Assignment-1

Software Tools and Techniques for CSE

Shardul Junagade (23110297)
Repository: cs202-stt

September 8, 2025

Contents

Lab 2: Mining Bug-Fixing Commits, LLM Inference, Rectifier, and Evaluation

Repository Link: cs202-stt/lab2

1.1 Introduction, Setup, and Tools

1.1.1 Introduction

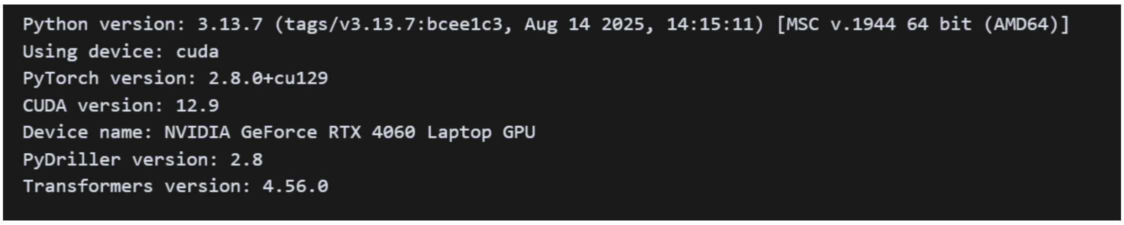
This lab focused on mining open-source repositories to study bug-fixing commits and commit message alignment. I implemented a pipeline to:

1. Identify bug-fixing commits from a real-world project.
2. Extract file-level diffs from those commits.
3. Use a pre-trained LLM to generate concise summaries for each file-level change.
4. Rectify those messages to make them more precise and context-aware.
5. Evaluate the quality of developer, LLM, and rectified messages using semantic similarity with CodeBERT.

The motivation behind this pipeline was that commit messages are not always reliable indicators of what a change actually fixes. Developers may batch multiple fixes, write vague messages, or skip details. Automated tools and rectifiers can help make these messages more consistent and useful.

1.1.2 Environment and Tools

- **Operating System:** Windows 11
- **Terminal:** Powershell 7
- **Python:** 3.13.7
- **PyTorch:** 2.8 (with CUDA 12.9)
- **Transformers:** 4.56
- **PyDriller:** 2.8
- **Models used:**
  - mamiksik/CommitPredictorT5 (LLM Inference)
  - codellama:7b via Ollama (Rectifier)
  - microsoft/codebert-base (Evaluation)
- **Repository analyzed:** 3b1b/manim



Environment Details

1.2 Methodology and Execution

1.2.1 Repository Selection

For this lab, I chose the repository 3b1b/manim. Manim (short for Mathematical Animation Engine) is a Python library that started with Grant Sanderson's 3Blue1Brown channel and has since grown into a large open-source project. It is mainly used to create mathematical animations and visualizations, and because of its popularity it now has a wide contributor base and frequent updates.

I felt Manim was a good choice because it's not just an academic toy project but a tool actually used by educators, researchers, and content creators. That also means its commit history has plenty of real bug fixes to study, which fits well with the aim of this assignment.

Selection Criteria

Repository Statistics

While narrowing down the repository, I kept the following points in mind:

1. **Number of commits:** Manim has more than **6300 commits**, which is large enough to give me enough bug-fixing commits for analysis.
2. **Popularity:** With around **78k stars** and **6.7k forks**, the project has a huge user base and community involvement, so the data is representative of real usage.
3. **Programming language:** The project is written in **Python**, which works well since the lab tools like PyDriller and radon are also Python-based.
4. **Relevance:** Since Manim deals with mathematical visualization and graphics, correctness and stability are very important. That makes bug-fixing commits here especially meaningful to analyze.
5. **Active development:** The repo is still very active, with the last commit in June 2025 and contributions from over **160 developers**, showing that the project is maintained and evolving.

