence of model and dropout layers. Dense layers are fully connected layers that connect every neurons of previous layer to the next, carry out linear transformations, and apply activation functions to learn complex data. We have 2 dense layers in our architecture. The final dense layer is the output layer. It outputs the model's prediction (0 or 1) for the input monthly time step.
Hyperparameters: We used ReLu used as the activation function for the model. The activation function adds non-linearity to the model and allows it to learn complex patterns of the data, without the activation, LSTM will behave like a linear regression model no matter how many layers we add. ReLu is used because it does not saturate when the input is positive (avoid the vanishing gradients problem), and is fast to compute. Other hyperparameters are mentioned below:

Activation function: ReLU
Optimizer: Adam
Learning rate: 0.0005
Loss function: Binary cross-entropy
Stopping criteria: Early stopping
Batch Size: 3

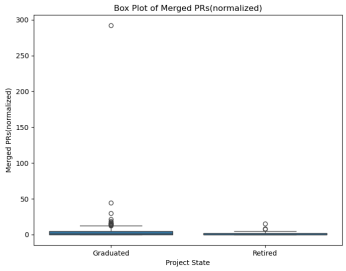
Evaluation: We run the each model to get the overall accuracy, precision, recall and F1-Score. We run each model 4 times and take the average to ensure consistency across our results.

Project Type	Total PRs	Merged PRs	Closed PRs	Unresolved PRs
Graduated (#180)	374500	195431	170864	8205
Retired (#26)	3729	2563	1076	90

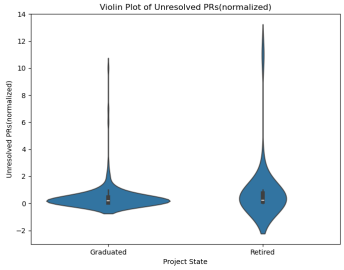
Table 1: Summary of Pull Request Samples



(a) Violin plot of PR acceptance to reject ratio



(b) Box plot of Accepted PRs (normalized)



(c) Violin plot of Unresolved PRs (normalized)

Figure 2: Plots on PR acceptance

3.4.4 SHAP Analysis

To understand how each PR-related feature affects the models' predictions, we used SHAP (Shapley Additive Explanations) analysis to interpret the predictions. SHAP calculates the contribution of each feature *(SHAP value)* to the prediction by adding or removing features one by one to identify how the prediction of the model changes. If a feature affects the predictions more compared to others, then it is more important. Negative SHAP values imply that a feature decreases the chance of the project to graduate, and vice versa. SHAP values close to 0 imply the feature has little influence. We visualize SHAP values with summary plots to show how each feature impacts the model.

4 Results

4.1 RQ1

Ratio of Merged to Closed PRs

We found that the mean of the merge-to-reject ratio is 2.34 for the graduated projects and 1.77 for retired ones. This means over a large sample of PRs in graduated projects, the merge rate is higher. This could indicate correlation between this metric and project sustainability. This can also be seen in 2a where graduated projects tend to have more merge acceptance for PRs, although the same could be said for some of the retired projects (which may be exceptions since retired project data is less).

Average Number of Accepted/Closed/Unresolved PRs per Project

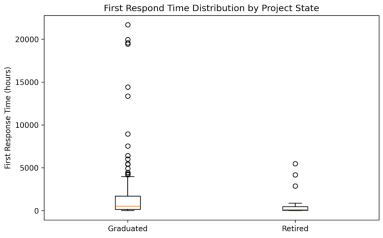
We found that mean of accepted PRs after normalization (by total_active_developers) for graduated projects is ≈ 5.54 and for retired is ≈ 1.78 . This is expected, the pattern is the same for closed ones where the values are ≈ 3.09 and ≈ 1.74 respectively. Interestingly, the same cannot be said for unresolved prs where the values are ≈ 0.45 and ≈ 0.85 respectively. We can see a couple of plots for this metric in 2. This 2b plot shows the higher amount of accepted PRs normalized by active developers in the graduated projects while the 2c plot shows relative high presence of unresolved PRs in the retired projects.

