

# Fetching Linked Pull Requests and Issues from GitHub

The goal is to create a dataset to train a risk assessment ai tool

## Objective

Collect all closed PRs,ISSUES link them respectively, then extract information from

- Title
- Description
- Comments
- Patch (code)
- How long did we work on the issues?

## Approach

### Using GitHub API

Download all PRs, ISSUES  
Saved in json,csv

### Mapping Issues and PRS

- ☐ keyword mapping  
(<https://docs.github.com/en/issues/tracking-your-work-with-issues/using-issues/linking-a-pull-request-to-an-issue>)
- ☐ Timeline Events via GitHub API  
(<https://docs.github.com/en/rest/using-the-rest-api/issue-event-types?apiVersion=2022-11-28>)  
**This gives you events like:**
  - cross-referenced
  - connected
  - Mentioned
- ☒ Semantic Matching ~~SentenceTransformer (all-MiniLM-L6-v2)~~

# Using graphql

is:pr linked:issue

GitHub Web UI uses internal GraphQL APIs.

These GraphQL APIs use a persisted query system (query IDs like 6e328ebc0ae57f444533040aec851589).

Some fields (like linkedIssues) are only available inside GitHub's internal GraphQL schema and NOT exposed to external developers.

```
{
  "query": "6e328ebc0ae57f444533040aec851589",
  "variables": {
    "includeReactions": false,
    "name": "product-apim",
    "owner": "wso2",
    "query": "is:pr linked:pr repo:wso2/product-apim sort:created-desc",
    "skip": 0
  }
}
```

## In the public GraphQL API

- linkedIssues is NOT a public GraphQL field on the PullRequest type!
- The GitHub Web UI is using private/internal GraphQL schemas for that linkedIssues field.
- We can't directly query linked issues using the public GitHub GraphQL API.

## GitHub Does Not Provide

- ❌ No direct REST API endpoint like:  
/repos/:owner/:repo/issues/:issue\_number/linked\_prs
- ❌ No guaranteed semantic match — you'll need AI/NLP for that
- ❌ No merging of all links into a single endpoint (UI-based + keyword + commit)

Next, we can try

Scrape GitHub Web UI using Playwright or Selenium.

---

# wso2/product-apim

## 1. Executive Summary

The analysis of the WSO2 Product API Manager GitHub repository reveals a total of 13,423 tracked items, consisting of 8,188 issues and 5,244 pull requests.

## 2. Repository Overview

Repository: wso2/product-apim

Total Items: 13,423

Item Type	Total	Open	Closed	Closure Rate
Issues	8,188	626	7,562	92.4%
Pull Requests	5,244	30	5,214	99.4%

## 3. Pull Request Analysis

The analysis of pull requests reveals important insights about the repository's contribution workflow:

- Merged PRs: The majority of closed pull requests were successfully merged into the codebase
- Closed without Merging: A smaller portion of pull requests were closed without being merged

## 4. Issue Linkage Analysis

Category	Open Issues	Closed Issues	Open PRs	Closed PRs
Linked to issues/PRs	16	1,680	2	257
Not linked	610	5,882	28	4,957

The linkage analysis shows:

- Only 1,696 issues (20.7%) are linked to pull requests
- Only 259 PRs (4.9%) are explicitly linked to issues
- The majority of both issues and PRs remain unlinked

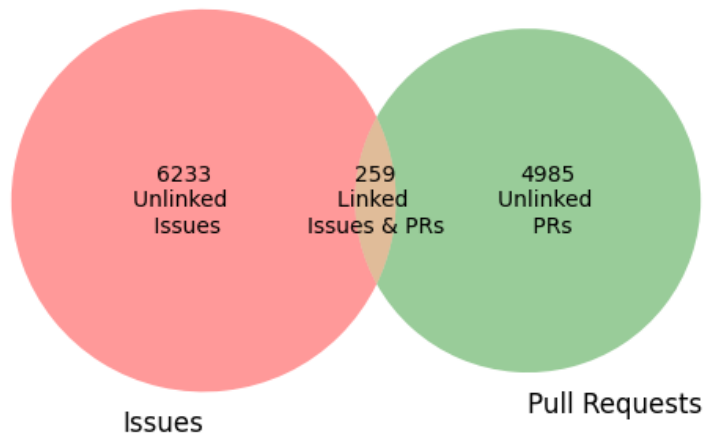
	Count
Total Issues	8188
Open Issues	626
Closed Issues	7562
Total PRs	5244
Open PRs	30
Closed PRs	5214
Merged PRs	4583

Data manually noted based on Github UI search query  
Github data manually collected based on Github UI search queries.

Name	Issues		Pull Request	
	Open	Close	Open	Close
is:issue/is:pr	626	7562	30	5214
linked:issue	18	1937	2	257
linked:pr	16	1680	2	257
-linked:issue	638	10839	28	4957
-linked:pr	610	5882	28	4957