Based on these reasons, Manim seemed like a balanced and practical choice for carrying out this lab. Then I cloned the repository locally using the `git clone <URL>` command.

```python
# clone the repositories if not already cloned
if not os.path.exists(repo_name):
    print(f"Cloning {repo_name} from {REPO_URL}...")
    subprocess.run(["git", "clone", REPO_URL])
else:
    print(f"Repository {repo_name} already exists. Skipping clone.")
```

```
📦 Repository manim already exists. Skipping clone.
```

Git Clone

1.2.2 Identifying Bug-Fixing Commits

Notebook Link: bugfix_commits.ipynb

I first defined a heuristic to detect bug-fixing commits. I scanned commit messages for keywords such as:

> fix, bug, patch, error, issue, defect, crash, flaw, repair, resolve, solve, fail, leak, vulnerability

This simple keyword filter is fast and transparent. The downside is that it may miss commits where developers did not explicitly mention a bug (false negatives), or capture irrelevant commits where the keyword appeared casually (false positives).

Using PyDriller, I traversed the commit history of the **manim** repository and stored each matching commit in a CSV (`bugfix_commits.csv`) with the following fields:

- Commit hash
- Commit message
- Parent hashes
- Is merge commit?
- Modified files

The following code snippet shows the implementation:

```python
bug_keywords = ['fix', 'bug', 'patch', 'error', 'issue', 'defect', 'crash', 'fault', 'flaw',
                'glitch', 'mistake', 'repair', 'resolve', 'solve', 'fail', 'break', 'broke',
                'overflow', 'leak', 'vulnerability']

def is_bugfix(msg):
    msg = msg.lower()
    return any(word in msg for word in bug_keywords)
```

```python
fields = ['commit_hash', 'commit_message', 'parent_hashes', 'is_merge_commit', 'modified_files']

with open(f"{output_folder}/{output_csv}", 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=fields)
    writer.writeheader()

    commits_list = list(Repository(repo_name).traverse_commits())
    for commit in tqdm(commits_list, desc="Processing commits"):
        if is_bugfix(commit.msg):
            writer.writerow({
                'commit_hash': commit.hash,
                'commit_message': commit.msg.replace('\n', ' ').replace('\r', ' '),
                'parent_hashes': '; '.join(commit.parents),
                'is_merge_commit': commit.merge,
                'modified_files': '; '.join([mf.filename for mf in commit.modified_files])
            })

print(f"Bug-fixing commits written to {output_folder}/{output_csv}")
```

```
Processing commits: 100%|██████████| 6344/6344 [02:48<00:00, 37.66it/s]
Bug-fixing commits written to bugfix_commits.csv
```

Code for extracting Bug Fixing Commits

I traversed 6,344 commits, out of which, the keyword filter flagged 1358 as bug-fix candidates (21%).

```
Number of bug-fixing commits found: 1358
```

	commit_hash	commit_message	parent_hashes	is_merge_commit	modified_files
0	014a277a97759bbc0e6ec8fba588bc	A few fixes to initial point_thickness i	c0994ed0a53f06a555ea8d42903c20	False	constants.py; displayer.py; mobject.p
1	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	096c5c1890e326e67ee387c921901f8	False	__init__.py; __init__.py; animation.py;
2	ac930952f151acf284f6e01e98e8f725	Beginning transform KA article, many	d294d0f951f01b307805d18679509c	False	animation.py; meta_animations.py; si
3	2322018875b218cc156f50d30d865a	quick rgb-should-be-numpy-array b	c7389e008d3dc79f61e8270ff187582	False	mobject.py
4	7ae5a0eccb13713b3439d9c0a0b79b	Slightly faster sort_points method, ar	7f45044bafb27c94ae1b3f900014b3f	False	mobject.py
5	f21f6619a5150d193d85ed31ef60ed5	Fix to Stars Mobject	7ae5a0eccb13713b3439d9c0a0b79b	False	geometry.py; three_dimensions.py
6	68160140233b8f7ebf00aa2894ea57	Bug fix to bug fix on Mobject.fade m	8f8eeea870c2f780cc93950a6de6bd6	False	mobject.py
7	4f57551344e2655e53c9dcf72ee7d17	Bug in Arrow buffer	68160140233b8f7ebf00aa2894ea57	False	geometry.py
8	5964d5206ac15fc0544325728e232b	Fixed ShowCreation	84054286824d0ca17219f7d3e11a5f2	False	simple_animations.py
9	9e54a5a6802d12994b007434b98c9f	Fixed PiCreature mouth, but more w	c0009064d6ee0183f2d36fd15d7442	False	characters.py

