LABORATORY SESSION 1

INTRODUCTION

The aim of the laboratory session was to provide gain a hands-on experience on how different algorithms like Myers and Histogram behave on real-world open source repositories. It also made us explore difference between code artifacts and non-code artifacts in the context of software development. It also aims to make us understand the difference in comparisons when the code ignores whitespace and blank lines. Importantly, how discrepancies can highlight the strengths of diff algorithms. After the lab, we aim to be proficient in analyzing diff outputs, generate datasets, visualize mismatches using python plots.

SETUP

- **Software and Tools Used:**
  - Operating System: macOS
  - Text Editor: Visual Studio Code
  - Version Control: git
  - Remote Hosting: GitHub
  - Continuous Integration: GitHub Actions
- **Repositories:**
  - `butterknife` – A popular view binding library for Android that helps reduce boilerplate code when working with UI components.
  - `retrofit` – A type-safe HTTP client for Android and Java, widely used for making API requests and handling network operations.
  - `glide` – An image loading and caching library for Android, optimized for smooth scrolling and efficient resource usage.
- **Python Libraries:**
  - Pydriller: A Python framework for mining software repositories. Used to analyze git repositories and extract commit information.

- Pandas: To manage dataframes and dealing with CSV files

- **Initial Steps:**
  - GitHub account: TejasLohia21
  - SSH key configured for secure push/pull
  - Virtual Environment named lab4 to manage library versions
  - Configured git username and email
  - Cloned/initialized three repository

METHODOLOGY AND EXECUTION

Part A: Repository selection

- `butterknife` – A popular view binding library for Android that helps reduce boilerplate code when working with UI components.
- `retrofit` – A type-safe HTTP client for Android and Java, widely used for making API requests and handling network operations.
- `glide` – An image loading and caching library for Android, optimized for smooth scrolling and efficient resource usage.

Part B: Adding README File and Pushing Changes

Repositories were chosen using the criterias which were mentioned in lectures and in the assignment:

- `butterknife` – A popular view binding library for Android that helps reduce boilerplate code when working with UI components.
  - **Commits:** The repository has 1016 commits, which is within the acceptable range for analysis.
  - **Stars:** The repository has 28.8k stars, indicating strong community usage and real-world relevance.
  - **Myers Algorithm:** Since Git uses Myers as the default diff algorithm, we can directly apply `git diff -diff-algorithm=myers` on Butterknife.
  - **Histogram Algorithm:** Git also supports the Histogram algorithm, which works with the Butterknife repository using `git diff -diff-algorithm=histogram`.

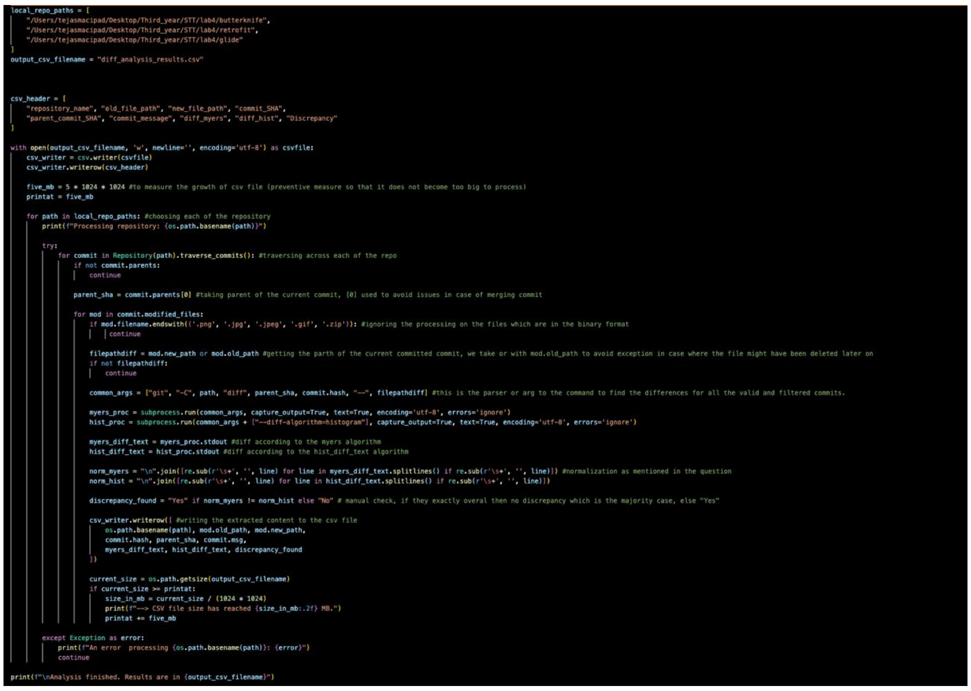
  - **Discrepancy Analysis:** With over a thousand commits and frequent UI-related code changes, Butterknife provides meaningful opportunities to compare Myers and Histogram diffs.
  - **Required Files:** The repository includes `butterknife/` source code, a dedicated `tests/` suite, along with `README.md` and `LICENSE`, thus satisfying all the required file-type conditions.
  - **Conclusion:** Butterknife is one of the academically researched repositories for discrepancy analysis in both the Myers and Histogram algorithms.