## 5. Visualization Insights

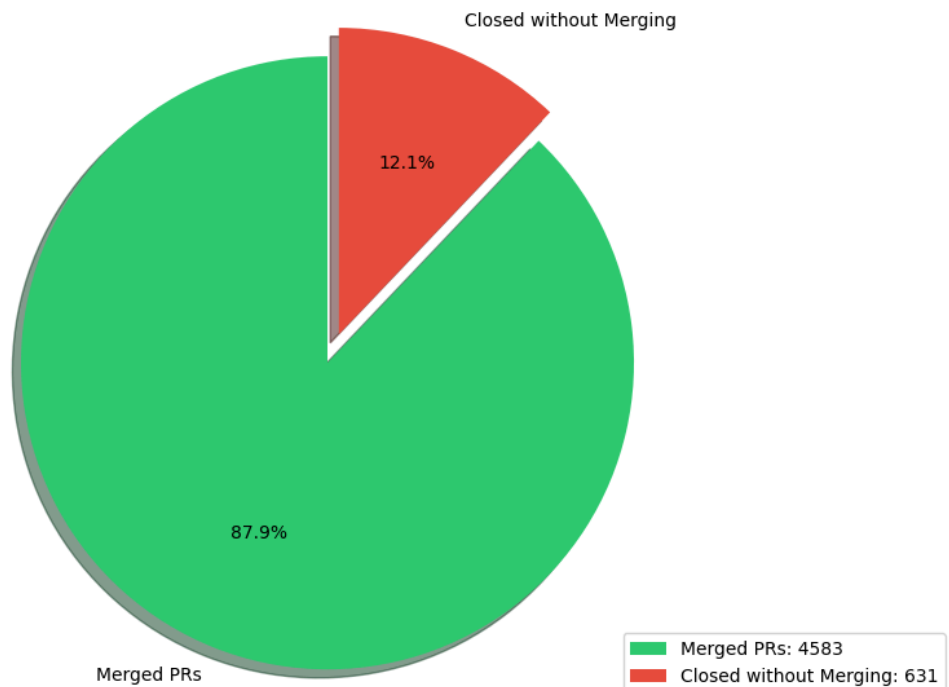
Venn Diagram: Issues and Pull Requests



The Venn diagram visualization illustrates the relationship between issues and pull requests, highlighting that

- Most issues exist independently without direct PR linkage
- Most PRs exist without explicit issue references
- Only a small intersection represents fully traceable work from issue to implementation

Distribution of Closed Pull Requests



The pie chart of pull request outcomes demonstrates:

- A significant majority of closed PRs were successfully merged
- A smaller portion was closed without merging

---

# ballerina-platform/ballerina-lang

## 1. Executive Summary

The analysis of the Ballerina Platform GitHub repository (ballerina-platform/ballerina-lang) reveals a total of 43,175 tracked items, consisting of 18,252 issues and 24,923 pull requests. This comprehensive examination provides insights into the project's development workflow, contribution patterns, and item management efficiency.

## 2. Repository Overview

Repository: ballerina-platform/ballerina-lang  
Total Items: 43,175

Item Type	Total	Open	Closed	Closure Rate
Issues	18,252	1,481	16,771	91.9%
Pull Requests	24,923	16	24,907	99.9%

## 3. Pull Request Analysis

The analysis of pull requests reveals important insights about the repository's contribution workflow:

- Merged PRs: 21,789 (87.5% of closed PRs) were successfully merged into the codebase
- Closed without Merging: 3,118 (12.5% of closed PRs) were closed without being merged

## 4. Issue Linkage Analysis

Category	Open Issues	Closed Issues	Open PRs	Closed PRs
----------	-------------	---------------	----------	------------

Linked to issues/PRs	108	6804	11	6,047
Not linked	1,373	9967	5	18,860

The linkage analysis shows:

- 6912 issues (37.9%) are linked to pull requests
- 6,058 PRs (24.3%) are explicitly linked to issues
- A significant portion of PRs (62.1%) remain unlinked to specific issues

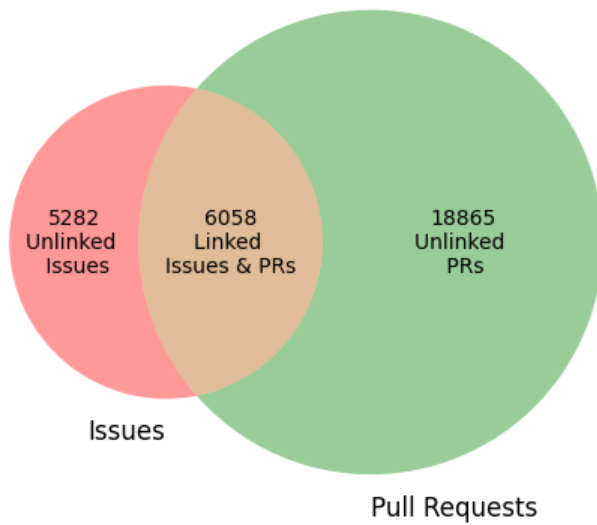
	Count
Total Issues	18252
Open Issues	1481
Closed Issues	16771
Total PRs	24923
Open PRs	16
Closed PRs	24907
Merged PRs	21793

Data manually noted based on Github UI search query  
Github data manually collected based on Github UI search queries.

Name	Issues		Pull Request	
	Open	Close	Open	Close
is:issue/is:pr	1481	16771	16	24907
linked:issue	119	12851	11	6047
linked:pr	108	6804	11	6047
-linked:issue	1378	28827	5	18860
-linked:pr	1373	9967	5	18860