20 rows x 5 cols 10 per page Page 1 of 2

Bug Fixing Commits

1.2.3 Extracting File-level Diffs

Notebook Link: diff_extract_and_llm_infer.ipynb

Since commits often modify multiple files, I processed each file separately. For each bug-fixing commit, I extracted the before and after source codes for each modified file and also stored metadata such as filename, change type, and the git diff.

The following image shows the code implementation:

```python
diffs_per_file_csv = 'diffs_per_file.csv'

fields = [
    'Hash','Message', 'File Name','File Path',
    'Change Type','Source Code (before)','Source Code (current)','Diff',
]

with open(f"{output_folder}/{diffs_per_file_csv}", 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=fields)
    writer.writeheader()

    commits_list = list(Repository(repo_name).traverse_commits())
    for commit in tqdm(commits_list, desc="Processing commits"):
        if not is_bugfix(commit.msg):
            continue
        for m in commit.modified_files:
            if m.diff is None or m.diff.strip() == '':
                continue
            writer.writerow({
                'Hash': commit.hash,
                'Message': commit.msg.replace('\n', ' ').replace('\r', ' '),
                'File Name': m.filename,
                'File Path': m.new_path or m.old_path,
                'Change Type': str(m.change_type),
                'Source Code (before)': m.source_code_before or '',
                'Source Code (current)': m.source_code or '',
                'Diff': m.diff,
            })
print(f"Diffs per file written to {output_folder}/{diffs_per_file_csv}")
```

```
Processing commits: 100%|██████████| 6344/6344 [02:45<00:00, 38.33it/s]
Diffs per file written to results/diffs_per_file.csv
```

Code for extracting Per-File Diffs

After running the code, I extracted 2041 file-level diffs and saved these entries to `diffs_per_file.csv`.

```
Number of file diffs extracted: 2041
```

	Hash	Message	File Name	File Path	Change Type	Source Code (before)	Source Code (current)	Diff
0	014a277a97759bbc0e6ec8fba588bc(	A few fixes to initial point_thickness i	constants.py	constants.py	ModificationType.MODIFY	import os import numpy as np GENE	import os import numpy as np PROD	@@ -1,9 +1,6 @@ import os import
1	014a277a97759bbc0e6ec8fba588bc(	A few fixes to initial point_thickness i	displayer.py	displayer.py	ModificationType.MODIFY	import numpy as np import itertools	import numpy as np import itertools	@@ -55,7 +55,7 @@ def paint_mob
2	014a277a97759bbc0e6ec8fba588bc(	A few fixes to initial point_thickness i	mobject.py	mobject\mobject.py	ModificationType.MODIFY	import numpy as np import itertools	import numpy as np import itertools	@@ -21,7 +21,7 @@ class Mobject(
3	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	__init__.py	__init__.py	ModificationType.MODIFY	from animation import * from mobje	from animation import * from scene	@@ -1,10 +1,13 @@ from animatio
4	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	__init__.py	animation__init__.py	ModificationType.MODIFY	from animation import * from transf	from animation import * from meta_	@@ -1,3 +1,4 @@ from animation i
5	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	animation.py	animation\animation.py	ModificationType.MODIFY	from PIL import Image from colour i	from PIL import Image from colour i	@@ -7,11 +7,10 @@ import os impo
6	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	meta_animations.py	animation\meta_animations.py	ModificationType.ADD	*Missing value*	import numpy as np import itertools	@@ -0,0 +1,95 @@ +import numpy
7	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	simple_animations.py	animation\simple_animations.py	ModificationType.MODIFY	import numpy as np import itertools	import numpy as np import itertools	@@ -2,42 +2,12 @@ import numpy
8	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	transform.py	animation\transform.py	ModificationType.MODIFY	import numpy as np import itertools	import numpy as np import itertools	@@ -4,39 +4,12 @@ import inspect
9	2e074afb60d13262ce1e42e83bcf0ed	middle of massive restructure, everyt	displayer.py	displayer.py	ModificationType.MODIFY	import numpy as np import itertools	import numpy as np import itertools	@@ -8,8 +8,7 @@ import cv2 from