- `requests` – A popular Python library for making HTTP requests in a simple and human-friendly way.
  - **Commits:** The repository has 6372 commits, which is in the range of [1206, 25000].
  - **Stars:** The repository has 53.2k stars, showing that it is widely used and well-established.
  - **Myers Algorithm:** Since Git uses Myers as the default diff algorithm, we can directly apply `git diff -diff-algorithm=myers` on the Requests repository.
  - **Histogram Algorithm:** Git also supports the Histogram algorithm, which works with this repository using `git diff -diff-algorithm=histogram`.
  - **Discrepancy Analysis:** With thousands of commits and frequent code updates, the Requests repository provides sufficient changes to compare Myers and Histogram diffs effectively.
  - **Required Files:** The repository contains the `requests/` source directory, a large `tests/` suite, along with `README.md` and `LICENSE`, fulfilling the file-type requirements.

- `glide` – A fast and efficient image loading and caching library for Android, widely used for handling media in mobile applications.
  - **Commits:** The repository has 3047 commits, which lies in the range of [1206, 25000], making it suitable for analysis.
  - **Stars:** The repository has 34.9k stars, showing its strong adoption and relevance in real-world projects.
  - **Myers Algorithm:** Since Git uses Myers as the default diff algorithm, we can directly apply `git diff -diff-algorithm=myers` on the Glide repository.
  - **Histogram Algorithm:** Git also supports the Histogram algorithm, which works seamlessly with this repository using `git diff -diff-algorithm=histogram`.

- **Discrepancy Analysis:** With over three thousand commits and consistent updates, the Glide repository provides enough modifications to effectively compare Myers and Histogram diffs.
- **Required Files:** The repository includes the `glide/` source code, a dedicated `tests/` suite, along with `README.md` and `LICENSE`, satisfying the file-type requirements.

Part C: Run Software Tool on the Selected Repository:



```python
local_repo_paths = [
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/butterknife",
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/retrofit",
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/glide"
]
output_csv_filename = "diff_analysis_results.csv"


csv_header = [
    "repository_name", "old_file_path", "new_file_path", "commit_SHA",
    "parent_commit_SHA", "commit_message", "diff_myers", "diff_hist", "Discrepancy"
]

with open(output_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(csv_header)

    five_mb = 5 * 1024 * 1024 #to measure the growth of csv file (preventive measure so that it does not become too big to process)
    printat = five_mb

    for path in local_repo_paths: #choosing each of the repository
        print(f"Processing repository: {os.path.basename(path)}")

        try:
            for commit in Repository(path).traverse_commits(): #traversing across each of the repo
                if not commit.parents:
                    continue

                parent_sha = commit.parents[0] #taking parent of the current commit, [0] used to avoid issues in case of merging commit

                for mod in commit.modified_files:
                    if mod.filename.endswith(('.png', '.jpg', '.jpeg', '.gif', '.zip')): #ignoring the processing on the files which are in the binary format
                        continue

                    filepathdiff = mod.new_path or mod.old_path #getting the parth of the current committed commit, we take or with mod.old_path to avoid exception in case where the file might have been deleted later on
                    if not filepathdiff:
                        continue

                    common_args = ["git", "-C", path, "diff", parent_sha, commit.hash, "--", filepathdiff] #this is the parser or arg to the command to find the differences for all the valid and filtered commits.

                    myers_proc = subprocess.run(common_args, capture_output=True, text=True, encoding='utf-8', errors='ignore')
                    hist_proc = subprocess.run(common_args + ["--diff-algorithm=histogram"], capture_output=True, text=True, encoding='utf-8', errors='ignore')

                    myers_diff_text = myers_proc.stdout #diff according to the myers algorithm
                    hist_diff_text = hist_proc.stdout #diff according to the hist_diff_text algorithm

                    norm_myers = "\n".join([re.sub(r'\s+', '', line) for line in myers_diff_text.splitlines() if re.sub(r'\s+', '', line)]) #normalization as mentioned in the question
                    norm_hist = "\n".join([re.sub(r'\s+', '', line) for line in hist_diff_text.splitlines() if re.sub(r'\s+', '', line)])

                    discrepancy_found = "Yes" if norm_myers != norm_hist else "No" # manual check, if they exactly overal then no discrepancy which is the majority case, else "Yes"

                    csv_writer.writerow([ #writing the extracted content to the csv file
                        os.path.basename(path), mod.old_path, mod.new_path,
                        commit.hash, parent_sha, commit.msg,
                        myers_diff_text, hist_diff_text, discrepancy_found
                    ])

                    current_size = os.path.getsize(output_csv_filename)
                    if current_size >= printat:
                        size_in_mb = current_size / (1024 * 1024)
                        print(f"--> CSV file size has reached {size_in_mb:.2f} MB.")
                        printat += five_mb

        except Exception as error:
            print(f"An error  processing {os.path.basename(path)}: {error}")
            continue

print(f"\nAnalysis finished. Results are in {output_csv_filename}")
```

