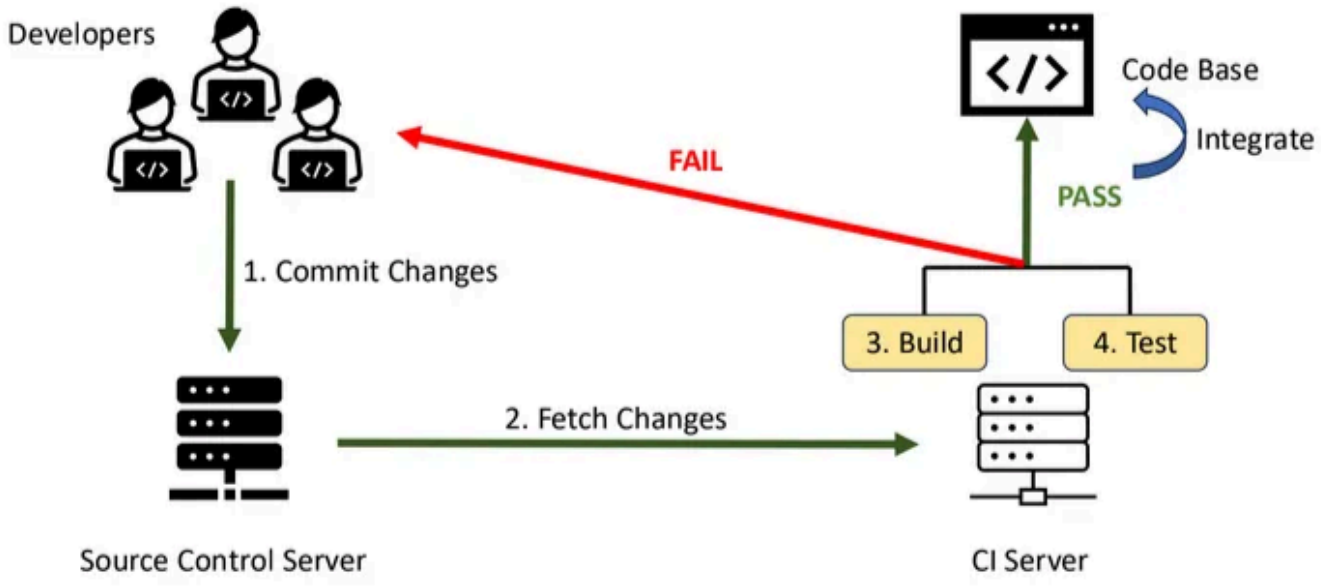
Assessment 1

Introduction

- Version Control System (VCS) : Git
  Version control is a very important part of software development. It allows us to have various snapshots of our code-base for back-up in case any new modifications cause bugs, or remove useful features, or revert progress in any other way. There are many version control systems such as Apache Subversion, Azure DevOps Server, GNU Bazaar, Panvalet, etc.
  Git is a very popular version control system (VCS). It relies on differencing algorithms such as Myers, Minimal, Patience, and Histogram algorithms. It also allows users to interact with remote repositories, enabling continuous integration.
  This helps in improving the speed of the DevOps cycle.



Continuous Integration (Source : Lecture Slides)

- Some features that Git provides
    i. Users can create local git repositories, which function for the sole purpose of version control without GitHub.
    ii. Users can link and clone remote repositories on GitHub, allowing easy code sharing. Then the users can push and pull the code to and from the remote repository.
    iii. Users can create pull requests, and review and merge the same. This allows systematic and incremental development.

  Moreover, GitHub has additional tools, such as deployment of static websites, and workflow systems such as PyLint.
- PyDriller

  PyDriller is a python package that allows mining commits from GitHub. It is used extensively for doing analysis of the project and the codebase. It has a lot of filters and options for querying.
  It can use “myers” and “histogram” algorithm for creation of “diff” as per user’s convenience.
  For every modified file in a commit, PyDriller provides us with this information:
    i. old and new paths of file
    ii. “diff” , which is short for difference between the new and old contents of file
    iii. source code before and after commit
    iv. McCabe’s Cyclomatic Complexity (MCC)

  and more.

Coding effort

This is the number of nodes that occur before a particular (buggy) node in the Abstract Syntax Tree (AST) of a program.

An AST is a parse tree that represents programs and is used for evaluation, or compilation. It is created by parsing the source code according to some unambiguous context free grammar for that language. Python proved the module `ast` for creating AST from python source code. For example, the code in `trial.py` gives the AST printed on the right in text format.

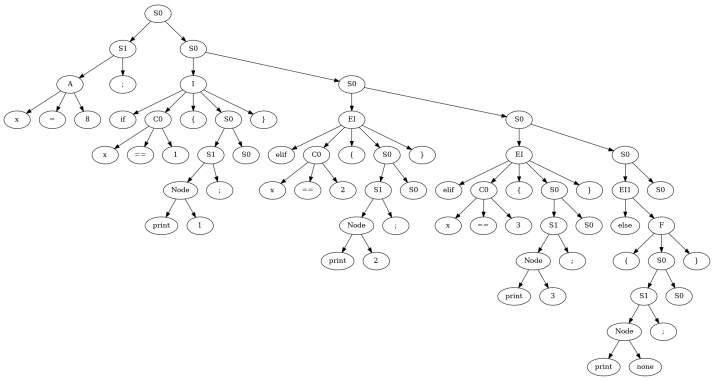
The tree-sitter parser gives the AST for almost all the popular programming languages.

https://tree-sitter.github.io/tree-sitter/

Another more general example for ASTs would be for this custom programming language code :

```
x = 8;
if x ==1 {print 1;}
elif x==2 {print 2;}
elif x==3 {print 3;}
else {print "none";}
```

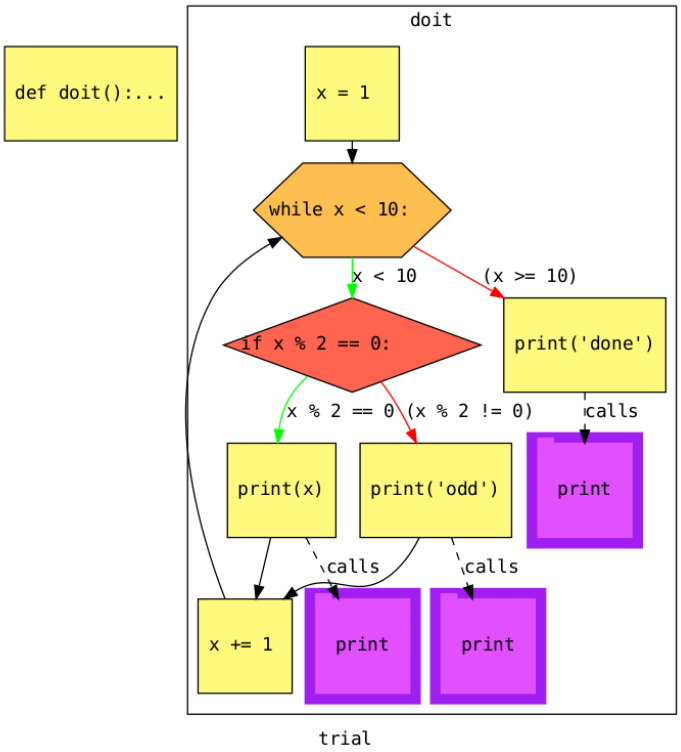
might have an AST like this :




```
Module(body=[
    FunctionDef(
        name='doit',
        args=arguments(
            posonlyargs=[],args=[],kwonlya
            defaults=[]),
        body=[
            Assign(targets=[Name(id='x', c
            While(
                test=Compare(
                    left=Name(id='x', ctx=
                    ops=[Lt()],
                    comparators=[Constant(
                body=[
                    If(
                        test=Compare(
                        left=BinOp(left=Na
                        ops=[Eq()],compara
                        body=[
                            Expr(
                            value=Call(
                            func=Name(id='
                            args=[Name(id=
                            keywords=[]))]
                        orelse=[
                            Expr(
                            value=Call(
                            func=Name(id='
                            args=[Constant
                    AugAssign(
                        target=Name(id='x'
                        op=Add(),value=Con
                orelse=[]),
            Expr(
            value=Call(
            func=Name(id='print', ctx=Load
            args=[Constant(value='done')],
    decorator_list=[])],
type_ignores=[])
```

CFG and McCabe's Cyclomatic Complexity

Tools such as `py2cfg` , `pycfg` and `staticfg` can take in programs and convert these to a Control Flow Graph (CFG).



For example, consider this python script named `trial.py` :

```
def doit():
        x = 1
        while x < 10:
                if x%2 == 0:print(x)
                else:print("odd")
                x += 1
        print("done")
```

The CFG for this script is the image to the left, which was created using `py2cfg` like this :

```
from py2cfg import CFGBuilder
import ast,lizard
def render_as_CFG(code_file="trial.py",gra
        src = open(code_file,'r').read()
        C = CFGBuilder().build(graph_file,
        g = C._build_visual()
        g.node_attr["width"] = g.node_attr
        g.render(graph_file,view=True)
        return (lizard.analyze_file.analyz
        .function_list[0].__dict__["cyclom
print("complexity is",render_as_CFG())
```

The cyclomatic complexity is defined as the number of different ways that a program might execute.

For this CFG, it would be 3, as per the package `lizard` . There are two answers actually. The first is $1 \times (2^9 + 1)$ since the loop executes 9 times, and checks the condition `x < 10` (which has complexity 1), 10 times. The body of the loop contains an `if-else` clause and thus has complexity of 2. Finally, when the condition `x<10` is untrue, it does to the `print("done")` with complexity 1.

The second answer assumes that every loop executes only once. Thus, it gives $1 \times (2^1 + 1) = 3$ as the answer. In general, it gives the number of regions that the CFG divides the 2D space into (including the outer region) . This number can easily be calculated using Euler's formula as $|E| - |V| + 2p$ where p is the number of disconnected components, and $G = (V, E)$ is the CFG as a directed graph.