20 rows x 8 cols 10 per page Page 1 of 2

File-level Diffs

1.2.4 LLM Inference of "fix type"

Notebook Link: diff_extract_and_llm_infer.ipynb

I used the Hugging Face model **CommitPredictorT5** to infer the type of fix from the diff. I gave the following prompt template to the pretrained model.

```
File: <filename>
Diff: <diff>
```

The model generated concise commit-style summaries (max length = 64 tokens). These were appended to the CSV as an extra column: *LLM Inference (fix type)* and I saved this dataset to `diffs_per_file_with_llm_infer.csv`. This allowed direct comparison between the developer written commit messages and the LLM predictions.

The following code snippet shows the implementation:

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
diffs_per_file_llm_infer_csv = 'diffs_per_file_with_llm_infer.csv'

MODEL = "mamiksik/CommitPredictorT5"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSeq2SeqLM.from_pretrained(MODEL).to(device)

def prepare_prompt(file_path, diff):
    if not isinstance(diff, str) or diff.strip() == '':
        diff = '<NO DIFF AVAILABLE>'
    return f"File: {file_path}\nDiff:\n{diff}"

def llm_infer(filepath, diff):
    prompt = prepare_prompt(filepath, diff)
    inputs = tokenizer(prompt, return_tensors='pt', truncation=True, max_length=2048).to(device)
    with torch.no_grad():
        gen = model.generate(**inputs, max_length=64)
    llm_msg = tokenizer.decode(gen[0], skip_special_tokens=True)
    return llm_msg

with open(f"{output_folder}/{diffs_per_file_llm_infer_csv}", 'w', newline='', encoding='utf-8') as f:
    writer = csv.DictWriter(f, fieldnames=fields + ['LLM Inference (fix type)'])
    writer.writeheader()

    for _, row in tqdm(diffs_per_file_df.iterrows(), total=len(diffs_per_file_df), desc="LLM Inference"):
        llm_msg = llm_infer(row['File Path'], row['Diff'])
        writer.writerow({
            **row.to_dict(),
            'LLM Inference (fix type)': llm_msg
        })

print(f"Diffs with LLM inference written to {output_folder}/{diffs_per_file_llm_infer_csv}")
```

```
LLM Inference:   0%|          | 0/2041 [00:00<?, ?it/s]
LLM Inference: 100%|██████████| 2041/2041 [20:19<00:00,  1.67it/s]
Diffs with LLM inference written to results/diffs_per_file_with_llm_infer.csv
```

Code for LLM Inference