## 5. Visualization Insights

Venn Diagram: Issues and Pull Requests



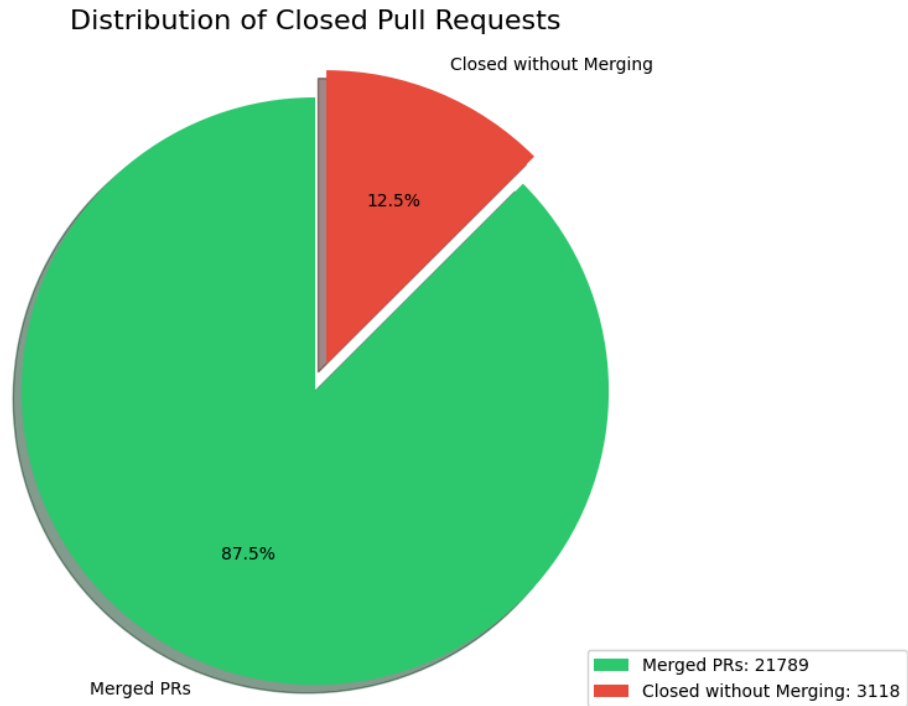
Illustrates the relationship between issues and pull requests

Shows 5,282 issues exist without linked PRs

Shows 18,865 PRs exist without explicit issue references

Only 6,058 items represent fully traceable work from issue to implementation

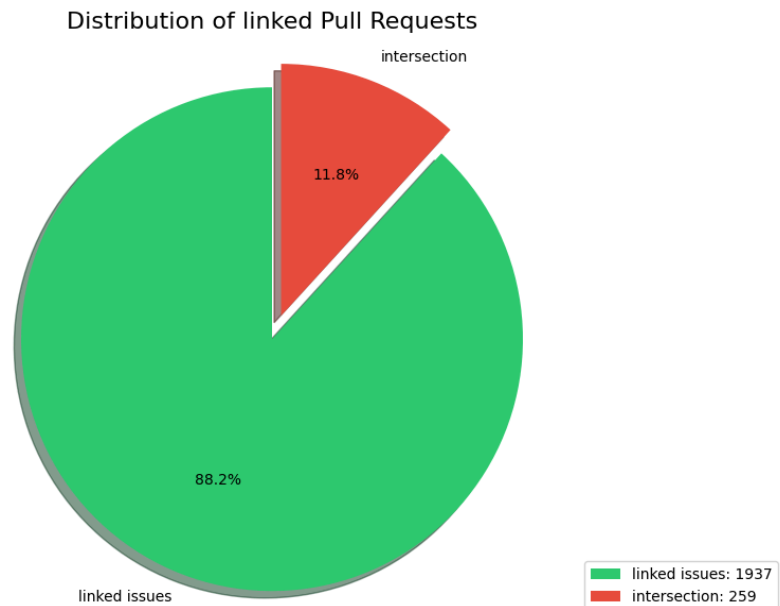




Demonstrates that 87.5% of closed PRs were successfully merged  
Only 12.5% were closed without merging

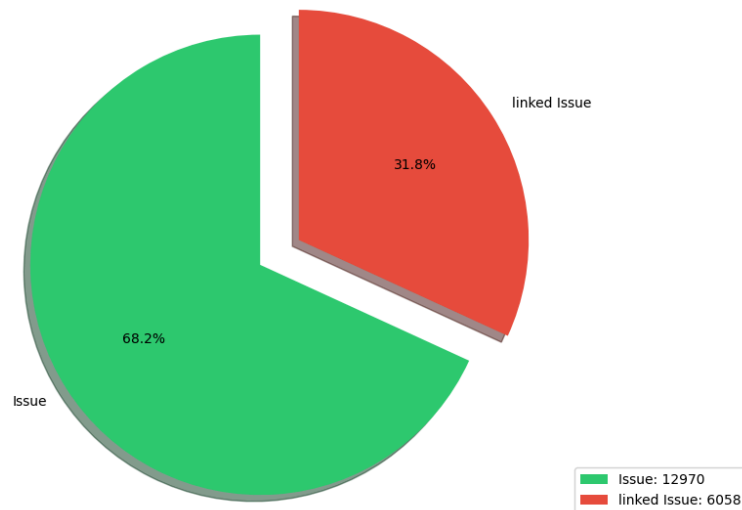
## 6. Distribution of linked Pull Request

wso2/product-apim



ballerina-platform/ballerina-lang

Distribution of linked Pull Requests



## Keyword matching from issues to pr

While implementing this I found some patterns like they use key word **closed with #123** and

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

Like key words

## Approach

Run the script -> manually validate the output csv -> randomly check unmatched issues ( but linked according github ) to find a new pattern

None

graph TD

```
A[Run Script] --> B(Manually Validate CSV Output);
B --> C{Randomly Check Unmatched Issues (Linked on GitHub)};
C -- Found New Pattern --> D[Update Script with New Pattern];
C -- No New Pattern --> E[End Process];
D --> A;
```

## Process

1. In first iteration I can match just **33** only out of 16784  
📄 linked\_issues\_prs\_issue\_only1.csv
2. In second iteration **103 by changing finding order** and added **closed with #123**  
pattern 📄 linked\_issues\_prs\_issue\_only2.csv
3. In third iteration **293** matched added **Fixed in #** and introduce priority  
📄 linked\_issues\_prs\_issue\_only3.csv
4. In 4th iteration also get same 296 matches only no big improvement  
📄 linked\_issues\_prs\_issue\_only4.csv

## NOTE

---

During manual review of issues and their timelines, it was observed that referencing pull request numbers within the title or body of an issue is infrequent. However, such references do occur periodically within the **comment sections**.