- Programming bugs

  It is essential to remove bugs as early as possible, since the more the code develops, the harder it becomes to pin-point the bug.

  There are several types of bugs. Some major distinctions are: Functional bug (improper behavior of the system), Visual bugs (distortions in the graphics), Syntactic bugs (spelling mistakes), Performance bugs (very slow execution due to un-optimized code), Crash bugs (the OS has to step in and kill processes like this).
- Minecraft

  A tool for **automated mining** of **software bug fixes (commit pairs)** with **precise code context (up-to line number)** from GitHub.

  This is a software developed at IITGN that is used for analyzing bug fixes for GitHub repositories. There are tools like this already existing, but none of them are language agnostic, while Minecraft aims to be. It does this using the tree-sitter parser, which can parse almost all prominent languages.

  It uses regular expressions to detect bug-fixing commits, and then the SZZ algorithm to get the bug-introducing commits. Thus, it forms pairs of commits for every bug-fix. It also mines the context of the bug-fix (3 lines above and below a function, and 5 lines above and below the changed lines).

  This, combined with the bug location (line numbers before and after the bug-fixing commit) , the commit message, and the bug type forms a “precise code context” for the bug.

COMPARISON OF MINECRAFT WITH EXISTING REPOSITORY MINED DATASETS

	FixJs [3]	*Tufano et al.* [4]	*Minecraft* (this paper)
Introduced in	MSR 2022	TOSEM 2019	This paper
Language	JavaScript	Java	C, C++, Python, and Java
Granularity	Function	Function	File, Function, and Statement (ranges)
#Commits	~2M	~787K	~2.2M
#Bug-Fixes	300K	~2.3M	~3.29M
Dataset size	5.47GB	~7GB	28.8GB
Bug Detection Strategy	Commit messages with 6 keywords [5]: [“fix”, “solve”, “bug”, “issue”, “problem”, “error”]	Commit messages with 6 keywords [5]: [“fix”, “solve”, “bug” “issue”, “problem”, “error”]	Commit messages with 52 keywords [6] (*c.f.* Sec. III-C) + RegEx. [7]: `'.((solv(ed|es|e|ing)) |(fix(s|es|ing|ed)?)' |((error|bug|issue)(s)?)).'` + SZZ Algorithm [8]

Source : Minecraft: Automated Mining of Software Bug Fixes (doi:10.1109/ASE56229.2023.00116)

Minecraft also has a database of repositories that have been processed by it : https://zenodo.org/records/8164641

- MineCPP/Minecraft++

  Adds to Minecraft features like : Coding effort, Test case indicator, Constructs around the bug, Cyclomatic Complexity and Similarity Score

  Thus, the final schema of the data-set is the one below :

Col#	Name	Type	Description
1	`Before Bug fix`	String	Code + Context of Bug
2	`After Bug fix`	String	Code + Context of after Bug fix
3	`Location`	List of ints.	Line #s of the bug and its fix
4	`Bug type`	String	Bug type deduced using `git diff` and LLM
5	`Commit Message`	String	Author's description about commit
6	`Project URL`	String	GitHub link of the repository
7	`File Path`	String	The path of file with a change or bug
8	`Fixed Commit`	String	git hash of `fixed` commit
9	`Buggy Commit`	String	git hash of `Buggy` commit
10	`Test File`	Bool	has test case or not
11	`Coding Effort`	Int	explained in the CodingEffort function
12	`Constructs`	Dict	type of constructs in which bug is present
13	`Lizard Features Buggy`	Dict	Cyclomatic Complexity of buggy code
14	`Lizard Features Fixed`	Dict	Cyclomatic Complexity of fixed code
15	`BLEU`	Float	text similarity score
16	`crystalBLEU_score`	Float	code similarity score
17	`BERT_score`	Float	code similarity score

  Source: Lecture Slides

  - Similarity Score

    This is a quantitative score for similarity between buggy and fixed code snippets (using the context of the code).

    Popular similarity score metrics are BLEU, CrystalBLEU and Code BERT score
  - Test case indicator

    This is an indicator function, which is 1 if the bug-fix pair has a test case and 0 otherwise.

    Test cases are found by navigating through the repository files and matching the file name to regex.
  - Constructs surrounding the bug

    These are basically nodes in the AST which are part of the context of the bug. For example.. “module” , “import_statement” , “import”, “dotted_name” ,etc. By finding frequencies of these constructs, one can determine the kind of setting in which this bug was introduced.

Lab 1

- Most of the commands that I used are from https://education.github.com/git-cheat-sheet-education.pdf and some are from miscellaneous internet sources.
- Configuring Git
  - Configuring Git with my name and email :

```
~$ git config --global user.name "[Pranav Joshi]"
~$ git config --global user.email "[pranav.joshi@iitgn.ac.in]"
~$
```

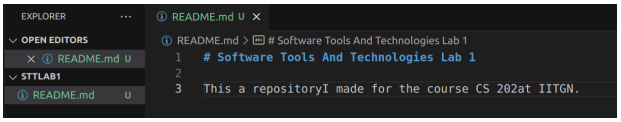
  - Verifying my git configuration (stored in the .config file)

```
~$ git config --list
user.email=[pranav.joshi@iitgn.ac.in]
user.name=[Pranav Joshi]
credential.https://github.com.helper=
credential.https://github.com.helper=!/usr/bin/gh auth git-credential
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/pranav-joshi-iitgn/ATS-Resume-Scanner-and-Scorer.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
~$
```

- Creation of Repositories
  i. Making a directory with name `STTLab1` and initializing it as a local git repository.

```
~$ mkdir STTLab1
~$ git init STTLab1
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/hp/STTLab1/.git/
~$
```

  ii. Making a `README.md` file



  iii. Adding the `README.md` file to the staging area

```
~$ cd STTLab1/
~/STTLab1$ git add README.md
~/STTLab1$
```

  iv. Committing the file with a message

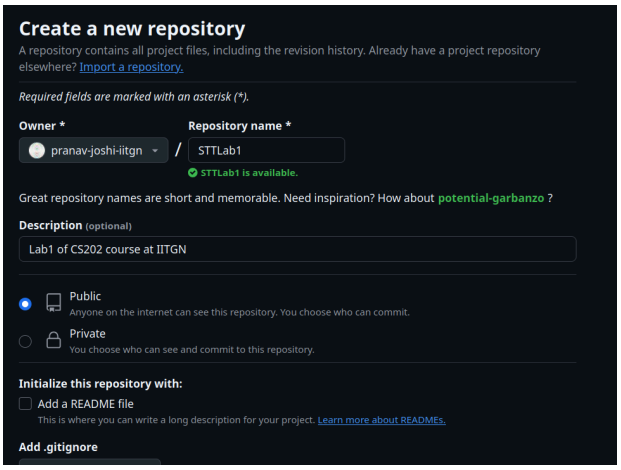
```
~/STTLab1$ git commit -m "Added README.md"
[master (root-commit) 79b63ec] Added README.md
 1 file changed, 3 insertions(+)
 create mode 100644 README.md
~/STTLab1$
```

  v. Viewing the commit history

```
~/STTLab1$ git log
commit 79b63ec46a9d9593b172426e8e29583447e12aa4 (HEAD -> master)
Author: [Pranav Joshi] <[pranav.joshi@iitgn.ac.in]>
Date:   Thu Jan 9 09:01:44 2025 +0530

    Added README.md
~/STTLab1$
```

  - You can view the repository on GitHub : https://github.com/pranav-joshi-iitgn/STTLab1

- Remote Repositories
  - i. Creating a New repository on GitHub (not locally)



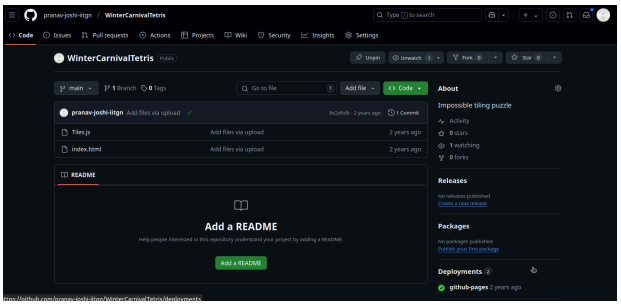
  - ii. Linking the local repository to the remote one

```
~/STTLab1$ git remote add origin https://github.com/pranav-joshi-iitgn/STTLab1.git
~/STTLab1$
```

  - iii. Pushing the committed changes to GitHub

```
~/STTLab1$ git remote -v
origin  https://github.com/pranav-joshi-iitgn/STTLab1.git (fetch)
origin  https://github.com/pranav-joshi-iitgn/STTLab1.git (push)
~/STTLab1$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 319.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/pranav-joshi-iitgn/STTLab1.git
 * [new branch]      master -> master
~/STTLab1$
```

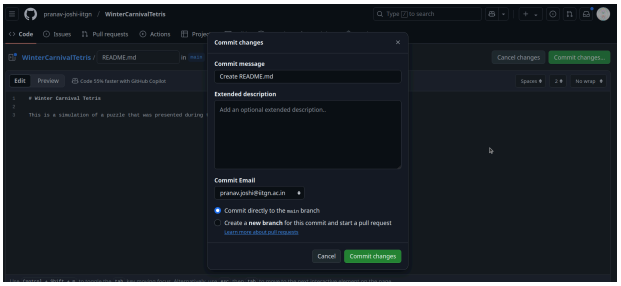
- Updating a cloned repository
  - i. The GitHub repository to be cloned