Figure 1: Central Code to extract diff.

Listing 1: Diff analysis script for selected repositories

```python
import csv
import re
import os
import subprocess
from pydriller import Repository
from git import Repo

local_repo_paths = [
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/butterknife",
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/retrofit",
    "/Users/tejasmacipad/Desktop/Third_year/STT/lab4/glide"
]
output_csv_filename = "diff_analysis_results.csv"

csv_header = [
    "repository_name", "old_file_path", "new_file_path", "commit_SHA",
    "parent_commit_SHA", "commit_message", "diff_myers", "diff_hist", "Discrepancy"
]
```

```python

with open(output_csv_filename, 'w', newline='', encoding='utf-8') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(csv_header)

    five_mb = 5 * 1024 * 1024
    printat = five_mb

    for path in local_repo_paths:
        print(f"Processing repository: {os.path.basename(path)}")

        try:
            for commit in Repository(path).traverse_commits():
                if not commit.parents:
                    continue

                parent_sha = commit.parents[0]

                for mod in commit.modified_files:
                    if mod.filename.endswith(('.png', '.jpg', '.jpeg', '.gif', '.zip')):
                        continue

                    filepathdiff = mod.new_path or mod.old_path
                    if not filepathdiff:
                        continue

                    common_args = ["git", "-C", path, "diff", parent_sha, commit.hash, "--", filepathdiff]

                    myers_proc = subprocess.run(common_args, capture_output=True, text=True, encoding='utf-8', errors='ignore')
                    hist_proc = subprocess.run(common_args + ["--diff-algorithm=histogram"], capture_output=True, text=True, encoding='utf-8', errors='ignore')

                    myers_diff_text = myers_proc.stdout
                    hist_diff_text = hist_proc.stdout

                    norm_myers = "\n".join([re.sub(r'\s+', '', line) for line in myers_diff_text.splitlines() if re.sub(r'\s+', '', line)])
                    norm_hist = "\n".join([re.sub(r'\s+', '', line) for line in hist_diff_text.splitlines() if re.sub(r'\s+', '', line)])

                    discrepancy_found = "Yes" if norm_myers != norm_hist else "No"

                    csv_writer.writerow([
                        os.path.basename(path), mod.old_path, mod.new_path,
                        commit.hash, parent_sha, commit.msg,
                        myers_diff_text, hist_diff_text, discrepancy_found
                    ])

                    current_size = os.path.getsize(output_csv_filename)
                    if current_size >= printat:
                        size_in_mb = current_size / (1024 * 1024)
                        print(f"--> CSV file size has reached {size_in_mb:.2f} MB.")
                        printat += five_mb

        except Exception as error:
            print(f"An error  processing {os.path.basename(path)}: {error}")
            continue

print(f"\nAnalysis finished. Results are in {output_csv_filename}")
```

Explanation of the Code

The code shown in Figure 2 is responsible for extracting the differences between file versions in the selected repositories. It leverages the Pydriller library to iterate through each commit and retrieve the modified files. For each file, the code applies both the Myers and Histogram diff algorithms using Git commands, capturing the output for further analysis.