Surprisingly every closed Issue has a **close event** it will refer either **closed by completing** or **closed by #123** but unfortunately the github end points are not exposing that information to us

Since there is another event called **cross-reference** we can try this

Also as **Srinath Perera** said rather than linking issue -> pr , pr -> issue would be appropriate next I will implement that

---

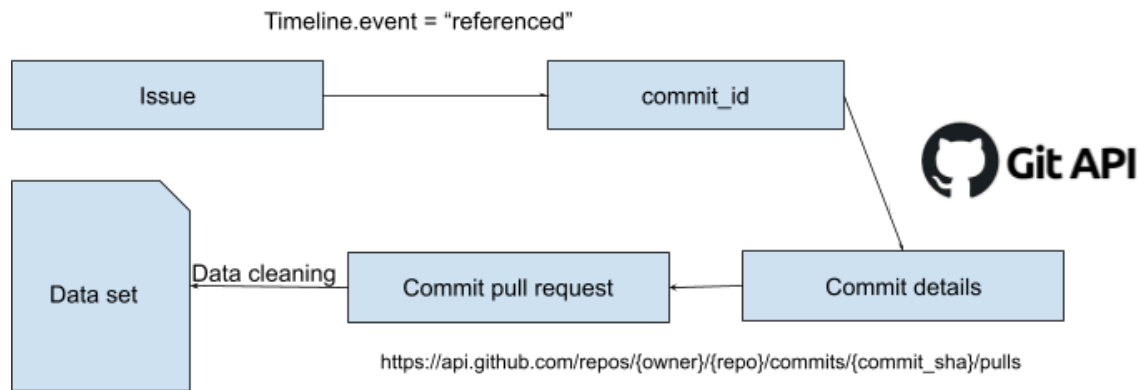
## Linking issues => pr by **referenced** event with **commit\_id**

```

JSON
{
  issies:{...},
  comments:[...],
  timeline:[{
    event:"referenced",
    commit_id:#,
    commit_url:"http://..."
    ...
  }]
}

```

## Linking Issues to Pull Requests Using Commit References



Among 16,788 closed issues, 1,497 have a timeline event marked as "referenced". This means a commit directly mentioned these issues.

We can confirm these links by checking if the commit message contains keywords like "fix" followed by the relevant issue number.

■ linked\_issues\_commite\_id.csv

Using this method, we initially found 4,218 combinations where 2,666 commits referenced a linked issue number (63%). The data is available in ■ message\_contains\_issue\_number.csv

By manually reviewing commit messages that were not initially linked, we discovered that direct links and keywords were often used. After improving the script, we successfully linked 4,104 combinations out of the initial 4,218 (97%).

The remaining 114 commits referenced issues as related or without using clear keywords or proper linking methods. This data is in ■ message\_contains\_issue\_reference.csv

Next, we tried to find the pull requests (PRs) associated with these commit IDs to directly link issues to PRs.

After retrieving the PRs for each commit ID, we found that 1,942 issues were closed by a commit **without an associated pull request**. This is a key finding: **some issues were resolved through direct commits**.

Our main goal is to connect issues with PRs. To achieve this, we cleaned the data by removing any duplicate combinations of issues and PRs. Finally, we identified 1,858 unique pairs of linked issues and PRs. This cleaned dataset is available at

📄 [linked\\_issues\\_commit\\_pr\\_data\\_cleaned.csv](#)

After the random manual validation of this dataset we can use this in our final data creation

---

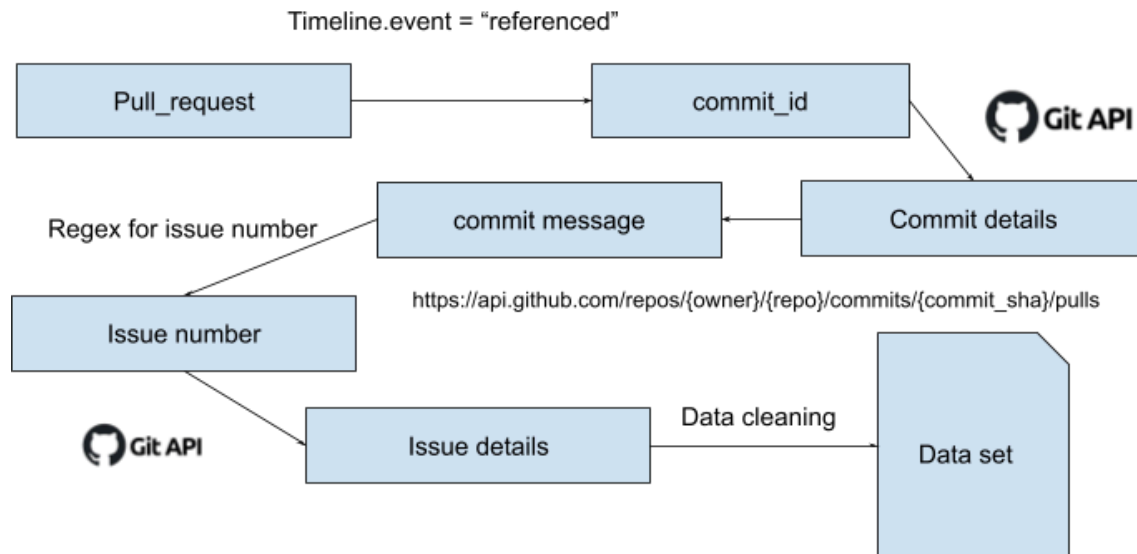
We can perform the same thing with Pull\_request data