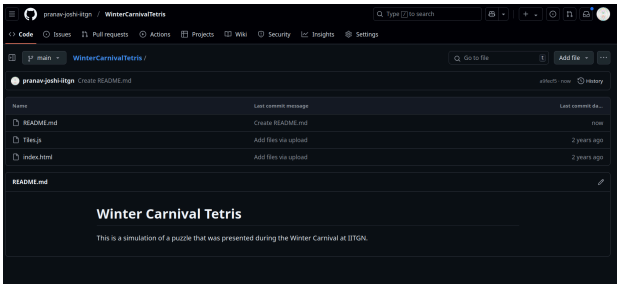


  - ii. Cloning the repository

```
~$ cd Web
~/Web$ git clone https://github.com/pranav-joshi-iitgn/WinterCarnivalTetris.git
Cloning into 'WinterCarnivalTetris'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.
~/Web$
```

  - iii. Changing the GitHub repository

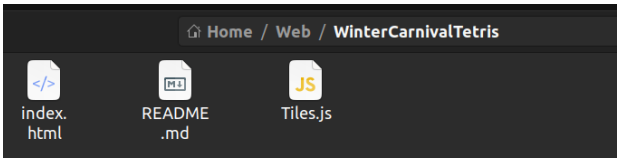




  - iv. Pulling changes

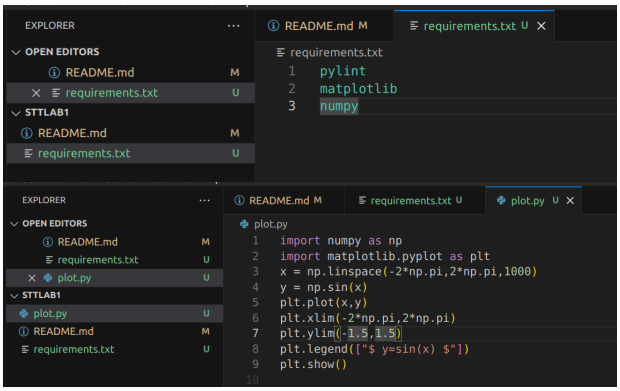
```
~/Web/WinterCarnivalTetris$ git pull origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 1.03 KiB | 1.03 MiB/s, done.
From https://github.com/pranav-joshi-iitgn/WinterCarnivalTetris
 * branch            main       -> FETCH_HEAD
   8e2d6db..a9fecf5  main       -> origin/main
Updating 8e2d6db..a9fecf5
Fast-forward
 README.md | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 README.md
~/Web/WinterCarnivalTetris$
```

  - v. State of local repository after pulling changes

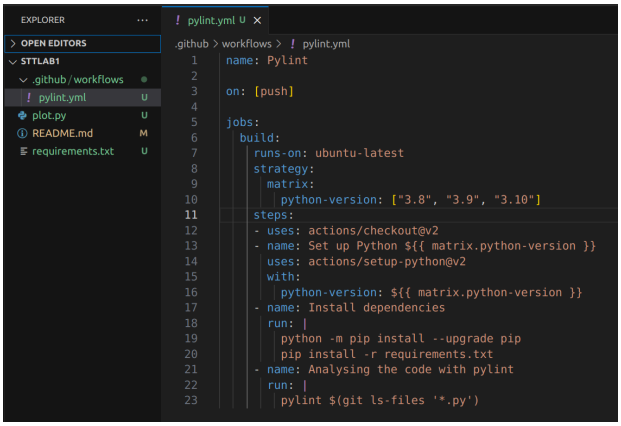


The `README.md` file wasn't there before, but now it is.

- Working with PyLint
  - i. Adding more files to `STTLab1` local repositories



  - ii. Adding YAML file `pylint.yml`



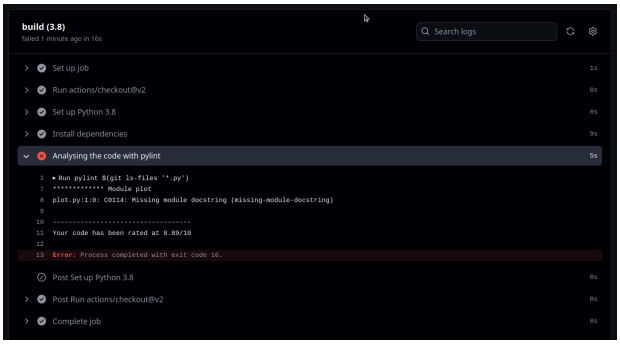
  - iii. Commit

```
~/STTLab1$ git add --all
~/STTLab1$ git commit -m "added pylint"
[master 176e740] added pylint
 4 files changed, 36 insertions(+), 1 deletion(-)
 create mode 100644 .github/workflows/pylint.yml
 create mode 100644 plot.py
 create mode 100644 requirements.txt
```

  - iv. Push

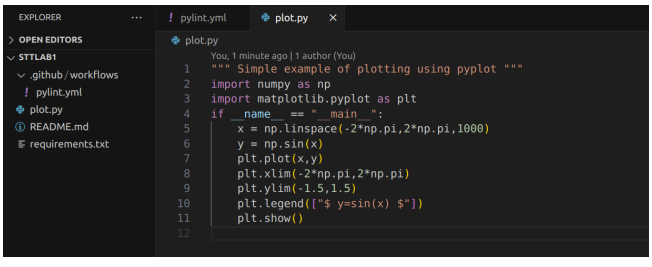
```
~/STTLab1$ git push origin master
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 954 bytes | 954.00 KiB/s, done.
Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/pranav-joshi-iitgn/STTLab1.git
   79b63ec..176e740  master -> master
```

  - v. Working of PyLint on GitHub

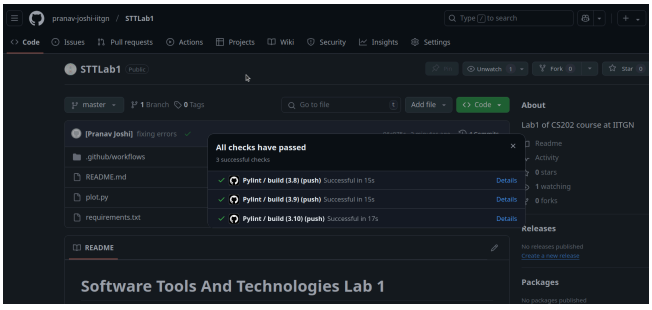


  - Fixing linting errors

    Rewriting code



    Linting errors fixed

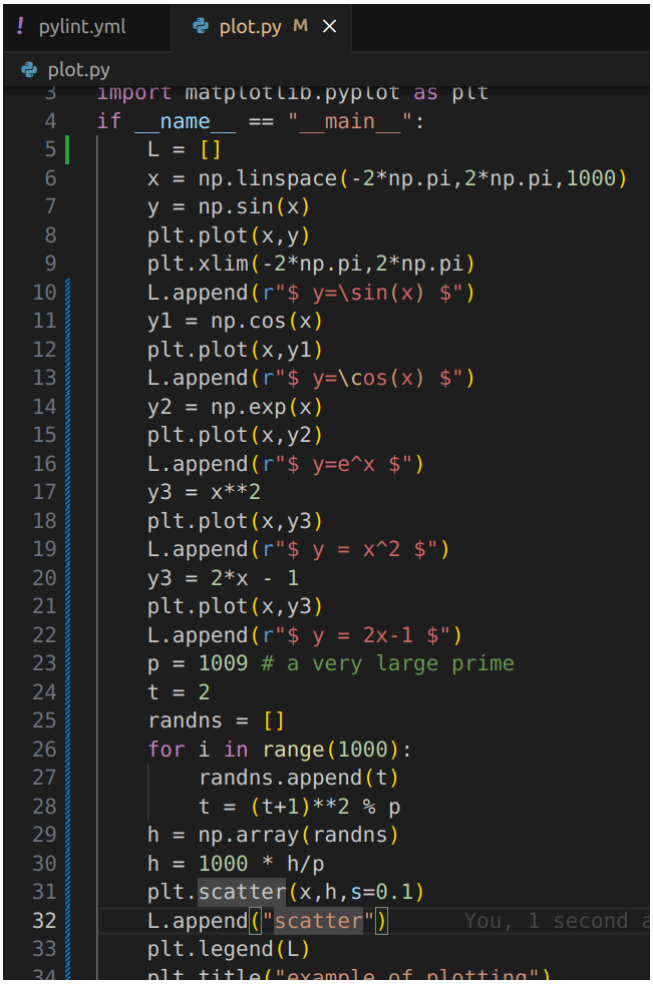


    Commit and Push

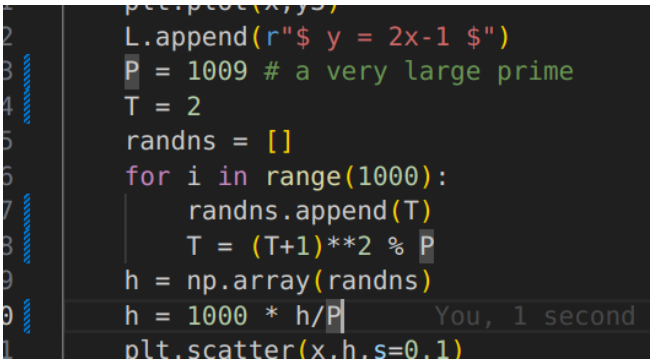
```
~/STTLab1$ git add --all
~/STTLab1$ git commit -m "fixing errors"
[master 698c458] fixing errors
 1 file changed, 9 insertions(+), 7 deletions(-)
~/STTLab1$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 449 bytes | 449.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/pranav-joshi-iitgn/STTLab1.git
   176e740..698c458  master -> master
~/STTLab1$ git add --all
~/STTLab1$ git commit -m "fixing errors"
[master 96c075c] fixing errors
 1 file changed, 1 insertion(+), 1 deletion(-)
~/STTLab1$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 295 bytes | 295.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/pranav-joshi-iitgn/STTLab1.git
   698c458..96c075c  master -> master
~/STTLab1$
```