Key steps in the code:

- Store all the paths to the repositories in `local_repo_paths`.
- Define the output csv_filename and also define the csv_header for that csv file which contains all the columns asked in the part (c) of the assignment.
- Algorithm then starts iterating through each repository path, ans also tracks the size of growing csv file.
- Iterating through every commit of that repository which initially fetches the hash value of the commmit.
- Extraction of parent_sha (We only take the first parent in case of a merge commit). Following this, we iterate through each of the modified files in that commit.
- Defining `common_args` which stores commands to be passed on to the `subprocess.run()` as arguments.
- We obtain the Myers diff and Histogram diff outputs by executing the respective Git commands and capturing their output and apply normalization
- Defining discrepancy_found if the outputs of the two algorithms differ and commit the output to the CSV file

This approach allows for systematic extraction and comparison of diff outputs, facilitating the analysis of discrepancies between the Myers and Histogram algorithms across real-world code changes.

Part D: Compare Diff Outputs for Discrepancy Analysis

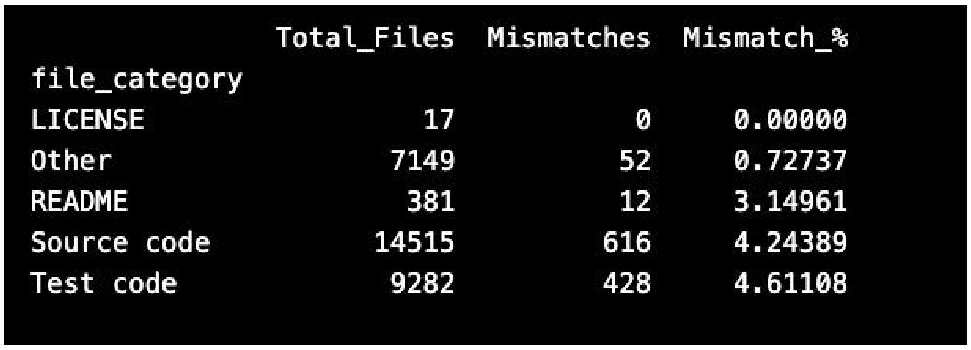
After processing all the repositories we record the output of both the algorithms (Myers and Histogram) in a CSV file.

1. **Normalization:** To ensure fairness in comparison, whitespace and blank lines were ignored.
2. **Comparison Logic:** The normalized Myers output and Histogram output were compared line by line.

- If both were identical, the entry was labeled `No` discrepancy.
- If they differed, the entry was labeled `Yes` discrepancy.

3. **CSV Output:** A dedicated column named `Discrepancy` was added to the CSV file. This column stores either `Yes` or `No` for each commit-file pair.

Part E: Final Dataset Statistics and Exploratory Analysis



```
              Total_Files  Mismatches  Mismatch_%
file_category
LICENSE                17           0     0.00000
Other                7149          52     0.72737
README                381          12     3.14961
Source code         14515         616     4.24389
Test code            9282         428     4.61108
```

Figure 2: Central Code to extract diff.

After collecting and storing the diff results from all three repositories, we performed an exploratory analysis of the final dataset. The analysis focused on quantifying discrepancies and categorizing them based on file types.

1. **File Categorization:** Each modified file was categorized into one of the following groups: *Source code*, *Test code*, *README*, *LICENSE*, or *Other*. This categorization was performed by checking the file extensions and naming patterns (e.g., files containing "test" were labeled as Test code).

2. **Mismatch Detection by Category:** For each category, the total number of files and the number of files with discrepancies between the Myers and Histogram algorithms were computed. This enabled us to analyze which file types were more prone to mismatches.

3. **Statistical Summary:** The following statistics were derived from the analysis:
   - Total files analyzed across all repositories: **31,344**.
   - Overall mismatches detected: **1,108**, corresponding to approximately **3.54%** of the total files.
   - Breakdown of mismatches by file category:
     - **Source code:** 14,515 files, 616 mismatches (**4.24%**).
     - **Test code:** 9,282 files, 428 mismatches (**4.61%**).
     - **README files:** 381 files, 12 mismatches (**3.15%**).