```
Number of file diffs with LLM inference: 2041
```

	Hash	Message	File Name	File Path	Change Type	Source Code (before)	Source Code (current)	Diff	LLM Inference (fix type)
0	014a277a97759bbc0e6ec8fba588bc6e6de65a86	A few fixes to initial point_thickness impleme...	constants.py	constants.py	ModificationType.MODIFY	import os\nimport numpy as np\n\n\nGENERALLY_B...	import os\nimport numpy as np\n\nPRODUCTION_QU...	@@ -1,9 +1,6 @@\n import os\n import numpy as ...	add missing constants
1	014a277a97759bbc0e6ec8fba588bc6e6de65a86	A few fixes to initial point_thickness impleme...	displayer.py	displayer.py	ModificationType.MODIFY	import numpy as np\nimport itertools as it\nim...	import numpy as np\nimport itertools as it\nim...	@@ -55,7 +55,7 @@ def paint_mobjects(mobjects,...	add nudge to displayer.py
2	014a277a97759bbc0e6ec8fba588bc6e6de65a86	A few fixes to initial point_thickness impleme...	mobject.py	mobject\mobject.py	ModificationType.MODIFY	import numpy as np\nimport itertools as it\nim...	import numpy as np\nimport itertools as it\nim...	@@ -21,7 +21,7 @@ class Mobject(object):\n ...	add missing docstring
3	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	middle of massive restructure, everything stil...	__init__.py	__init__.py	ModificationType.MODIFY	from animation import *\nfrom mobject import *...	from animation import *\nfrom scene import *\n...	@@ -1,10 +1,13 @@\n from animation import *\n-...	add missing import statements
4	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	middle of massive restructure, everything stil...	__init__.py	animation__init__.py	ModificationType.MODIFY	from animation import *\nfrom transform import...	from animation import *\nfrom meta_animations ...	@@ -1,3 +1,4 @@\n from animation import *\n-fr...	add missing newline
5	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	middle of massive restructure, everything stil...	animation.py	animation\animation.py	ModificationType.MODIFY	from PIL import Image\nfrom colour import Colo...	from PIL import Image\nfrom colour import Colo...	@@ -7,11 +7,10 @@ import os\n import copy\n im...	add missing config to missing color animation
6	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	middle of massive restructure, everything stil...	meta_animations.py	animation\meta_animations.py	ModificationType.ADD	NaN	import numpy as np\nimport itertools as it\nfr...	@@ -0,0 +1,95 @@\n+import numpy as np\n+import...	add tests for animation.update and animation.u...
7	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	middle of massive restructure, everything stil...	simple_animations.py	animation\simple_animations.py	ModificationType.MODIFY	import numpy as np\nimport itertools as it\nfr...	import numpy as np\nimport itertools as it\nfr...	@@ -2,42 +2,12 @@ import numpy as np\n import ...	add some more classes to the animation class

LLM Inference Samples

1.2.5 Rectifier Formulation

Notebook Link: ollama_rectifier.ipynb

Developer messages and LLM outputs can still be vague or misaligned, especially when multiple files are involved. Many times developers may not clearly specify the bug or issue being addressed. Developers often combine multiple changes/fixes in a single commit and the LLM may not capture all relevant context. To improve clarity, I designed a rectifier with the following rules:

- Input: file name, change type, diff, and optionally developer + LLM messages.
- Output style: `[file]: Fix <bug/issue> in <component> by <specific action>`
- Keep messages short (< 20 words) and avoid vague verbs.

I implemented this rectifier with **Ollama + codellama:7b** for local inference on my NVIDIA RTX 4060 machine and added 1 more column named *Rectified Message* to the CSV for the rectified messages. I saved the results to `ollama_rectified_commits.csv`.

The following image shows the code implementation for the rectifier:

```python
def rectify_message(row):
    file_name = str(row.get("File Name", "file"))
    dev_msg = str(row.get("Message", ""))
    llm_msg = str(row.get("LLM Inference (Fix Type)", ""))
    diff = str(row.get("Diff", ""))
    change_type = str(row.get("Change Type", ""))

    prompt = f"""
You are refining commit messages for bug-fixing commits.