- Another example

Writing more code



Fixed errors

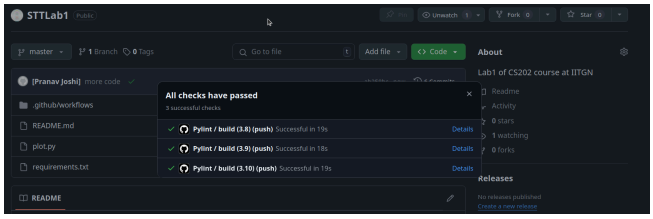


Committing and pushing

```
~/STTLab1$ git add --all
~/STTLab1$ git commit -m "more code"
[master a10b0e6] more code
 1 file changed, 26 insertions(+), 2 deletions(-)
~/STTLab1$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 623 bytes | 623.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/pranav-joshi-iitgn/STTLab1.git
   96c075c..a10b0e6  master -> master
~/STTLab1$
```

Linting Errors

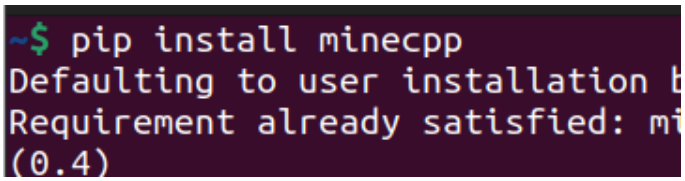
Linting Errors Resolved

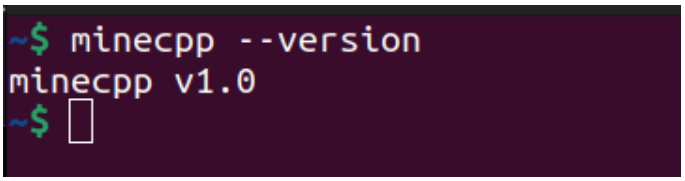


Commit history

Lab 2

- Installing `minecpp`





- Open Source Repository to analyze with `minecpp`

  https://github.com/pallets/flask

  Flask is used to deploy websites. Currently, the GitHub repository has 68.6K stars.

  My selection criterion were:

  - Must be written in Python
  - Must have more than 50K stars
  - Must have less than 50K commits
  - Must have more than 5K commits

- Running `minecpp` on the Flask repository

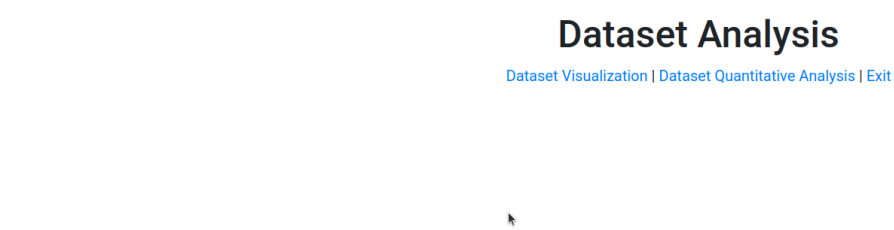
```
$ minecpp -u https://github.com/pallets/flask.git
/home/hp/.local/lib/python3.10/site-packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0
. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
[nltk_data] Downloading package punkt to /home/hp/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Processing for project:  flask.git
Getting fixed commits...
Repo:  /home/hp/.local/lib/python3.10/site-packages/Minecpp/flask.git/flask.git
Total commits:  5391
Processing SZZ
Processing Commits:  62%|                    | 1046/1676 [00:00<?, ?it/s]Processing for commit:  701faf0724d8aa18d7ed2907b63c143725dd3e7d
Corresponding commit msg:  Merge pull request #2320 from markshannon/fix-click-dependency-info
Processing for commit:  c03a82713a8fe4d256a2d2cf1b1f3ab36d5a5916
Corresponding commit msg:  Cleanup test_blueprint.py to use test fixtures
/home/hp/.local/lib/python3.10/site-packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0
. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
/home/hp/.local/lib/python3.10/site-packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0
. Downloads always resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Processing Commits:  63%|                     | 1048/1676 [01:08<5:57:45, 34.18s/it]Processing for commit:  849fc4b90c009d9bbbb604b2997d18340fb7ec2d
Corresponding commit msg:  Merge pull request #2323 from dawran6/test-fixture
Processing for commit:  c62b614d9cbe813d7803c8ecfea2b7bf5a6eaab9
Corresponding commit msg:  Merge pull request #2324 from rzelayafavila/2313-refactor-gevent-tests-into-class
Processing for commit:  65b22926f7673b419a9da82f3213d7243c968af1
Corresponding commit msg:  reduce number of tox and travis envs
Processing Commits:  63%|                     | 1051/1676 [01:10<1:58:26, 11.37s/it]Processing for commit:  471c7f3220f45f60a56515f9c1dc55c794b6ff36
Corresponding commit msg:  Merge pull request #2331 from davidism/less-travis
Processing for commit:  7e14f6706aa1e1edd95b2210627b20c3148ea2d7
Corresponding commit msg:  Fix typo
Processing Commits:  63%|                     | 1053/1676 [01:10<1:11:47,  6.91s/it]Processing for commit:  9f61ed99da17c3553307ebc8c8477372b8ce6622
Corresponding commit msg:  Merge pull request #2333 from D4D3VD4V3/patch-1
Processing for commit:  90854c756a8ccbe0bedf35dd92eb7ff69f7401b1
Corresponding commit msg:  resolve merge conflicts
Processing Commits:  63%|                     | 1055/1676 [01:10<46:05,  4.45s/it]Processing for commit:  31174fecd2dccd48aa56d4eefc7c808ae93d9262
Corresponding commit msg:  resolve merge conflicts
Processing for commit:  d2a46dc56d9af45af2ea4eb43c23686b14a76b92
Corresponding commit msg:  fix doc build error
Processing Commits:  63%|                     | 1057/1676 [01:10<30:37,  2.97s/it]Processing for commit:  aeb82a404f2926613f83070e039de34cc5842b52
Corresponding commit msg:  Merge pull request #2342 from jrbaez01/issue/#2341-accept-one-default-argument
```

There are 5391 commits that the tool attempts to process.

The tool can resume progress from where it left the last time, except for extraction of bug-fixing and bug-introducing commit pairs, since it runs the SZZ algorithm every time.

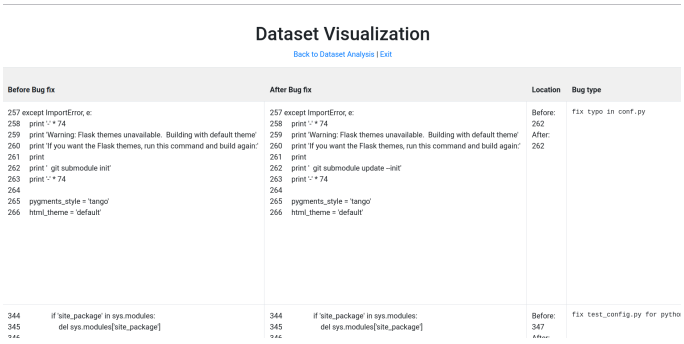
```
Processing Commits: 100%|                    | 1676/1676 [4:31:43<00:00, 25.88s/it]
Removing project folder:  flask.git
 * Serving Flask app 'Minecpp.app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [18/Jan/2025 21:07:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 21:07:46] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Jan/2025 21:57:00] "GET /visualization/1 HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 21:59:56] "GET /visualization/2 HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 22:00:01] "GET /visualization/3 HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 22:00:30] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 22:00:33] "GET /quantitative HTTP/1.1" 200 -
/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py:114: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
127.0.0.1 - - [18/Jan/2025 22:00:41] "POST /quantitative HTTP/1.1" 200 -
/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py:114: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
127.0.0.1 - - [18/Jan/2025 22:01:05] "POST /quantitative HTTP/1.1" 200 -
/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py:114: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
127.0.0.1 - - [18/Jan/2025 22:01:19] "POST /quantitative HTTP/1.1" 200 -
/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py:114: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
127.0.0.1 - - [18/Jan/2025 22:01:33] "POST /quantitative HTTP/1.1" 200 -
[2025-01-18 22:01:39,905] ERROR in app: Exception on /quantitative [POST]
Traceback (most recent call last):
  File "/home/hp/.local/lib/python3.10/site-packages/flask/app.py", line 2528, in wsgi_app
    response = self.full_dispatch_request()
  File "/home/hp/.local/lib/python3.10/site-packages/flask/app.py", line 1825, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/home/hp/.local/lib/python3.10/site-packages/flask/app.py", line 1823, in full_dispatch_request
    rv = self.dispatch_request()
  File "/home/hp/.local/lib/python3.10/site-packages/flask/app.py", line 1799, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py", line 85, in quantitative
    rows = int(request.form['num_rows'])
ValueError: invalid literal for int() with base 10: ''
127.0.0.1 - - [18/Jan/2025 22:01:39] "POST /quantitative HTTP/1.1" 500 -
/home/hp/.local/lib/python3.10/site-packages/Minecpp/app.py:114: UserWarning: Starting a Matplotlib GUI outside of the main thread will likely fail.
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
127.0.0.1 - - [18/Jan/2025 22:01:48] "POST /quantitative HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2025 22:02:15] "GET / HTTP/1.1" 200 -
```

At the end of processing, it shows an interface using flask server that helps for visualisation.



One might need to reload the browser page for this.

- Data-set visualization for Flask repository



  - The code snippets redirect to the lines in the GitHub repository. But this only works when the URL is given correctly as input. An example URL that it was redirecting to is https://github.com/pallets/flask.git/commit/5d011c356dfc659a4c2ef66e8c847d489aef4e9c . This address is wrong. After removing the `.git` from the URL, it redirects to the correct place. This happened because the input was `https://github.com/pallets/flask.git` rather than `https://github.com/pallets/flask` to the `minecpp` shell command. Instead of giving an error, the tool just continued with the input. The same happens, if I miss-spell it as `https://github.com/**palets**/flask` .
  - The coding effort metric is really good. It does convey the effort that the developer put in modifying the code.

- “Bug Type” for Flask

  The text generated for “Bug Type” is usually more descriptive compared to the commit message, but is really inaccurate. Because of this, it performs very poorly at times.