- **LICENSE files:** 17 files, 0 mismatches (**0%**).
- **Other files:** 7,149 files, 52 mismatches (**0.73%**).

- **File Extension Summary (Top 10):**
  - **.java:** 23,281 files, 1,040 mismatches (4.47%)
  - **.kt:** 350 files, 14 mismatches (4.00%)
  - **.md:** 541 files, 15 mismatches (2.77%)
  - **Other:** 142 files, 3 mismatches (2.11%)
  - **.json:** 70 files, 1 mismatch (1.43%)
  - **.gradle:** 1,410 files, 14 mismatches (0.99%)
  - **.yml:** 150 files, 1 mismatch (0.67%)
  - **.iml:** 152 files, 1 mismatch (0.66%)
  - **.xml:** 2,773 files, 15 mismatches (0.54%)
  - **.properties:** 587 files, 2 mismatches (0.34%)

4. **Top 10 Files Causing Mismatches:**
   - `library/src/main/java/com/bumptech/glide/load/resource/bitmap/Downsampler.java` – 15 mismatches
   - `README.md` – 12 mismatches
   - `library/src/com/bumptech/glide/Glide.java` – 12 mismatches
   - `butterknife-compiler/src/main/java/butterknife/compiler/BindingClass.java` – 11 mismatches
   - `library/src/main/java/com/bumptech/glide/load/engine/DecodeJob.java` – 9 mismatches
   - `library/src/main/java/com/bumptech/glide/request/target/ViewTarget.java` – 8 mismatches
   - `library/src/main/java/com/bumptech/glide/load/engine/EngineJob.java` – 7 mismatches
   - `library/src/main/java/com/bumptech/glide/RequestBuilder.java` – 7 mismatches
   - `butterknife-compiler/src/main/java/butterknife/compiler/ButterKnifeProcessor.jav` – 6 mismatches

- `samples/giphy/src/main/java/com/bumptech/glide/samples/giphy/MainActivity.java` – 6 mismatches

5. **Commit-Level Summary:**
   - **Average mismatches per commit:** 0.03
   - **Top 10 commits with most mismatches:**
     - f389e91ccecac6ddf736bbf1a4346782609eb034 - 229 mismatches
     - f7a6d65cf7c1a41908dd48e0dab68ee5b881387e - 49 mismatches
     - 8f8a1600826fd042abae9251cbad063dee5144b2 - 36 mismatches
     - c375a2fbf594bdf422c45a1395a65823141a8bd5 - 35 mismatches
     - 0dcc33fe6957657b910484599c499430cdf3461c - 32 mismatches
     - ee71a6858dfddcc4650f6ff4d71112911bdf6ca7 - 31 mismatches
     - 6f1d71715361d68384afa0cf5fc9ad0e898b3697 - 18 mismatches
     - ed20643fb94d4e17f4cdb3699a6d83621408dd34 - 15 mismatches
     - 91cb86225967b1e12cdab52d2d3ea35afc2b8c9e - 12 mismatches
     - eac2c5f7190d148e72341ea289b69cae1ae2a866 - 12 mismatches
6. **Repository-Level Summary:**
   - **glide:** 20,105 files, 820 mismatches (4.08%)
   - **butterknife:** 3,120 files, 109 mismatches (3.49%)
   - **retrofit:** 8,119 files, 179 mismatches (2.20%)
7. **Visualization:** To better understand the dataset, we generated multiple plots summarizing files, mismatches, and repository behavior:
   - **Total Files by Category:**
   - **Mismatches by File Category:**
   - **Mismatch Percentage by File Category:**
   - **Top 10 File Extensions by Mismatch Percentage:**
   - **Mismatch Percentage by Repository:**
   - **Distribution of Mismatches per Commit:**
   - **Top 10 Commits by Number of Mismatches:**