## Linking pull request => issue by **referenced** event with commit\_id

Among 25055 pull request

```
None
{
  pull_request:{...},
  comments:[...],
  timeline:[{
    event:"referenced",
    commit_id:#,
    commit_url:"http://..."
    ...
  }]
}
```

998 are referenced pull\_request her mostly pull\_request referencing another pull\_request but in some cases issues also referenced our goal is extract those issue for that I used



This approach as a result we got 2308 combinations issue

■ linked\_pull\_request\_commite\_id.csv

From there keyword matching specially i Used

Python

```

# Covers: fix(es/ed/ing), clos(e/es/ed), resolv(e/es/ed), issue(s)
keyword_pattern =
r'\b(?:fix(?:es|ed|ing)?|clos(?:e|es|ed)|resolv(?:e|es|ed)|issue(?:s)?)\b'

# Optimized patterns to find issue numbers, ordered by specificity.
issue_patterns = [
    # Priority 1: Keywords followed by a full GitHub issue or pull request
    # URL.
    rf'{keyword_pattern}\s+https?://github\.com/[^\s/]+\s+[^\s/]+/(?:issues|pull)/(\d+)',

    # Priority 2: Keywords followed by #number.
    rf'{keyword_pattern}\s+#(\d+)\b',

    # Priority 3: Keywords followed by a plain number (no #).
    rf'{keyword_pattern}\s+(\d+)\b',

    # Priority 4: Keywords followed by "Fix <something> (from #number)".
    rf'{keyword_pattern}.*\s+(from\s+#(\d+)\s+)',

```

```

# Priority 5: Keywords followed by "Fix <something> #number".
rf'{keyword_pattern}.*#(\d+)\b',

# Priority 6: Keywords followed by "part of #number" or "part of issue
#number".
rf'{keyword_pattern}.*(?:#(\d+)\b|part of\s+(?:issue\s+)?#(\d+)\b)',

# Priority 7: Keywords followed by "/fix-#number" or "/fix-number".
rf'{keyword_pattern}.*(?:/fix-#(\d+)\b|/fix-(\d+)\b)',

# Priority 8: Specific pattern to extract issue numbers from "Fixes"
URL.
rf'Fixes:\s+https?://github\.com/[^/]+/[^/]+/issues/(\d+)\'

]

```

Through several attempts, patterns were found in commit messages to identify issue numbers or links. These patterns looked for keywords like "fix," "resolve," or "close" followed by an issue reference. This process resulted in 515 matches, extracting issue numbers from commit messages, as detailed in `commits_with_extracted_issue_references.csv`. During this filtering, references to backports, patches, addressing review suggestions, and merging pull requests were excluded.

After cleaning this data, 426 unique connections between pull requests, commits, and issues were identified. This refined dataset can be found in

`linked_pr_commit_issue_data_cleaned.csv`.

The patterns used to find these connections included:

- Keywords like "fix(es/ed/ing)," "clos(e/es/ed)," "resolv(e/es/ed)," and "issue(s)".

The search for issue numbers followed these rules, ordered by how specific they were:

- Keywords with a Full GitHub URL:** Looking for keywords followed by a complete link to a GitHub issue or pull request.
- Keywords with #number:** Finding keywords followed by a hash symbol (#) and a number.
- Keywords with a Plain Number:** Identifying keywords followed by a number without a

hash symbol.

4. **Keywords with "Fix <something> (from #number)":** Searching for the keyword "Fix" along with other words and an issue number in parentheses.
5. **Keywords with "Fix <something> #number":** Finding the keyword "Fix" with some text and then an issue number with a hash symbol.
6. **Keywords with "part of #number" or "part of issue #number":** Identifying keywords followed by a phrase indicating it's part of a specific issue number.
7. **Keywords with "/fix-#number" or "/fix-number":** Looking for keywords followed by "/fix-" and an issue number.
8. **Specific "Fixes:" URL Pattern:** Identifying lines that start with "Fixes:" followed by a GitHub issue URL.

These patterns helped to automatically link issues mentioned in commit messages to the corresponding code changes.

---

## Common GitHub Timeline Events

- **Assigned** → An issue or pull request was assigned to a user.
- **Closed** → The issue or pull request was closed.
- **Commented** → A comment was added to an issue or pull request.
- **Committed** → A commit was added to a pull request's branch.
- **Cross-referenced** → An issue was referenced from another issue.
- **Demilestoned** → An issue was removed from a milestone.
- **Deployed** → A pull request was deployed.
- **Head Ref Deleted** → The pull request's branch was deleted.
- **Head Ref Restored** → The pull request's branch was restored.
- **Labeled** → A label was added to an issue.
- **Locked** → An issue was locked.
- **Mentioned** → A user was @mentioned in an issue or pull request.
- **Merged** → A pull request was merged.
- **Milestoned** → An issue was added to a milestone.
- **Referenced** → An issue was referenced from a commit message.
- **Renamed** → The issue title was changed.
- **Reopened** → A closed issue was reopened.
- **Subscribed** → A user subscribed to receive notifications for an issue.
- **Unassigned** → An issue was unassigned from a user.
- **Unlabeled** → A label was removed from an issue.
- **Unlocked** → An issue was unlocked.
- **Unsubscribed** → A user unsubscribed from an issue.