  - Example 1

    For example, consider this commit : https://github.com/pallets/flask/commit/5da2c00419d09ce2d3488263036ffd14563fb05a

    In this commit, the `pytest` library is imported and modifications are made to the file `tests/test_appctx.py` and lots of methods are removed from `tests/__init__.py` . It should be noted that `pytest` library was already in use beforehand. All of this is done with the commit message “Rewrite assertion methods” , which, although isn’t descriptive enough, is a valid description of what changes took place. Meanwhile the message generated by the LLM is “update test_appctx.py to use pytest 2.x” . This isn’t exactly what happened. In my opinion, the commit message, although not very helpful is more suitable to describing what happened, rather than the LLM generated text.

  It also seems that the LLM has a bias towards including any libraries or technologies mentioned in the code context of the bug.

  - Example 2

    For another example, consider this other commit : https://github.com/pallets/flask/commit/df711eac90223db990eae05a60ee1f6b403443e1

    Here, the Python version requirements are changed to 3.4 in `.travis.yml` , an attribute `monkeypatch` is added to the method `test_egg_installed_paths` , and some other changes are made in `tox.ini` . Nowhere was anything related to “Python 2.5” changed. The commit message is “Some fixes” , which although really bad, is true about this commit. The message generated by the LLM is ”fix test_config.py for python 2.5” . The inclusion of “Python 2.5” was unnecessary. It could have been something more obvious, such as “added monkeypatch to test_egg_intalled_paths in test_config.py” and it would be both, more helpful, as well as less mis-understandable.

  For deciding whether a commit message better explains a commit over other messages, I am using these criterion with decreasing priority from this list:

  i. It must not be more than 50 characters.
  ii. It must not give false information and must not be inaccurate
  iii. It must be descriptive and inform about either all the different changes done, or about some main goal accomplished by doing these changes

  - Here’s some more comparisons between the LLM’s output and the commit messages:

	Bug type	Commit Message	...	Better	Reasor
0	fix typo in conf.py	Fixed command	...	BOTH	LLM's
1	fix test_config.py for pyt...	Some fixes	...	NONE	The cc
2	fix test_config.py for pyt...	Some fixes	...	NONE	The cc
3	fix test_config.py for pyt...	Some fixes	...	NONE	The cc
4	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
5	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
6	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
7	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
8	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
9	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
10	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
11	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
12	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
13	update test_appctx.py to u...	Rewrite assertion methods	...	DEV	pytest
14	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
15	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
16	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
17	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
18	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
19	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
20	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
21	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
22	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
23	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
24	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
25	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
26	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
27	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
28	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
29	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
30	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
31	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
32	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
33	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
34	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
35	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
36	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
37	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
38	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
39	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
40	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert
41	use assert_equal and asser...	Rewrite assertion methods	...	DEV	assert

```
42  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
43  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
44  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
45  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
46  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
47  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
48  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
49  use assert_equal and asser...  Rewrite assertion methods  ...   DEV  assert
```

- This was done more efficiently using these scripts :

```
import pandas as pd
import os
data = pd.read_csv("flask.csv")
data["URL"] = ("https://github.com/pallets/flask/commit/"
+ data["Fixed Commit"])
data = data.iloc[:50]
context = ["Bug type","Commit Message",
"File Path","Fixed Commit","Buggy Commit","URL"]
C = data[context]
L = data["Location"]
before = data["Before Bug fix"]
after = data["After Bug fix"]
for i in range(50):
    fold = f"row{i+1}"
    try:os.mkdir(fold)
    except: pass
    open(fold + "/before.txt",'w').write(before[i])
    open(fold + "/after.txt",'w').write(after[i])
    open(fold + "/context.txt",'w')
    s = L[i] + "\n" + "\n".join(
    [f"{attr} : {C[attr][i]}"
    for attr in context])
    f.write(s)
    f.close()
```

```
import os
import pandas as pd
Winner = []
Reason = []
for i in range(50):
    file = f"row{i+1}/context.txt"
    s =open(file,'r').read().split("\n")
    n = len(s)
    assert n == 10
    Winner.append(s[-2])
    Reason.append(s[-1])
context = ["Bug type","Commit Message","File Path","Fixed Commit"]
data = pd.read_csv("flask.csv")[context].iloc[:50]
data["Better"] = Winner
data["Reason"] = Reason
pd.options.display.max_colwidth =30
print(data)
data.to_csv("data.csv",index=False)
```

- Cross verification of my findings

  > Overall I think this might be a great tool if it worked as advertised. While the setup and tool execution were really smooth, also thanks to the simple interface, the obtained output made me wonder if the submitted paper only tells half the story. It seems that the obtained data may entail significant and non-trivial post-processing to obtain an actual bug dataset.

  - Reviewer (FSE)

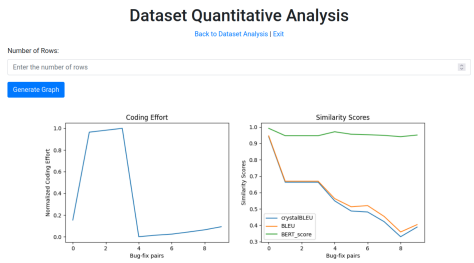
Source: Lecture Slides

- A few things that can be made better
  a. Condense everything related to one file changed in a commit in a single row.
  There were up-to 30 rows for the same file in a commit in the first 50 rows themselves. All of them had the same LLM output. In such a scenario, it makes sense to bunch them together.
  b. Having simple obvious truths for commit message serves better than trying to write a more descriptive, but inaccurate, or misleading messages.
  c. Using the tokens generated by a lexer for the NLP tasks is probably better than just using the “tokens” generated by the usual tokenization (punctuation separated tokens). This allows for operators to be recognized by the LLM, so that it isn’t confused between things like `assert x == y` and `assert_equal_equal` .
  Of course, this means we can’t use any tricks like transfer learning, or fine-tuning pre-trained models, making the problem a lot harder. One way might be to connect to an already trained model like chatGPT or Google’s Bard for this task.
  For example, after giving this to chatGPT as input , it generated this output :

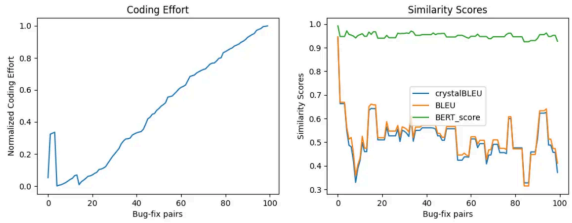
```
Refactor tests to use `assert` instead of `self.assert_equal`
```

  Chat link : https://chatgpt.com/share/678d49b3-cb54-8010-af3f-ccb2dadbeefa
  This is much better than the message generated by MineCPP, and was generated in less than a second, making it very practical.
  d. Since the context outside a file is not available, the LLM lacks the knowledge to actually know what the developer is doing, and so it makes inaccurate predictions as to what the developer is doing.
  e. A lot of things which are not bugs might also be involved unfortunately. For example, a commit updating the methods for asserting conditions can be classified as a bug when it might not be. This is because of high variety of keywords used while extracting the bug-fixing commits. Limiting them, or making them be in a particular order, such as `fix(ed|es|ing|)|...` , and then `bug(s?)|problem(s?)|...` after that will be better. And keywords like `update` should be avoided. This selectivity of course comes at the cost of not having a large enough data-set.

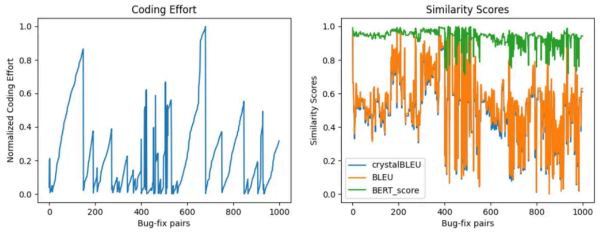
- Quantitative Analysis for Flask repository
  - 10 rows



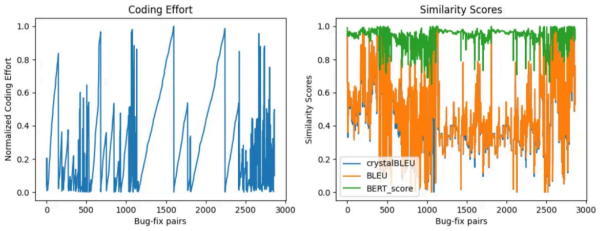
  - 100 rows



  - 1000 rows



  - Full data-set



As you can see, the similarity scores using `BLEU` and `CrystalBLEU` are usually close to 0.5, while the BERT scores are very high.

The coding effort has saw-tooth patterns (sudden jumps in the coding effort). This is most probably because of the changes in the import statements and other library related changes.

Lab 3

Script for mining latest 500 commits

```
import sys,csv,os
from pydriller import Repository
columns = ["old_file path","new_file path","commit SHA","parent commit SHA","commit mess
if sys.argv[3]=="hist":hist = True
elif sys.argv[3]=="myers":hist = False
else:raise ValueError("Give proper arguments")
columns.append(f"diff_{sys.argv[3]}")
rows = []
count=0
last_n=500
commits = []
Repo = Repository(sys.argv[1],only_no_merge=True,order='reverse',num_workers =1,histogra
for x in Repo.traverse_commits():
  if (x.in_main_branch==True):
    count=count+1
    commits.append(x)
    if count == last_n:break
in_order = []
for value in range(len(commits)):in_order.append(commits.pop())
commits=in_order
for i,commit in enumerate(commits):
  print('[{}/{}] Mining commit {}.{}'.format(i+1,len(commits),sys.argv[1],commit.hash))
  diff = []
  try:
    for m in commit.modified_files:
      if len(commit.parents) > 1:continue
      rows.append([m.old_path,m.new_path,commit.hash,commit.parents[0],repr(commit.msg),
  except:pass
try:os.mkdir(sys.argv[2]+'_results')
except:pass
f = open(sys.argv[2]+'_results/commits_info_'+sys.argv[3]+'.csv', 'w')
f.write("")
f.close()
with open(sys.argv[2]+'_results/commits_info_'+sys.argv[3]+'.csv', 'a') as csvFile:
  writer = csv.writer(csvFile)
  writer.writerow(columns)
  writer.writerows(rows)
```

This script is inspired from `getCommitsInfo.py` and thus has the same name. It takes in 3 arguments: the project URL, name of directory to store output in, and the method for evaluating “git diff”.