Project State	PR First Response Time (hrs)		PR Resolution Time (hrs)		Average Response Time (hrs)	
	Mean	Median	Mean	Median	Mean	Median
Graduated	1739.8	16.4	912.7	47.1	1361.8	4.4
Retired	773.7	18.2	426.9	10.2	1565.8	10.9

Table 2: Comparison of Communication latency for Graduated and Retired Projects

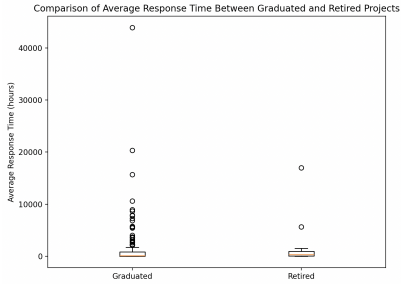
Communication latency in PR

Fig.3 shows PR first response time, resolution time, and average response time distribution by project state. We can observe that graduated projects have a higher median and wider quartiles for first response time compared to retired projects. For resolution time, graduated projects have a slightly lower median and thinner quartiles. For the average response time, the quartiles are similar but graduated projects have a slightly lower median. Outliers are present in all cases.



(a) Box plot on first respond time

(b) Box plot on PR resolution time



(c) Box plot on Average Response time

Figure 3: Data analysis on PR response time

PR Acceptance/Rejection rate among Core Developers

All analysis related to core developers is given in the left sub-table of table 3. It can be seen that 56.44% of the PRs by core developers have been merged and 43.56% have been closed. This shows that PRs initiated by core developers are more likely to be accepted into the project than be rejected/closed, although by a small margin.

Average time to PR Acceptance/Rejection for Core Developers

Average time for PR acceptance/ rejection for Core Developers is found to be 12.45 days, whereas the average time to PR acceptance/rejection is about 37.4 days (middle sub-table of table 3) across all developers. Thus, we hypothesize that the PRs initiated by core developers are given more preference and they are likely to be merged or closed sooner. This is also supported by the right sub-table of table 3 shows that the average time of PR acceptance/rejection for core developers for both sustainable (12.64 days) and unsustainable projects (2.79 days) is less than that of average time (37.60 days for sustainable and 17.47 days for unsustainable projects) across all developers. This again proves that the PRs made by core developers are given more preference and they are merged or closed earlier.

PRs by Core Developers		Avg time to PR acceptance/rejection		Avg time to PR acceptance/rejection in sustainable and unsustainable projects		
					Sustainable	**Unsustainable**
Merged	56.4%	Core Developers	12.45 days	Core Developers	12.64 days	17.47 days
Closed	43.6%	All Developers	37.4 days	All Developers	37.6 days	2.79 days

Table 3: Analysis on the relation between core developers and pull requests

Model	Test Accuracy	Precision (Graduated)	Precision (Retired)	Precision (Overall)	Recall (Graduated)	Recall (Retired)	Recall (Overall)	F1-Score (Graduated)	F1-Score (Retired)	F1-Score (Overall)
LSTM	0.88	0.98	0.55	0.92	0.89	0.88	0.88	0.93	0.67	0.89
BiLSTM	0.71	1.00	0.42	0.92	0.67	1.00	0.71	0.78	0.56	0.75
GRU	0.86	0.87	0.00	0.75	1.00	0.00	0.86	0.93	0.00	0.81
BiGRU	0.74	1.00	0.36	0.91	0.70	1.00	0.74	0.82	0.52	0.77

Table 4: Overall and per-class performances of models

4.2 RQ2

Feature importance

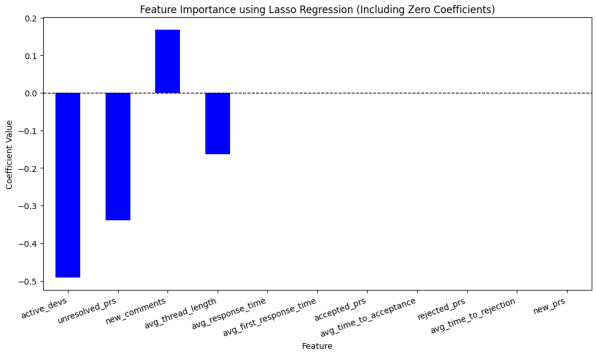
As discussed in Section 3.4.2, we determine the importance of our PR features using lasso regression. We can see from the relevant plot in figure 4a that *unresolved_prs* has a high negative coefficient value which means large number of ***unresolved_prs*** might have a significant impact on a project being retired. Surprisingly, ***active_devs*** shows the same behaviour which is kind of unexpected. On the other hand, feature like ***new_comments*** positively impact the graduation chances according to this analysis. Other notable features like ***accepted_prs*** and ***average_time_to_acceptance*** have coefficients close to zero which indicates that they do not have strong impact on a project's sustainability. In summary, this analysis provided us insights to chose relevant features from our PR data.