Here we can use for linking pullrequest to issue



Closed, croess-refferenced, referenced

---

## Keyword matching from pull request to issue

To map the issues lets break down the approached into smaller parts and and focus one by one

- Key word mapping
  - Title **Completed** ▾
  - Description (body) **Completed** ▾
  - Comments body **Not Started** ▾
- Timeline event
  - Closed **Blocked** ▾
  - Referenced **Completed** ▾
  - Cross-referenced **Not Started** ▾

Let's create a base csv with pull\_request number , pull\_request link ,title,description  
From there perform the keyword matching very straight forward

📄 closed\_prs\_summary.csv

Out of 25055 pull request 25042 are closed

### Title

For mapping the pull request to issue first find the key words and extract the issue number or github link then create a new csv , some title address more than one issues for that case we create more than one records separately in the output we get all the original csv headers and additionally

- extracted\_issue\_number,
- Extracted\_issue\_link,
- Found\_in\_title,
- original\_row\_index

Through several iterations of regex mapping and updating newly found patterns, 313 combinations were identified, detailed in

📄 closed\_prs\_summary\_with\_extracted\_issues\_from\_title.csv . The remaining pull requests did not reference issue numbers or GitHub links in their titles; some were used for merges and migrations. All titles containing numbers or links were manually verified.

So this complete we can't extract issue more than this from pull\_request title according to this repository all patterns were discovered

These are the keywords and pattern that we have used

```
Python
# Covers: fix(es/ed/ing), clos(e/es/ed), resolv(e/es/ed), issue(s), Backport,
Revert(s)
keyword_pattern =
r'\b(?:fix(?:es|ed|ing)?|clos(?:e|es|ed)|resolv(?:e|es|ed)|issue|Backport|Rever
t(?:s)?)\b'

# Comprehensive patterns to find issue numbers, ordered by specificity
issue_patterns = [
    # Priority 1: Keywords followed by colon and GitHub URL (most specific)

rf'{keyword_pattern}:\s+https?://github\.com/[^/\\s]+/[^/\\s]+/(?:issues|pull)/(\d
+)',

    # Priority 2: Keywords followed by full GitHub issue URL

rf'{keyword_pattern}\s+https?://github\.com/[^/\\s]+/[^/\\s]+/(?:issues|pull)/(\d
+)',

    # Priority 3: Keywords followed by repo path and issue number
rf'{keyword_pattern}\s+[^^/\\s]+/[^/\\s]+/issues/(\d+)',

    # Priority 4: Keywords + <text> + GitHub issue links + optional plain
number

rf'{keyword_pattern}.*<.*>\s+https?://github\.com/[^/\\s]+/[^/\\s]+/(?:issues|pu
ll)/(\d+)(?:\s+(\d+))*',

    # Priority 5: Keywords + <text> + multiple issue numbers
rf'{keyword_pattern}.*<.*>\s+#(\d+)(?:\s+#(\d+))*',

    # Priority 6: Keywords followed by square brackets
rf'{keyword_pattern}.*\[(\d+)\]',

    # Priority 7: Square brackets followed by keywords
rf'\[(\d+)\].*{keyword_pattern}',

    # Priority 8: Keywords followed by (from #number)
rf'{keyword_pattern}.*\s(from\s+(\d+)\s+)',
```

```

# Priority 9: Keywords followed by "part of #number" or "part of issue
#number"
rf'{keyword_pattern}.*part\s+of\s+(?:issue\s+)?#(\d+)',

# Priority 10: Keywords followed by "/fix-#number" or "/fix-number"
rf'{keyword_pattern}.*(?:/fix-#(\d+)/fix-(\d+))',

# Priority 11: Multiple issues in one line (e.g., "fixes #123, #456,
#789")
rf'{keyword_pattern}[^#]*?((?:#\d+(?:\s*,\s*#\d+)*))+',

# Priority 12: Keywords followed by any text and #number
rf'{keyword_pattern}.*#(\d+)',

# Priority 13: Keywords followed by #number (direct)
rf'{keyword_pattern}\s+#(\d+)',

# Priority 14: Keywords followed by plain number (no #)
rf'{keyword_pattern}\s+(\d+)\b',

# Priority 15: #number followed by keyword
rf'#(\d+)\s+{keyword_pattern}',

# Priority 16: Specific "Address review suggestions" pattern
rf'Address\s+review\s+suggestions.*#(\d+)',

# Priority 17: Multiple issue numbers after "issues" (e.g., "issues
#1159 #947")
r'\bissues?\s+#(\d+)(?:\s+#(\d+))*',

# Priority 18: GitHub URLs without keywords (issues only, not pull
requests)
r'https?://github\.com/[^/\s]+/[^/\s]+/issues/(\d+)',
]

```

## Description (Body)

From 25042 closed pull request there are 2772 pull request has no description (Body) we can process only 22270 ,pull request body address more than one issues for that case we create more than one records separately in the output we get all the original csv headers and additionally

- extracted\_issue\_number,
- Extracted\_issue\_link,
- found\_in\_body
- original\_row\_index

In the initial run we found 15 565 but

## NOTE

---

For the moment we need to ignore issues mentioned in Remarks section

None

do we need to consider issues mentioned in remark section

### ## Purpose

> Increase the test coverage in symbols() for module level declarations

Fixes #32159

Fixes #29628

### ## Approach

> Describe how you are implementing the solutions along with the design details.