- Output of the script

```
python3 STTLab3/cs202_miner/getCommitsInfo.py
https://github.com/pallets/flask Flask myers
```

outputs

```
   old_file path new_file path commit SHA parent commit SHA commit message diff_myers
0  .pre-c...     .pre-c...     a025ee...  9e50ad...         'updat...      '@@ -2...
1  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -8...
2  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -1...
3  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -6...
4  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -8...
```

```
python3 STTLab3/cs202_miner/getCommitsInfo.py
https://github.com/pallets/flask Flask hist
```

outputs

```
   old_file path new_file path commit SHA parent commit SHA commit message  diff_hist
0  .pre-c...     .pre-c...     a025ee...  9e50ad...         'updat...      '@@ -2...
1  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -8...
2  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -1...
3  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -6...
4  requir...     requir...     a025ee...  9e50ad...         'updat...      '@@ -8...
```

I am printing the “head” of the pandas data-frames, since the actual CSV file is too large. All the columns for the CSV files are not listed, once again, because they take up space. The “diff” is stored by converting it to a raw string using `repr` function.

- Matches between outputs from “histogram” and “myers” algorithms

```
import pandas as pd
import os
data1 = pd.read_csv("Flask_results/commi
data2 = pd.read_csv("Flask_results/commi
data2["diff_hist"] = data1["diff_hist"]
data2["Matches"] = 1*(data2["diff_hist"]
eq = data2["Matches"].values.sum()
neq = len(data2["Matches"]) - eq
print("equal :",eq)
print("not equal :",neq)
print(round(100*neq/(neq+eq),2),"% match
path = os.getcwd() + "/STTLab3/cs202_min
data2.to_csv(path,index=False)
```

I have used this Python script to calculate the values for “Matches” column, which is just an indicator variable (either 0 or 1).

This script also outputs the number of matches and mis-matches. The output is:

```
equal : 1460
not equal : 55
3.63 % match
```

The CSV file output, as a dataframe looks like this :

	old_file path	new_file path	commit SHA	parent commit SHA	commit message	diff_myers
0	.pre-c...	.pre-c...	a025ee...	9e50ad...	'updat...	'@@ -2...
1	requir...	requir...	a025ee...	9e50ad...	'updat...	'@@ -8...
2	requir...	requir...	a025ee...	9e50ad...	'updat...	'@@ -1...
3	requir...	requir...	a025ee...	9e50ad...	'updat...	'@@ -6...
4	requir...	requir...	a025ee...	9e50ad...	'updat...	'@@ -8...

- Mis-matches for different file formats

  I am using this script for this task:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = pd.read_csv("STTLab3/cs202_miner/Flask_results/commits_info_full.csv")
file_names = data["new_file path"].values
file_extensions =[x.split(".")[-1] if isinstance(x,str) else None for x in file_name
Matches = data["Matches"].values
Ext ={'json':"data",'yaml':"data",'yml':"data",'png':"data",'rst':"doc",'md':"doc",'
matches = {Type:0 for Type in list(Ext.keys())+["data","config","doc","non-code","co
mismatches = {Type:0 for Type in list(Ext.keys())+["data","config","doc","non-code",
for i,m in enumerate(Matches):
    ext = file_extensions[i]
    if ext is None:continue
    if m:Lis = matches
    else:Lis = mismatches
    Lis[ext] += 1
    Lis[Ext[ext]] += 1
    Lis["full"] +=1
    if ext != "code":matches["non-code"] += 1
T = []
for Type in matches:
    eq = matches[Type]
    neq = mismatches[Type]
    T.append([Type,eq,neq,100*neq/(eq + neq)])
T = pd.DataFrame(T,columns = ["type","matches",'mis-matches',"% mis-match"])
print(T)
plt.bar(T["type"],T["% mis-match"],color = ["green"]*(len(matches)-6) + ["blue"]*3 +
plt.xlabel("file type")
plt.xticks(T.index,rotation='vertical')
plt.ylabel("% mis-match")
plt.savefig("STTLab3/cs202_miner/Flask_results/mismatch_percentage.png",format='png'
plt.pie(list(mismatches.values())[:-4],labels=list(mismatches.keys())[:-4],labeldist
plt.savefig("STTLab3/cs202_miner/Flask_results/pie.svg",format='svg')
```

  The dictionary `Ext` gives the type of the artifacts.

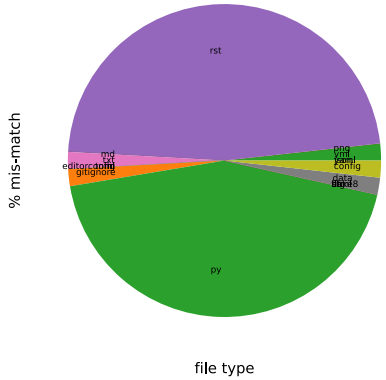
The script gives this data-frame as the output:

	type	matches	mis-matches
0	json	1	0
1	yaml	160	0
2	yml	7	1
3	png	10	0
4	rst	364	27
5	md	13	0
6	txt	189	1
7	editorconfig	1	0
8	ini	23	0
9	in	32	0
10	toml	58	0
11	gitignore	5	1
12	py	545	25
13	sh	3	0
14	cfg	6	0
15	html	4	0
16	flake8	4	0
17	data	178	1
18	config	119	1
19	doc	566	28
20	non-code	1425	55
21	code	562	25
22	full	1425	55

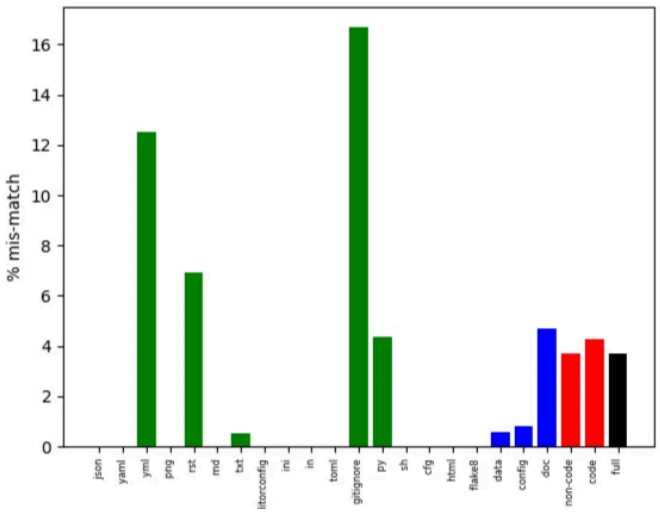
From this, we can get the distribution of matches and mismatches over code and non-code artifacts

Artifact Type	Matches	Mismatches
Code	562	25
Non-code	1425	55

The distribution of mismatches over different files looks like this:



The percentage mismatches can be visualized by this bar plot:



As you can see, the highest percentage mismatches occur in documentation type files, such as text, markdown, etc.

Lab 4

- Repository selected : Flask
- For this task, I'll be using last 3000 non-merge commits for better accuracy of statistics.
- Creating the main table

  The script (named `getCommitsInfo` in last task) for creating the main table containing old file path, new file path … `diff_hist` , and in this case, `new_file_MCC` as well, is only slightly modified, due to the addition of a new column.

```
import sys,csv,os
from pydriller import Repository
columns = ["old_file path","new_file path","commit SHA","parent commit SHA","commit
if sys.argv[3]=="hist":hist = True
elif sys.argv[3]=="myers":hist = False
else:raise ValueError("Give proper arguments")
columns.append(f"diff_{sys.argv[3]}")
columns.extend(["new_file_MCC"])
rows = []
count=0
last_n=3000
commits = []
Repo = Repository(sys.argv[1],only_no_merge=True,order='reverse',num_workers =1,hist
for x in Repo.traverse_commits():
  if (x.in_main_branch==True):
    count=count+1
    commits.append(x)
    if count == last_n:break
in_order = []
for value in range(len(commits)):in_order.append(commits.pop())
commits=in_order
for i,commit in enumerate(commits):
  print('[{}/{}] Mining commit {}.{}'.format(i+1,len(commits),sys.argv[1],commit.has
  diff = []
  try:
    for m in commit.modified_files:
      if len(commit.parents) > 1:continue
      row = [m.old_path,m.new_path,commit.hash,commit.parents[0],repr(commit.msg),re
      rows.append(row)
  except:pass
try:os.mkdir(sys.argv[2]+'_results')
except:pass
f = open(sys.argv[2]+'_results/commits_info_'+sys.argv[3]+'.csv', 'w')
f.write("")
f.close()
with open(sys.argv[2]+'_results/commits_info_'+sys.argv[3]+'.csv', 'a') as csvFile:
  writer = csv.writer(csvFile)
  writer.writerow(columns)
  writer.writerows(rows)
```

The output of this is too big to include and is instead represented by this data-frame :

	old_file path	new_file path	commit SHA	parent commit SHA	commit message	diff_hist
0	flask/...	flask/...	eba2f7...	153d72...	'Clari...	'@@ -2...
1	docs/d...	docs/d...	c726e5...	eba2f7...	'Fix l...	'@@ -6...
2	tests/...	tests/...	fafcc0...	eba2f7...	'Added...	"@@ -5...
3	docs/p...	docs/p...	5304a1...	a070b4...	'Add r...	'@@ -7...
4	CHANGES	CHANGES	7155f1...	a070b4...	'Added...	'@@ -1...

As you can see, some of these are non-code files. Thus, `new_file_MCC` has value `NaN` .

- Getting MCC for old version of file in a commit