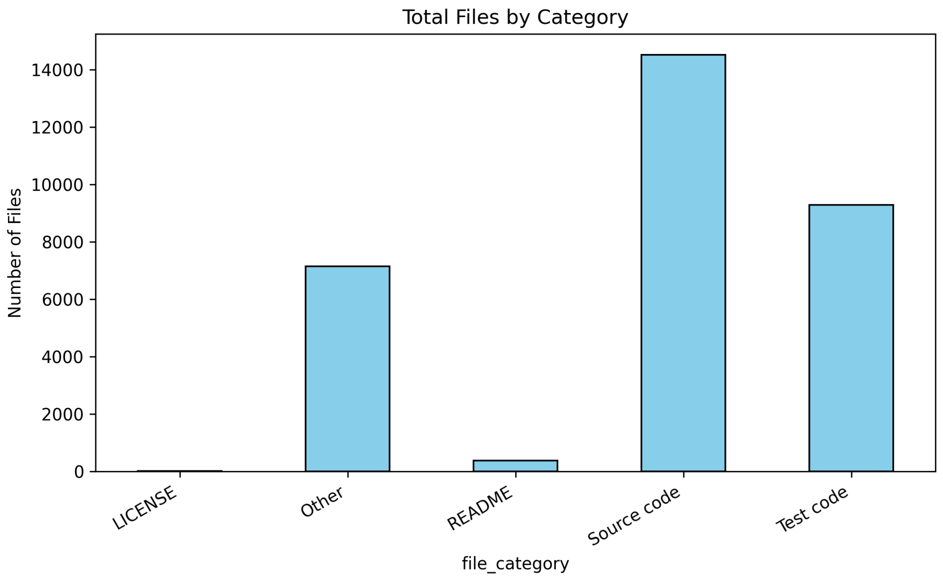


Figure 3: Bar chart showing the total number of files per file category.

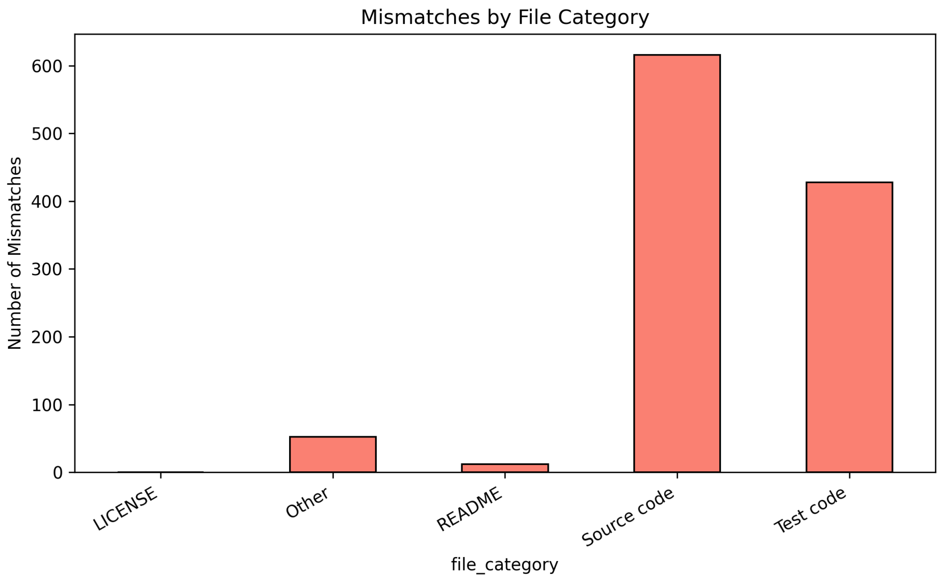


Figure 4: Bar chart showing the number of mismatches for each file category.

These plots and statistics prove an understanding of the dataset's structure and the nature of the mismatches present.

Part F: Which algorithm performed better

- `Metric Definition` – We can define multiple metrics to evaluate the performance of the algorithms, such as number of mismatches, percentage of mismatches, runtime efficiency (Compute required).
- `Run Model to find various metrics` – Execute the models on the dataset to obtain