### ## Samples

> Provide high-level details about the samples related to this feature.

### ## Remarks

Some of the tests/checks in this PR are commented (ignored). Those are buggy areas in the compiler/Semantic API. Following issues are for tracking these issues.

#32660

#32661

#32662

### ## Check List

```
[ ] Read the [Contributing
  Guide](https://github.com/ballerina-platform/ballerina-lang/blob/master/C
  ONTRIBUTING.md)

[ ] Updated Change Log

[ ] Checked Tooling Support (#<Issue Number>)

[x] Added necessary tests

- [x] Unit Tests
- [ ] Spec Conformance Tests
- [ ] Integration Tests
- [ ] Ballerina By Example Tests

[x] Increased Test Coverage

[ ] Added necessary documenta
tion

- [ ] API documentation
- [ ] Module documentation in Module.md files
- [ ] Ballerina By Examplesm
```

---

And I made a mistake when generating the issue link. If an issue is already mentioned as a link I extract the number and generate a link, but the issue is in some cases, issues from outside this repository are also referenced. So at that point the data is wrong I need fix that

The problem sorted now moving forward next steps

Creating data set to train the model

---

## Git blame

## Finding the buggy code

The **SZZ algorithm** is a technique used in software engineering to identify the commits in a version control system that likely introduced bugs. It was introduced by Śliwerski, Zimmermann, and Zeller—hence the name "SZZ."

Here's how it works in a nutshell:

1. **Start with a bug-fixing commit**—a change that resolves a known bug.
2. **Trace back** through the version history to find the lines of code that were modified to fix the bug.
3. **Identify the last commit** that touched those lines before the fix. That commit is flagged as a potential *bug-introducing change*

This method is widely used in **empirical software engineering research**, especially for tasks like defect prediction, software quality analysis, and training machine learning models to detect risky code changes

According to this research article <https://www.scribd.com/document/866959848/SZZ-Unleashed>

SZZ Unleashed: An Open Implementation of the SZZ Algorithm

Conference'17, July 2017, Washington, DC, USA

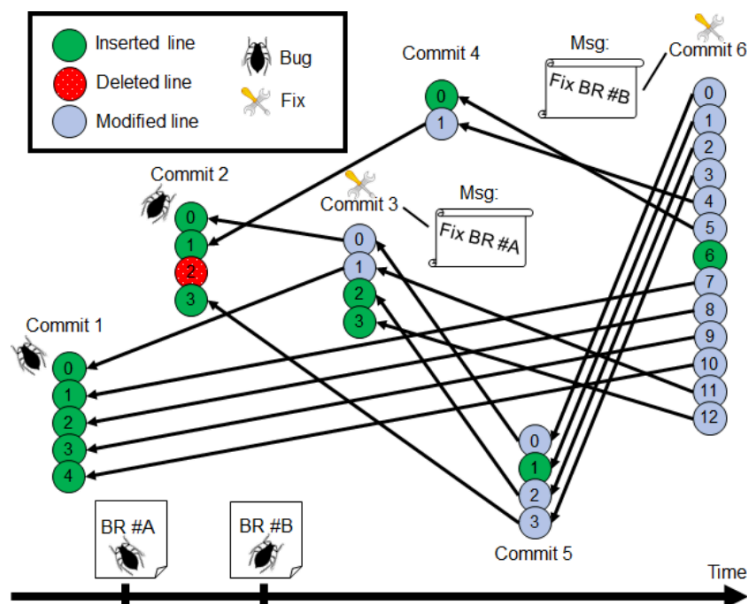


Figure 3: Lines of modified code in six commits to an example file. Arrows show the result of line mapping between commits.

The core SZZ algorithm operates on **commits**, not Pull Requests. The key is to bridge this gap. A merged Pull Request is represented in the repository's history by a specific **merge commit**. This merge commit is the "fix commit" we need for the SZZ analysis.

As an initial stage I started to extract the merged commit

But I found that in pull\_request there some pr has **merge\_commit\_sha** with **merged\_at:null** Among **25042 closed pull\_request** there are only **13 pull requests** are like this So I considering them as unmerged pull\_requests

Initially perform a pilot version with small number of data once all complete then we use our entire data set

---

As a pilot, I extracted merged commits from 100 records and successfully mapped 94 of them.

■ data\_with\_merge\_commit\_100.csv

Now we have 94 rows of data let's perform the SZZ analysis

All data has been successfully mapped. ■ data\_with\_merge\_commit.csv

---

The SZZ algorithm is a method used to pinpoint commits in a software repository that introduces bugs. The process involves two main phases:

- **Phase 1:** Bug reports (BRs) from an issue tracker are linked to the specific commits that fix the bugs. This is typically done by finding references to BRs in commit messages using regular expressions.
- **Phase 2:** For each bug-fixing commit, the **git blame** command is utilized to find previous commits that modified the same lines of code. These are considered candidates for the bug-introducing commit. Further analysis is done to rule out candidates, for example, by comparing the commit time with the bug report submission time.