```
import pandas as pd
pd.set_option("max_colwidth",10)
data = pd.read_csv(
"STTLab4/cs202_miner/Flask_results/"+
"commits_info_hist.csv")
old_id = data[[
"parent commit SHA","old_file path"]]
old_id.columns = ["commit SHA",
"new_file path"]
data.set_index(["commit SHA",
"new_file path"],inplace=True)
old_id = [
tuple(x) for x in old_id.values]
ind= set(data.index)
Comp = data["new_file_MCC"]
val = [(Comp[x].values[0] if x in ind
else pd.NA) for x in old_id]
data["old_file_MCC"] = val
data.to_csv(
"STTLab4/cs202_miner/Flask_results/"+
"commits_info_comp.csv")
data.dropna(subset=['new_file_MCC'],
inplace=True)
print(len(data))
data.to_csv(
"STTLab4/cs202_miner/Flask_results/"+
"commits_info_comp_clean.csv")
```

I utilized simple database lookups using the parent commit SHA to get the MCC for the older version of the file modified in a commit. This was done using the Python script to the left.

Here I'm using the "Multi-Index" feature in pandas to query from the database. This is possible because `commit SHA` and `new_file path` combined acts as super-key for the table, while `parent commit SHA` and `old_file path` combined act as a foreign key, referencing rows of the same table.

I am storing the raw output of this in a file named `commits_info_comp.csv` , and the cleaned (without `nan` values for cyclomatic complexity after commit) output in `commits_info_comp_clean.csv` .

The size of cleaned data is 3465 rows.

Once again, since I can't include the whole CSV file, I'll represent this with view of the data-frame:

```
                     old_file path parent commit SHA commit message  diff_hist  nev
commit SHA new_fil...
eba2f70... flask/w...  flask/...     153d72...         'Clari...      '@@ -2...
fafcc02... tests/f...  tests/...     eba2f7...         'Added...      "@@ -5...
7155f11... flask/a...  flask/...     a070b4...         'Added...      '@@ -1...
           tests/f...  tests/...     a070b4...         'Added...      "@@ -2...
afe5d3c... flask/a...  flask/...     a0fd8f...         'Added...      '@@ -9...
```

- Getting the top 3 most changed files

  Since a file name might get changed, I must keep track of that as well. So, to do this, I am adding using the “foreign key” that we discussed about previously, to traverse the database for a particular file and get all the commits that modified this file, as well as the cyclomatic complexity after every commit. All this is done using this Python program :

```
import matplotlib.pyplot as plt
import pandas as pd
import os
from ast import literal_eval
data = pd.read_csv(
"STTLab4/cs202_miner/Flask_results/commi
).dropna(subset=["new_file path","old_fi
print(len(data))
edges = data[["old_file path","new_file
"commit SHA","new_file_MCC"]]
edges = list(edges.values)
print("file name changes will be printed
fc = {}
for e in edges:
    old,new,commit,comp =e
    if old == new:
        if old in fc:fc[old].append((com
        else:fc[old] = [(commit,new,comp
        continue
    e_desc = f"{commit} : {old} -> {new}
    print(e_desc)
    if old in fc:
        fc[new] = fc[old]
        fc[new].append((commit,new,comp)
        fc[old] = []
    else:
        assert (new not in fc or fc[new]
        f"{fc[new][-1]} already uses fil
        + " So, {commit} can't rename {o
        fc[new] = [commit]
lens = [(len(fc[final]),final) for final
lens = sorted(lens,reverse=True)
top3 = lens[:3]
print("\nTop 3 are :")
for x in top3:print(x[1],":",x[0],"commi
top3 = [x[1] for x in top3]
top3_commits = [fc[x] for x in top3]
for i,final_name in enumerate(top3):
    commits = top3_commits[i]
    f = open("STTLab4/cs202_miner/Flask_
    + final_name.split("/")[-1] + ".chan
    S = "\n".join(
    [",".join([str(y) for y in x]) for x
    S = "commit_SHA,new_file path,new_fi
    f.write(S)
    f.close()
```

This also saves the commit hashes for different files, so that we can create the control flow graphs later.

The output of running this program is

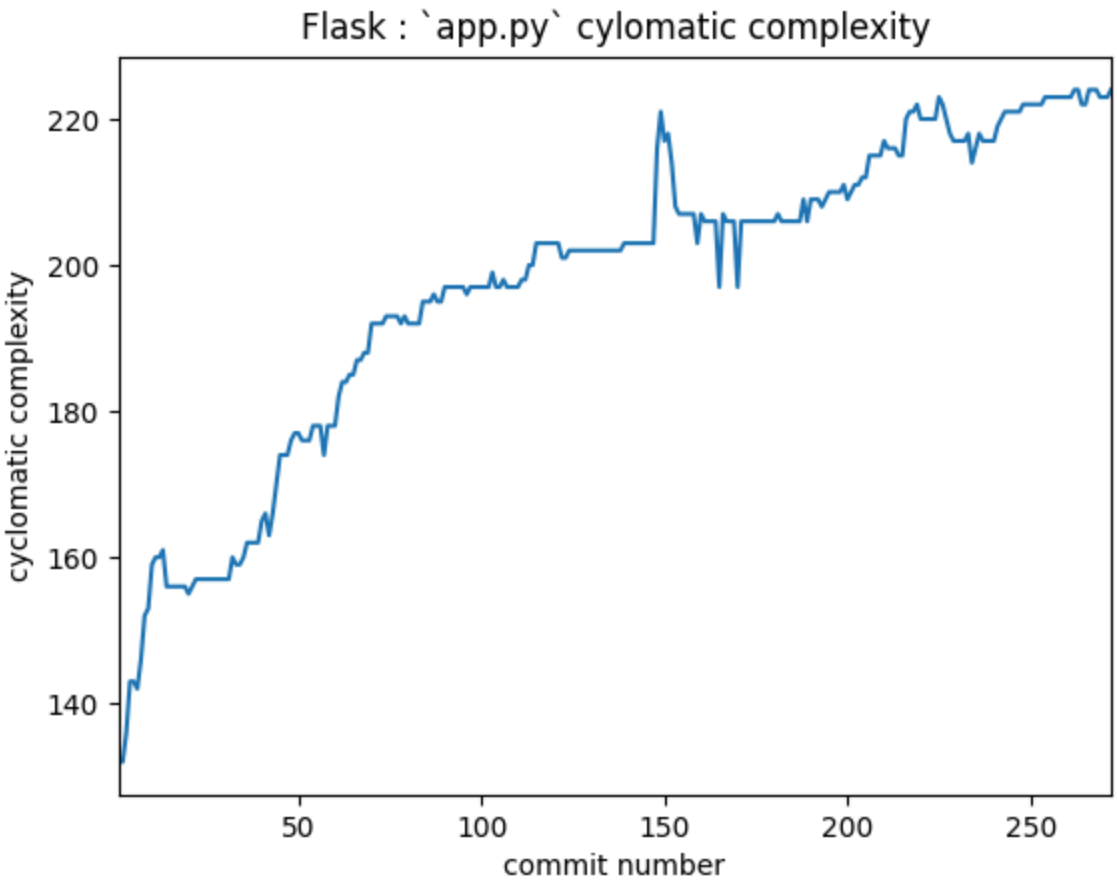
```
3350
file name changes will be printed :
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/appctx.py -> tests/test_appctx.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/basic.py -> tests/test_basic.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/blueprints.py -> tests/test_bluepri
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/config.py -> tests/test_config.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/deprecations.py -> tests/test_depre
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/examples.py -> tests/test_examples.
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/ext.py -> tests/test_ext.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/helpers.py -> tests/test_helpers.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/regression.py -> tests/test_regress
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/reqctx.py -> tests/test_reqctx.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/signals.py -> tests/test_signals.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/subclassing.py -> tests/test_subcla
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/templating.py -> tests/test_templat
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/testing.py -> tests/test_testing.py
961db8ad7290f1bd91058b11a5de31e27bf28563 : tests/views.py -> tests/test_views.py
17d4cb3828b9520be6c4e64e6b7548a561aa6e7a : examples/flaskr/test_flaskr.py -> example
5f009374fd28aba381f375391b70ac131926bd9d : examples/minitwit/test_minitwit.py -> exa
8cf32bca519723c5d97f123127cdec81dba868f6 : examples/flaskr/flaskr/flaskr.py -> examp
5e1ced3c055f7eb567bf7266c98de3d44ceea1b4 : flask/json.py -> flask/json/__init__.py

Top 3 are :
flask/app.py : 272 commits
flask/helpers.py : 155 commits
tests/test_basic.py : 128 commits
```

Note: Here, some rows will still have null values for cyclomatic complexity before commit. The reason I am not dropping those row is because we are only interested in the “most frequently changed” files. Many of these files have parent commits that are not in the block; that is, the parent commit happened much before the commit that we are interested in.

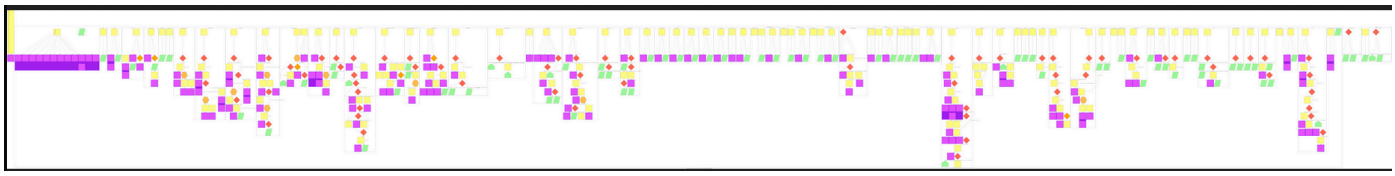
Thus, `flask/app.py` is the most changed file; and it makes sense, since it contains the `Flask` class, one of the central pieces of the project. The first few lines of the file created for `app.py` reads