Model performance

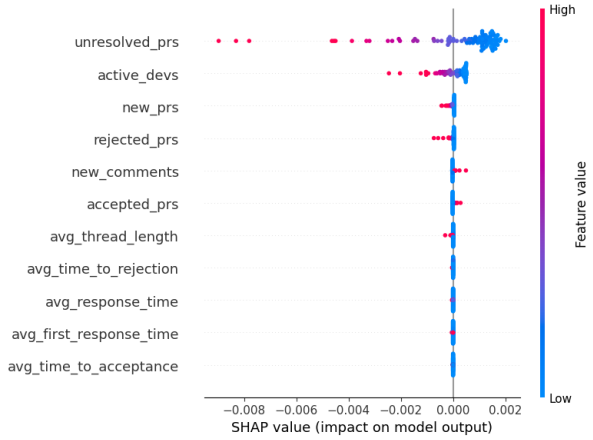
The overall accuracy, precision, recall, and f1-score of the 4 models are shown in table 4. The precision, recall, and f1-score per class are also shown. It is noticeable that LSTM is the best-performing among all the models with an overall f1-score of 0.89. We consider f1-score as the main evaluation parameter since it is a harmonic mean between precision and recall, the other two evaluation parameters. On the other hand, GRU despite having an overall f1-score of 0.81, has 0 precision, recall, and f1-score for the class "retired". This signifies that the model didn't learn properly with the current hyperparameters perhaps. Again, it is noticeable that all the models have a higher precision, recall, and f1-score on the "graduated" class. The "retired" class has fewer correct predictions, but it is much better than what we had previously obtained without zero padding and lack of a proper train-test split.

SHAP Analysis

Just like the lasso regression, the SHAP analysis in Fig 4b shows that features like "unresolved_prs" and "active_devs" have more impact on the LSTM model in the negative direction, whereas features like "avg_thread_length" have less impact. Interestingly, Lasso assigns zero weight to "new_prs" and "rejected_prs" and "accepted_prs", but they do seem to be used by the model according to SHAP summary in Fig 4b. This analysis answers the second part of our RQ2 i.e. ***How does each feature impact the model outcome?*** The plots for other 3 models are shown in the Appendix section and follow similar trends.



(a) Feature importance determined by LASSO



(b) SHAP features influencing the LSTM model

Figure 4: Comparison of feature importance methods

5 Threats to validity

One of the principal limitations of our study is the dataset, as it contains only ASFI projects. Thus, the findings of our work may not generalize well to other non-incubated open-source projects. Further, our data has an inherent imbalance in the status of the projects i.e. 180 graduated and 26 retired projects, which translates to class imbalance for our prediction models, which can result in artificially high overall accuracy. We observed this limitation by assessing class-wise precision shown in Table 4. To mitigate this, we computed class weights, which improved the performance from a precision of 0 for retired projects across most models (except GRU) to the current precision values shown in Table 4. Adding Zero padding and fixing train-test split also increased the precision for retired projects from ≈ 0.15 across all models to the current values (except GRU). Additionally, our PR features are structured per month, and some PRs show activity (e.g. acceptance/rejection/comments) after several months, which leads to a *NaN* value being recorded for the months where there was no activity for a specific PR. We mitigate this by using a value of "0" for such cases, so the model can learn the absence of activity. However, this may not be optimal since it introduces sudden spikes in the data (e.g. if there is only 1 PR, and it is accepted after 5 months, *avg_time_to_acceptance* metric will reflect that jump in the 5th month, after a series of 0s).