---

In our case the phase one we followed a different approach since ballerina not using any Bug reporting system like jira bugreporting

Let's start the second phase

Identifying the bug introducing pull requests following the data set

■ data\_with\_merge\_commit\_100.csv

I extracted 372 rows from the first 10 records, as this process involves every added and deleted line of code. 📄 szz\_bug\_introducing\_commits\_for\_10\_records.csv

Field names are :

**bug\_introducing\_commit,commit\_message,author,date,file\_path,blamed\_by\_pr\_number,blamed\_by\_merge\_commit,original\_issue\_number**

Next move to the extract other features to then we can train the ai model

Let's continue our work by tackling the next major step **Feature Extraction**

successfully identified the "bug-introducing" commits, which will serve as our labels (the 'Y' variable). Now, we need to calculate the 16 features for *every commit* that will be used to train the model (the 'X' variables).

This is a complex task, so we will approach it incrementally. We'll start by creating a script that iterates through all the commits in your repository and calculates the most straightforward features first.

## The Plan for Feature Extraction

1. **Create a Foundation:** We will write a Python script (in notebook format) that iterates through each commit in the repository.
2. **Calculate Easy Features First:** We will implement the logic for features that can be easily derived from a commit object, like code churn (lines added/deleted, files changed).
3. **Outline Complex Features:** We will leave placeholders and provide guidance for the more complex features, such as developer experience and code coupling, which require more elaborate logic or external tools.

## Process

- ☒ **Create a Foundation python script**
- ☒ **Extract easy features first**
  - ☒ **Focus master branch**
- ☒ **Extract complex features**
  - ☒ **Experience matrices**
  - ☒ **Code coupling**
- ☒ **Labeling**

NOTE

---



I have mapped all the merge commit id `data_with_merge_commit.csv` 12831 merged PRs (from 15355 total)

**Assumption** > other commits are not properly merged

With these data stated the SZZ analysis

There instead of use all the data I just used **first 10 records only** to test the code then




I end up with **372 rows**

Manually validated randomly seems working (so we can iterate for whole dataset)

Then I started to script the Feature extraction

---

In this [research](#) they using 16 features those are

- ☒ Ft1 Lines of code added
- ☒ Ft2 Lines of code deleted
- ☒ Ft3 Files churned / Number of files
- ☒ Ft4 Lines of code in previous version 
- ☒ Ft5 Number of modified subsystems
- ☒ Ft6 Number of modified sub-directories
- ☒ Ft7 Entropy (spreading of changes)
- ☐ Ft8 Purpose of a change (e.g., bug fix) 
- ☒ Ft9 Number of previous committers
- ☒ Ft10 Time between committer's contributions
- ☒ Ft11 Number of unique changes 
- ☒ Ft12 Overall experience of committer
- ☒ Ft13 Recent experience of committer
- ☒ Ft14 Number of highly coupled files
- ☒ Ft15 Number of coupled files for all degrees
- ☒ Ft16 Number of non-modified coupled files

In the initial stage I'm extracting the **easy features** (Ft1,Ft2,Ft3,Ft6,Ft7)

Seems other features need complex operation **[Ft9-13] Experience metrics**

**[Ft14-16] - Coupling metrics**

For Ft4,Ft5 need to investigate

---

The script running in the main branch

With the help of pyGit I could automate this the ballerina repository was cloned locally then this extraction happening in the local

This takes too much time


```
2025-06-24 10:31:36.666 | INFO | __main__:<module>:1 - --- Starting Feature Extraction Process ---
2025-06-24 10:31:36.666 | SUCCESS | __main__:<module>:6 - Git repository initialized successfully.
2025-06-24 10:31:36.673 | INFO | __main__:<module>:12 - Fetching all commits from branch 'master'...
Exception ignored in: <function tqdm.__del__ at 0x00000195EC267790>
Traceback (most recent call last):
  File "c:\Users\pradishan\code\wso2-AI-Tool\.venv\lib\site-packages\tqdm\std.py", line 1148, in __del__
    self.close()
  File "c:\Users\pradishan\code\wso2-AI-Tool\.venv\lib\site-packages\tqdm\notebook.py", line 279, in close
    self.disp(bar_style='danger', check_delay=False)
AttributeError: 'tqdm_notebook' object has no attribute 'disp'
2025-06-24 10:31:40.481 | INFO | __main__:<module>:14 - Found 126808 commits to analyze.
Extracting Features: 73%|██████████ | 92349/126808 [29:44:13<6:19:26, 1.51it/s]
```

It's been a day still running in the mean time I doing the [literature analysis](#) with the help of geminai

Then I found a latest paper [Neural SZZ Algorithm](#) reading it

Shell we have a chat tomorrow

---

The initial version of the feature extraction successfully done  commit\_features.csv


In this version

- Ft1 Lines of code added
- Ft2 Lines of code deleted
- Ft3 Files churned / Number of file
- Ft6 Number of modified sub-directories
- Ft7 Entropy (spreading of changes) were extracted

Now processing 2nd version

---

Insight about literature analysis

 AI-powered risk assessment before merging code

---

## Extracted Features and Assumptions

Feature extraction focuses solely on the **master** branch of this repository.

[Feature Extraction for the statistical model](#) <= ful detailed document

 final\_training\_dataset.csv the feature extracted data set

---

## Labeling

With the help of the SZZ analysis output I labeled the `final_training_dataset.csv`  
The final data set is `final_labeled_training_dataset.csv`

---

## Model training

According to the research paper, I trained a random forest model and, based on Srinath Perera suggestion, I also experimented with the XGBoost model.

Using ParameterGrid, I performed hyperparameter tuning to evaluate model performance and feature importance. The research experiments included applying the models with various sampling techniques: **baseline**, **SMOTE**, and **SMOTE+Tomek**.

[statistical feature based model performance](#) detailed experiments

## Prepare dataset for codeBERT embeddings