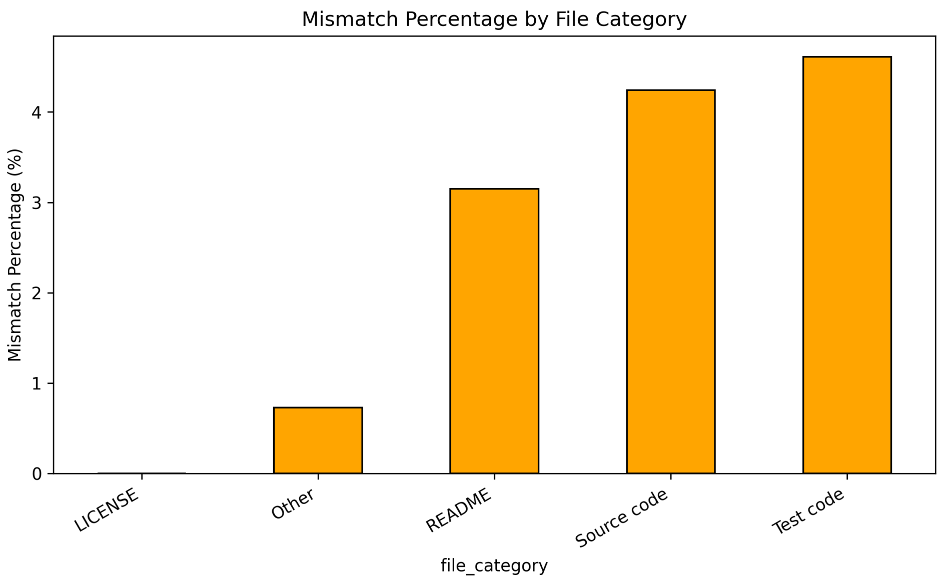


Figure 5: Bar chart displaying the mismatch percentage per file category.

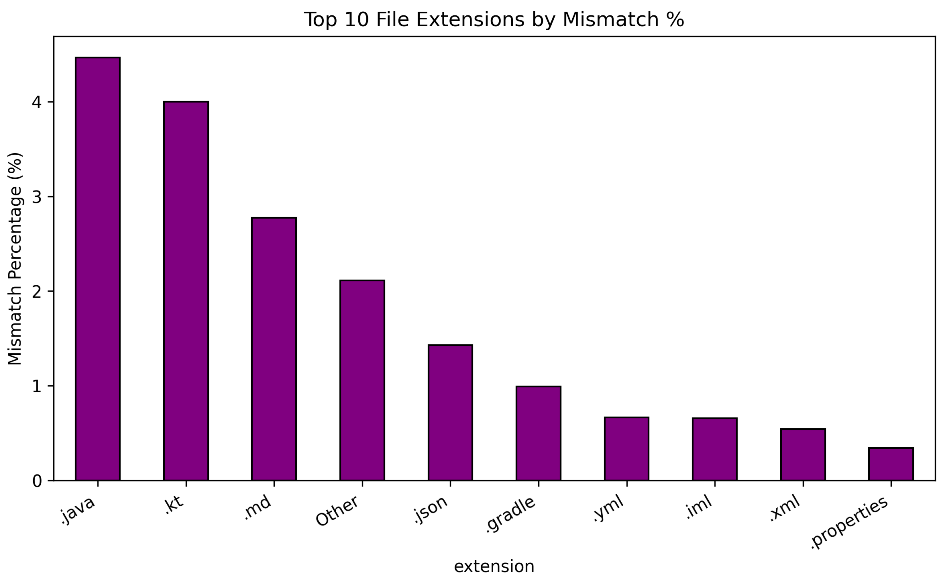


Figure 6: Bar chart highlighting the top 10 file extensions with the highest mismatch percentages.

the defined metrics for each algorithm.

- `Conclusive analysis` – Just based on the results on a single repository, we cannot conclude which algorithm is better. However, based on the results from all three repositories, we can perform Statistical tests to compare the algorithms more rigorously to provedisprove any hypothesis.
- `Script to run these tests` – Automating the script to report the result which includes to run the model to get metrics, run statistical tests(optional), and report the one which scores better.

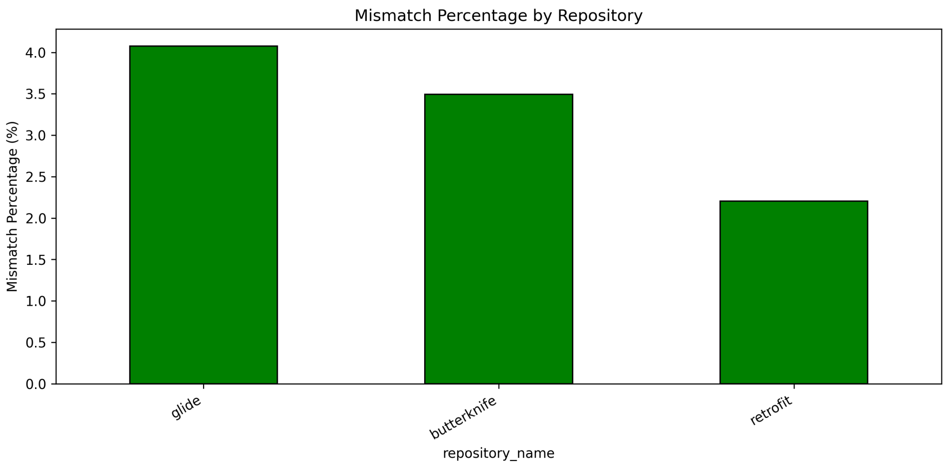


Figure 7: Bar chart showing mismatch percentages across repositories.