6 Discussion

6.1 Principal Outcomes

RQ1[1]

- **PR Number Metrics:** On the initial analysis, we found metrics like the Ratio of Merged to Closed PRs quite promising as this ratio was higher in the graduated projects compared to the retired ones. It matches our initial assumption as merging more means the project is getting updated constantly and thriving. Now let us discuss the number of accepted, closed, and unresolved PR metrics which are normalized with the number of total active developers. The metric "no. of accepted PRs" shows similar behavior but is not as clear cut as the merge-to-reject ratio. On the other hand, the metric "no. of rejected PRs" does not show a clear impact on sustainability, as PR rejection will generally go up with the number of new open PRs. More rejected PRs in this case does not necessarily mean that the project is dying, rather it could also mean the contributors are being very careful about the quality of coding. So this metric alone does not give a good indication. For the final one, the

[1]This RQ asked an analysis question, thus the "specific answer" encompasses these all final thoughts on the result.

number of unresolved PRs seems to have a more visible effect on a project's sustainability. It is understandable that some of the PRs may remain unresolved if there are too many PRs getting created on a weekly basis, but if a large portion of them remain unresolved it could indicate the inactiveness of the developers of the project which eventually may lead to halted progression and getting retired.

- **Communication Latency:** The mean values are very large due to the outliers, causing the values to not reflect on most of the projects. To understand the difference between graduated and retired projects more accurately, we will focus on the median and the box plots.
  - **First Response Time:** The median of graduated projects was slightly lower compared to the retired projects. For box plot 3a, the median lines are similar, but graduated projects show more variability in the response time. However, the median values show that graduated projects generally respond slightly faster. The faster response time might be due to the graduated projects having a more active and engaged community, this increases the chances that a developer is available to respond to the new PRs.
  - **PR Resolution Time:** The median of graduated projects was higher, and the box plot 3b median line for the graduated projects is also slightly higher. This shows graduated projects generally take longer to resolve PRs. The higher resolution time might be due to a more complex PR review process involving more developers, discussions, and quality checks. The process requires more time but will result in more accurate merges on higher quality PRs or rejects less useful PRs. On the other hand, retired projects might have quick but low-quality merges/rejects due to less active developers.
  - **Average Response Time:** Box plots 3c are similar but the median values show that graduated projects generally have faster response times. This can also be due to graduated projects having a more active and engaged community than retired projects. So the PR issues get discussed more frequently, leading to better PR resolution, which leads to better project sustainability.
- **Core Developers:** Core developers' PRs are accepted/rejected significantly more rapidly across both sustainable and unsustainable projects compared to other developers' PRs, which aligns with our expectations since core developers are usually also the maintainers of the project (especially for smaller projects), and can merge or close their own PRs as well. Further, core developers usually contribute important code so it is expected that other core developers might prioritize evaluating their PRs more than they would for other PRs.

RQ2

- The reason for low predictive performance on the "retired" class could be inferred upon from the point that the number of instances of "retired" class is still low. However, it is still evident from table 4 that **PR features can help predict sustainability to a good extent** with appropriate hyperparameter tuning and if there were more instances of "retired" class to learn from.
- **Feature Importance:** Based on the Lasso Regression plot in Figure 5, **"unresolved_prs" and "active_devs" are the most important features** and they negatively impact the project's predictions, meaning if the number of the two features increases, then the model predicts the project is less likely to graduate. The summary plot of SHAP for LSTM also aligns with the Lasso, as "unresolved_prs" and "active_devs" have high negative values.
  - **Unresolved PR:** If a project has a high number of unresolved PR, then that indicates the project has inefficient PR review processes such as slow review time and inconsistent review standards across developers. Also, the lack of developer engagements can cause PRs to be

abandoned and unresolved. All these factors can causing the PRs to take longer to solve and the unresolved PR piling up. Furthermore, the productivity and development of the project will be slowing down, leading to the failure of the project.

- **Active Developer:** More active developers are generally assumed to be beneficial for a project's success. However, the higher number of active developers might cause difficulty in coordination. The developers might create more code conflicts and take longer to make decisions due to communication challenges. Also, the PR reviewers might get overwhelmed as more developers mean more PR and more time for PR review. These factors will cause the projects to be difficult to maintain and lead to failure.
- All other features have low coefficients (Lasso) and SHAP values, meaning they have low contributions to the model's predictions and are not very important.