```
commit_SHA,new_file path,new_file_MCC
7155f11a723695352323d484610b9a1c6798eb69,flask/app.py,132.0
afe5d3cbd44056893aa8c11f4bf17496cee92022,flask/app.py,132.0
5500986971b28f270a27db633acf19984eee609e,flask/app.py,136.0
02a131746074203dc8d747308f4d2c7ece8aa743,flask/app.py,143.0
ae00f6d149c353e1ecb683ccbf01be221324a646,flask/app.py,143.0
acac64e36a046cebae07bee7f3e8106b4a80288a,flask/app.py,142.0
68473291344629dff3955fdc07776f2fca7b9b69,flask/app.py,146.0
153ecbc9202f249c4037602115ed2c2b5785550d,flask/app.py,152.0
187cb80dcc1b087f1a7b7d4d67afbd531ad01cd2,flask/app.py,153.0
175d43b2f9c7825b0df4c1049ab571cb1877a807,flask/app.py,159.0
```

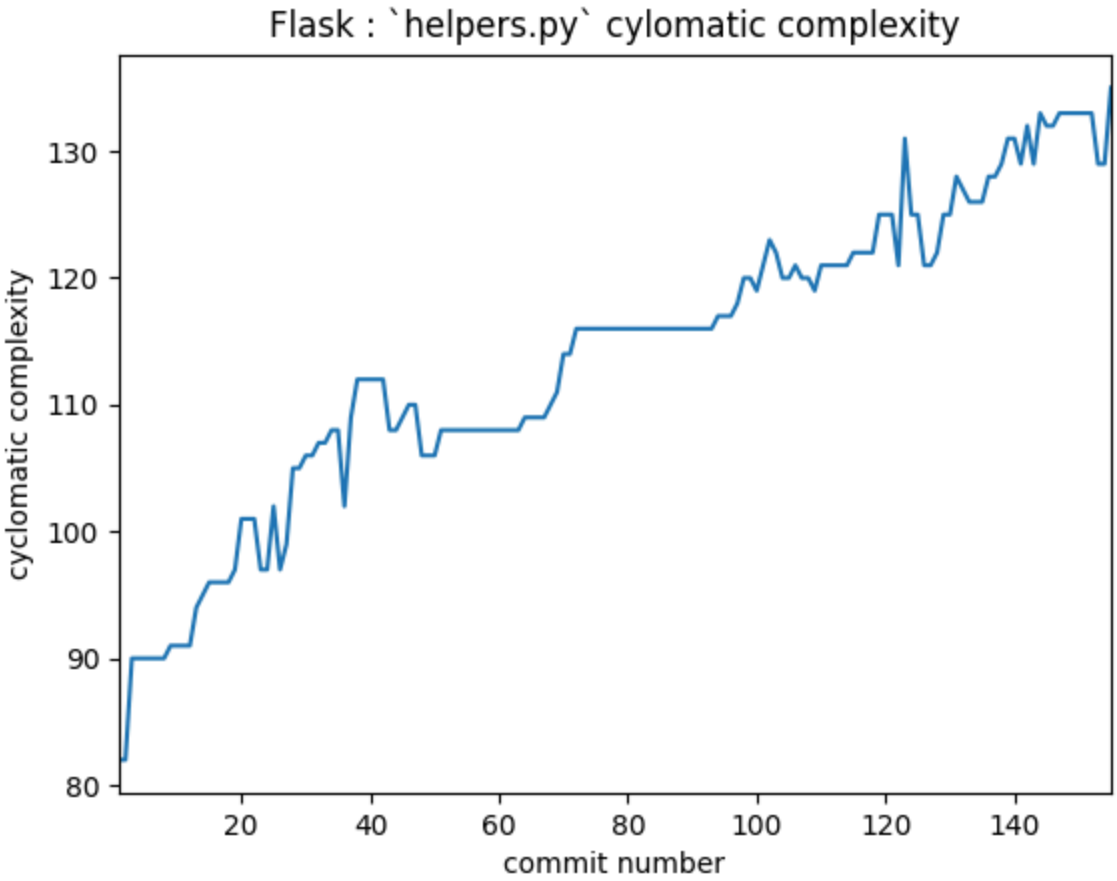
With this, we can plot the change in the cyclomatic complexity over time. The plot thus obtained is :



Unfortunately, it is very difficult to draw the CFGs for the code in this file, since the graphs become very big. For example, this is one such CFG.



So instead, I will try doing this for the second most frequently changed file, namely `flask/helpers.py` .
The complexity varies with time for this file as follows:



- The CFG evolves like this

- Side by side comparison of commit “0” and “105”

From the CFG evolution, it does seem that the code complexity, although already high (around 80 at commit 0), is increasing with more commits (around 120 after 105 commits).

- The CFGs were created using this Python script :

```
import pandas as pd, matplotlib.pyplot as plt
from pydriller import Repository
from py2cfg import CFGBuilder
import ast,re,traceback,os
def render_as_CFG(code_file="trial.py",graph_file="trial"):
    f = open(code_file,'r')
    src = f.read()
    f.close()
    AST = ast.parse(src)
    g = CFGBuilder().build(graph_file,AST)._build_visual()
    (g.node_attr["width"],g.node_attr["height"],
    g.node_attr["fontsize"],g.graph_attr["dpi"]
    ) = ("4","4","0.1","50")
    g.render(graph_file,view=False,format='jpeg')
top = "STTLab4/cs202_miner/Flask_results/helpers.py.changes"
changes = pd.read_csv(top)
y = changes["new_file_MCC"].values
x = range(1,len(y)+1)
plt.plot(x,y)
plt.title("Flask : `helpers.py` cylomatic complexity")
plt.xlabel("commit number")
plt.ylabel("cyclomatic complexity")
plt.xlim(1,len(y))
plt.savefig("STTLab4/cs202_miner/Flask_results/helpers.py.changes.png")
Repo = Repository("https://github.com/pallets/flask",
only_no_merge=True,num_workers =1,histogram_diff = True,
only_commits = list(changes.iloc[0:n:n//10]["commit_SHA"].values),)
folder = "STTLab4/cs202_miner/Flask_results/helpers.py.changes.cfgs"
try:os.mkdir(folder)
except:pass
i = 0
for commit in Repo.traverse_commits():
    C = [x for x in commit.modified_files if
    x.new_path == "flask/helpers.py"][0].source_code
    filepath = folder + f"/commit{i}:{commit.hash}.py"
    f = open(filepath,'w')
    f.write(C)
    f.close()
    try:render_as_CFG(filepath,folder + f"/commit{i}:{commit.hash}")
    except:print("skipped",i)
    i += n//10
```

Conclusion

The labs provided a comprehensive introduction to various tools, techniques, and methodologies used in software development, particularly focusing on version control systems, bug mining, and code analysis. Below is a detailed conclusion summarizing the key findings and insights:

1. **Version Control Systems (VCS) and Git**:
   Git is a widely used VCS that enables developers to manage code changes efficiently. It supports local and remote repositories, pull requests, and continuous integration, which are essential for collaborative development. The labs highlight Git's importance in maintaining code history, enabling rollbacks, and improving the DevOps cycle.
2. **PyDriller and Code Analysis**:
   PyDriller is a powerful Python library for mining commits from GitHub repositories. It provides detailed information about code changes, including file paths, diffs, and cyclomatic complexity. The labs demonstrate how PyDriller can be used to analyze codebases and extract valuable insights from commit histories.
3. **Coding Effort and Abstract Syntax Trees (AST)**:
   Coding effort measures the number of nodes in the AST before a buggy node. ASTs are crucial for understanding the structure of code and are used in various tools for code analysis, compilation, and evaluation. I was also introduced to tools like `tree-sitter` for generating ASTs across multiple programming languages via the labs.
4. **Control Flow Graphs (CFG) and Cyclomatic Complexity**:
   CFGs are graphical representations of a program's control flow, and cyclomatic complexity measures the number of independent paths in a program. Tools like `py2cfg` can generate CFGs and calculate cyclomatic complexity, which is useful for identifying complex and potentially bug-prone code.
5. **Bug Mining**:
   We were introduced to tools like Minecraft and MineCPP, which automate the mining of bug fixes from GitHub repositories. These tools use algorithms like SZZ to identify bug-introducing and bug-fixing commits, providing precise code context for each bug.
6. **MineCPP Enhancements**:
   MineCPP extends Minecraft by adding features like coding effort, test case indicators, cyclomatic complexity, and similarity scores. These enhancements provide a more comprehensive dataset for analyzing bugs and their fixes. I also found some limitations of MineCPP, particularly in generating accurate bug type descriptions using LLMs.
7. **Quantitative Analysis and Similarity Scores**:
   The report presents a quantitative analysis of bug fixes using similarity scores like BLEU, CrystalBLEU, and Code BERT. These scores help measure the similarity between buggy and

fixed code snippets. The analysis reveals that BERT scores are generally higher than BLEU and CrystalBLEU scores.

8. **Lab Experiments and Practical Applications**:
   The report details several lab experiments, including configuring Git, creating and managing repositories, using PyLint for code linting, and running MineCPP on the Flask repository. These experiments demonstrate the practical application of the tools and techniques discussed.
9. **Challenges and Limitations**:
   There are still several challenges, including the inaccuracy of LLM-generated bug type descriptions, the difficulty in handling non-code artifacts, and the need for better selection criterion while mining bugs. But it is still a large step forward for language agostic tools, considering the limitations of existing tools in handling large and complex codebases.
10. **Future Improvements**:
    There are several improvements possible, such as :
    i. condensing data related to the same file in a commit
    ii. using lexer-generated tokens for NLP tasks
    iii. integrating pre-trained models like ChatGPT for better bug type descriptions.
    These improvements could enhance the accuracy and usability of bug mining tools.
11. **Visualization and CFG Evolution**:
    The report includes visualizations of cyclomatic complexity over time and the evolution of CFGs for frequently changed files like `flask/app.py` and `flask/helpers.py`. These visualizations provide insights into how code complexity evolves with each commit and how CFGs can be used to understand code changes.