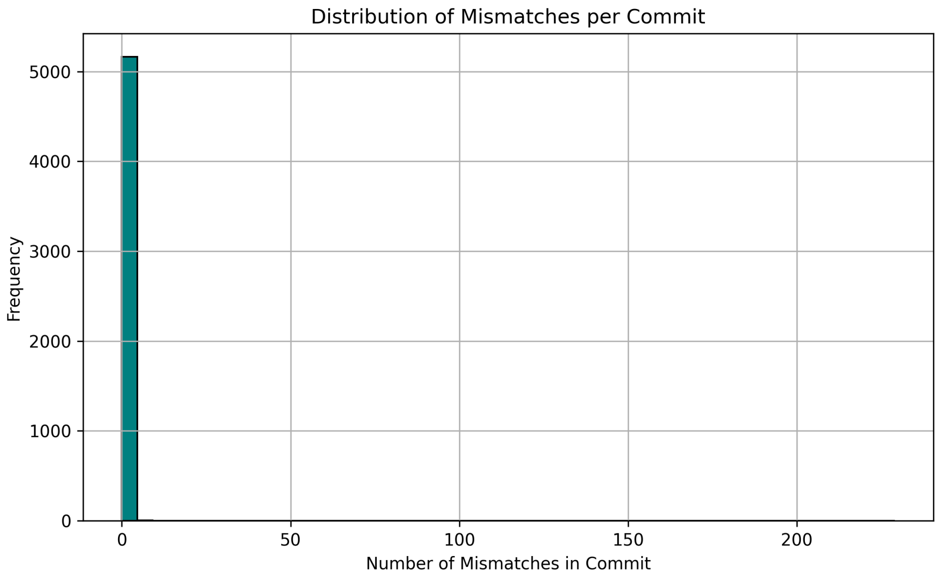


Figure 8: Histogram illustrating the distribution of mismatches per commit.

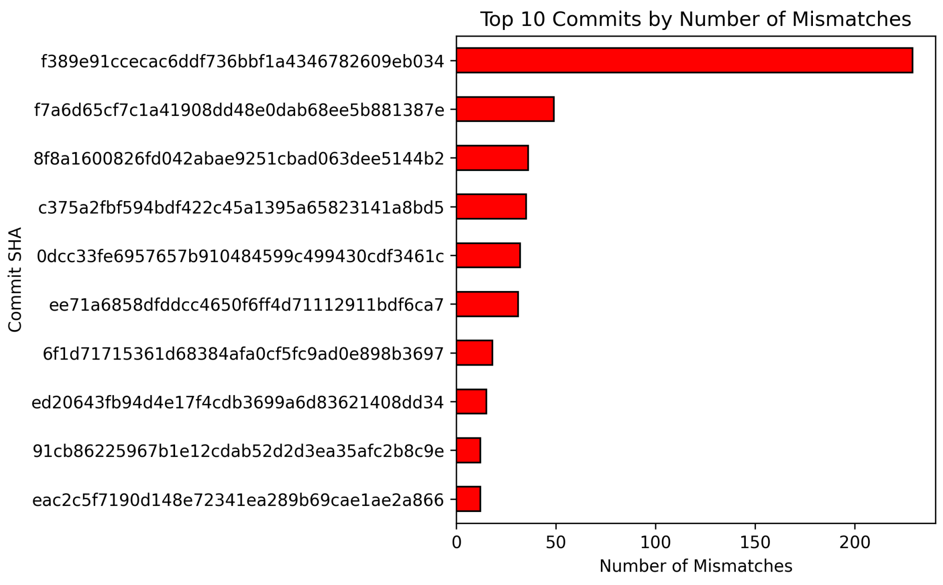


Figure 9: Horizontal bar chart displaying the top 10 commits responsible for the most mismatches.