6.2 Comparison to Prior Studies

Our study focuses on the sustainability of Apache Software Foundation Incubator (ASFI) projects by focusing on pull request (PR) activity and its impact on project graduation. While prior works have explored socio-technical network modeling in forecasting sustainability [6] and code quality metrics [7], our approach provides a more detailed analysis of PR interactions, core developer contributions, and PR response times as key sustainability indicators. We also train models based on PR features and infer if they are suitable for predicting sustainability of projects. Yin et al. [6] integrates social and technical networks to predict sustainability early in the incubation period. Their findings suggest that projects with decentralized communication and strong commit networks are more likely to graduate, a conclusion that aligns with our observations regarding the importance of core developers in PR discussions and acceptance rates. However, unlike our study, their approach does not specifically analyze PR activity.

Similarly, Stănciulescu et al. [7] provides insights into how code complexity, commit types, and developer contributions correlate with sustainability. They find that graduated projects have more structured contributions, lower code duplication, and a balance between major and minor contributors—factors that complement our findings on core developer PR engagement. While their study emphasizes code quality and process metrics, our work highlights the impact of PR merge or close in project sustainability. Thus, our PR-focused analysis answers the relation between PRs and predicting sustainability.

6.3 Future Work

Future studies can experiment with more models (e.g. Transformers) and finetune them further, since we were restricted by time constraints and could only test a limited number of hyperparameters for our models. In addition, more measures can be taken to deal with the *NaN* values in our dataset, which occur due to inherent PR properties, especially in timing attributes since some PRs are open across months. Such measures could include filling the previous known value, accumulating data on a tri-monthly basis, or equal distribution of a value across previous months that recorded no activity. We could not try these variations due to time constraints of running the 4 models multiple times after each change, but we believe this could help improve model performance. Future research can also evaluate to what extent the *content* of the PRs and language of the comments made on them help predict whether the PR will be closed or merged.

Additionally, our research mainly involved working with pull request data so we ignored the ASFI projects that were using SVN(Subversion). We also observed that the retirement rate of recent ASFI projects was low which explains our low sample of retired projects compared to graduated projects. If SVN projects have any equivalent of PRs, future work can include them to increase the retired project sample and then test our models. However, it will be challenging to perform data collection and pre-processing to include SVN as the structure of SVN and GitHub is very different.

6.4 Conclusion

This study explored how pull request (PR) activity influences the sustainability of Apache Software Foundation Incubator (ASFI) projects. We found that graduated projects exhibit higher PR merge-to-close ratios, fewer unresolved PRs, and more structured review processes. Core developers play a key role, with their PRs receiving priority across all projects. Our predictive models, trained with PR features, performed well for graduated projects but struggled with retired ones due to class imbalance and feature limitations. Lasso regression and SHAP analysis identified unresolved PRs and active developer counts as key factors, though the latter showed an unexpected negative correlation. While our study is limited by dataset constraints and the exclusion of non-GitHub projects, future work could enhance accuracy by incorporating socio-technical factors, transformer-based models, and PR discussion analysis. These insights can help maintainers improve PR processes and enhance long-term project sustainability.

7 Team Membership and Attestation

Declaration: We agree with the contents of this final project report. - Jun Wu, Raiyan Jahangir, Humaira Fasih Ahmed, Zane Reis, Ahnaf Faisal
Contributions: Jun Wu - Worked on communication latency metrics, developed and finetuned LSTM, ran all the models to compute the results, and developed Lasso and SHAP. Wrote section 2, 3.3, 3.4, 4, 6.1 **Zane Reis** - Developed the pipeline for scraping and formatting the PR data using GitHub URLs and detailed the process in 3.1 and 3.2. Created monthly data for models. **Raiyan Jahangir** - Worked on determining the relationship between PR by core developers and project sustainability, developed and ran the models and edited the whole report. **Ahnaf Faisal** -Worked with Zane to rectify the issue with GitHub URLs, worked on PR number metrics + analysis, edited model code + SHAP analysis, edited sections 3, 4 and 6 of report. **Humaira Fasih Ahmed** - Refined RQs, re-wrote section 5, 6.3, 3.4.2 with new results, corrected zero padding + train-test split issue, fixed lasso, finetuned final LSTM, edited/proofread report extensively for writing clarity.

Code & Data Availability Statement

Our data and code are available on Github: link. We have a README file that states how to download the project and run the codes. **train_all_models.ipynb** file has our final models.

LLM Usage Documentation

We have used LLMs to generate our models, and PR analysis codes (***model***, ***pr_data_analysis***, ***communication_latency*** directories) and some parts of ***core_developers***, ***data***, and ***tools*** directories, since there was no restriction against code use for trivial tasks. We modified code to fit our needs where needed. We have done our best to write the report without LLMs and have proofread/edited multiple times. However, there may be some usage to achieve clarity of sentences e.g. Section 6.2 and 6.4 might be paraphrased with LLM.

References

[1] R. G. Rocha, C. Costa, C. Rodrigues, R. Ribeiro de Azevedo, I. H. Junior, S. Meira, and R. Prikladnicki, "Collaboration models in distributed software development: A systematic review," *CLEI Electronic Journal*, vol. 14, no. 2, pp. 1–1, 2011.

[2] C. R. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 241–250.

[3] F. Zampetti, G. Bavota, G. Canfora, and M. Di Penta, "A study on the interplay between pull request review and continuous integration builds," in *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2019, pp. 38–48.

[4] M. Alfadel, N. A. Nagy, D. E. Costa, R. Abdalkareem, and E. Shihab, "Empirical analysis of security-related code reviews in npm packages," *Journal of Systems and Software*, vol. 203, p. 111752, 2023.

[5] L. Yin, Z. Zhang, Q. Xuan, and V. Filkov, "Apache software foundation incubator project sustainability dataset," in *2021 ieee/acm 18th international conference on mining software repositories (msr)*. IEEE, 2021, pp. 595–599.

[6] L. Yin, Z. Chen, Q. Xuan, and V. Filkov, "Sustainability forecasting for apache incubator projects," in *Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 1056–1067.

[7] S. Stănciulescu, L. Yin, and V. Filkov, "Code, quality, and process metrics in graduated and retired asfi projects," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 495–506.

[8] L. Yin, X. Zhang, and V. Filkov, "On the self-governance and episodic changes in apache incubator projects: An empirical study," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 678–689.

[9] A. Dhasmana, A. Roy, D. S. Jas, K. Kaur, and P. Prugsanapan, "Forking around: Correlation of forking practices with the success of a project," *arXiv preprint arXiv:2112.14464*, 2021.

[10] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[11] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

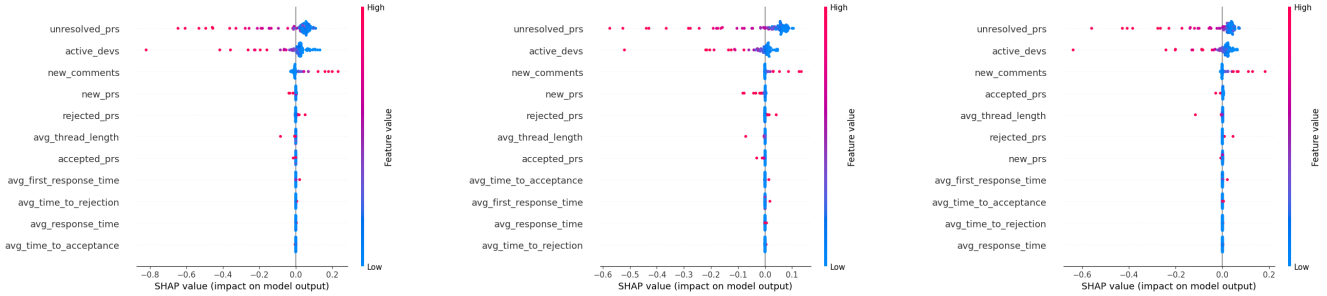
[12] R. Jahangir, N. S. Mohim, N. I. Khan, M. Akhtaruzzaman, and M. N. Islam, "Proposing novel recurrent neural network architectures for infant cry detection in domestic context," in *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*. IEEE, 2023, pp. 7–12.

[13] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[14] Z. Zhu, W. Dai, Y. Hu, and J. Li, "Speech emotion recognition model based on bi-gru and focal loss," *Pattern Recognition Letters*, vol. 140, pp. 358–365, 2020.

[15] J. Ranstam and J. A. Cook, "Lasso regression," *Journal of British Surgery*, vol. 105, no. 10, pp. 1348–1348, 2018.

Appendix



(a) Bi-LSTM (b) GRU (c) Bi-GRU

Figure 5: SHAP results for different RNN-based models.