File: {file_name}
Change type: {change_type}

Git Diff:
{diff}

Developer's commit message: {dev_msg if dev_msg else "N/A"}
LLM inference: {llm_msg if llm_msg else "N/A"}

Task:
Generate a **detailed rectified commit message** in this style:
[{file_name}]: Fix [bug/issue/etc] in [function/component] by [specific action].
Keep under 20 words, precise, and avoid vague terms like 'update' or 'change'.
"""

    try:
        response = ollama.chat(model="codellama:7b", messages=[
            {"role": "user", "content": prompt}
        ])
        rectified = response["message"]["content"].strip()
        return rectified if rectified else f"{file_name}: Minor {change_type}"
    except Exception as e:
        return f"{file_name}: Minor update"
```

Code for Rectifier

Hash	Message	File Name	File Path	Change Type	Source Code (before)	Source Code (current)	Diff	LLM Inference (fix type)	Rectified Message
c6e6de65a86	A few fixes to initial point_thickness impleme...	constants.py	constants.py	ModificationType.MODIFY	import os\nimport numpy as np\n\n\nGENERALLY_B...	import os\nimport numpy as np\n\nPRODUCTION_QU...	@@ -1,9 +1,6 @@\n import os\n import numpy as ...	add missing constants	[constants.py]: Fix default point thickness to...
c6e6de65a86	A few fixes to initial point_thickness impleme...	displayer.py	displayer.py	ModificationType.MODIFY	import numpy as np\nimport itertools as it\nim...	import numpy as np\nimport itertools as it\nim...	@@ -55,7 +55,7 @@ def paint_mobjects(mobjects,...	add nudge to displayer.py	[displayer.py]: Fix potential offset issue in ...
c6e6de65a86	A few fixes to initial point_thickness impleme...	mobject.py	mobject\mobject.py	ModificationType.MODIFY	import numpy as np\nimport itertools as it\nim...	import numpy as np\nimport itertools as it\nim...	@@ -21,7 +21,7 @@ class Mobject(object):\n ...	add missing docstring	[mobject.py]: Fix inconsistent point_thickness...
ed28d95ad82	middle of massive restructure, everything stil...	__init__.py	__init__.py	ModificationType.MODIFY	from animation import *\nfrom mobject import *...	from animation import *\nfrom scene import *\n...	@@ -1,10 +1,13 @@\n from animation import *\n-...	add missing import statements	[__init__.py]: Fix [bug/issue/etc] in [functio...
ed28d95ad82	middle of massive restructure, everything stil...	__init__.py	animation__init__.py	ModificationType.MODIFY	from animation import *\nfrom transform import...	from animation import *\nfrom meta_animations ...	@@ -1,3 +1,4 @@\n from animation import *\n-fr...	add missing newline	Fix regression in transform module by adding m...
ed28d95ad82	middle of massive restructure, everything stil...	animation.py	animation\animation.py	ModificationType.MODIFY	from PIL import Image\nfrom colour import Colo...	from PIL import Image\nfrom colour import Colo...	@@ -7,11 +7,10 @@ import os\n import copy\n im...	add missing config to missing color animation	[animation.py]: Fix bug in writeGif function b...
ed28d95ad82	middle of massive restructure, everything stil...	meta_animations.py	animation\meta_animations.py	ModificationType.ADD	NaN	import numpy as np\nimport itertools as it\nfr...	@@ -0,0 +1,95 @@\n+import numpy as np\n+import...	add tests for animation.update and animation.u...	[meta_animations.py]: Fix bug in DelayByOrder ...

Rectifier Sample Results

1.2.6 Evaluation with CodeBERT

Notebook Link: ollama_evaluation.ipynb

To measure how well each message aligned with its code change, I used **microsoft/codebert-base**. First, I generated embeddings for both the diff text and the corresponding messages (developer, LLM, and rectified). Then, I computed the cosine similarity between these embeddings to quantify how semantically close each message was to the actual code change. In this setup, a higher similarity score indicates a stronger alignment between the text and the code.

I decided to keep the threshold for precision at 0.9. Any score greater than 0.9 was considered "precise." Using this rule, I could compute the hit rate for each category of message and answer the three research questions (RQ1–RQ3). The results were saved to `ollama_scores_codebert.csv`. The following image shows the code implementation for the evaluation:

```python
from transformers import RobertaTokenizer, RobertaModel

MODEL_NAME = "microsoft/codebert-base"
tokenizer = RobertaTokenizer.from_pretrained(MODEL_NAME)
model = RobertaModel.from_pretrained(MODEL_NAME).to(device)


def get_code_embedding(code_snippet):
    tokens = tokenizer(code_snippet, return_tensors="pt", truncation=True, padding=True, max_length=512)
    inputs = {key: val.to(device) for key, val in tokens.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    # Use the [CLS] token representation as the embedding
    cls_embedding = outputs.last_hidden_state[:, 0, :]
    return cls_embedding

def cosine_sim(vec1, vec2):
    return F.cosine_similarity(vec1, vec2).item()

def score(code, msg):
    if not msg.strip() or not code.strip():
        return 0
    code_emb = get_code_embedding(code)
    msg_emb = get_code_embedding(msg)
    return cosine_sim(code_emb, msg_emb)

def evaluate_with_codebert(df):
    print("Evaluating with CodeBERT...")
    print("\nScoring original commit messages...")
    df["dev_score"] = df.progress_apply(lambda r: score(r["Diff"], r["Message"]), axis=1)
    print("\nScoring LLM inference messages...")
    df["llm_inference_score"] = df.progress_apply(lambda r: score(r["Diff"], r["LLM Inference (fix type)"]), axis=1)
    print("\nScoring rectified messages...")
    df["rectifier_score"] = df.progress_apply(lambda r: score(r["Diff"], r["Rectified Message"]), axis=1)
    return df
```

Code for Evaluation

	Hash	File Name	dev_score	llm_inference_score	rectifier_score
0	014a277a97759bbc0e6ec8fba588bc6e6de65a86	constants.py	0.962658	0.942259	0.976323
1	014a277a97759bbc0e6ec8fba588bc6e6de65a86	displayer.py	0.918488	0.902997	0.941550
2	014a277a97759bbc0e6ec8fba588bc6e6de65a86	mobject.py	0.937761	0.921378	0.959316
3	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	__init__.py	0.942789	0.933025	0.963519
4	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	__init__.py	0.970627	0.962571	0.969677
5	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	animation.py	0.953300	0.954102	0.969746
6	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	meta_animations.py	0.897986	0.911280	0.977757
7	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	simple_animations.py	0.892042	0.898379	0.903630
8	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	transform.py	0.889994	0.870787	0.966823
9	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	displayer.py	0.911511	0.924990	0.937381
10	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	extract_scene.py	0.913572	0.917471	0.941413
11	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	helpers.py	0.889769	0.906124	0.927119
12	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	image_mobject.py	0.907444	0.911569	0.917670
13	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	images2gif.py	0.883028	0.872721	0.963090
14	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	mobject.py	0.959424	0.955415	0.977553
15	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	__init__.py	0.968253	0.959519	0.970230
16	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	complex_multiplication_article.py	0.900738	0.912112	0.918707
17	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	generate_logo.py	0.955927	0.955502	0.972877
18	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	moser_main.py	0.975104	0.965233	0.985425
19	2e074afb60d13262ce1e42e83bcf0ed28d95ad82	region.py	0.954561	0.950039	0.982617

CodeBERT Scores

```
RQ1 - Developer precise hit rate: 91.28%
RQ2 - LLM precise hit rate: 93.39%
RQ3 - Rectifier precise hit rate: 100.00%
```

Mean Hit Rates

We can see that the developer messages had a precision hit rate of 91%, the LLM inference messages had a hit rate of 92%, and the rectified messages improved the hit rate to 100%. This indicates that the rectifier was effective in enhancing the alignment between the messages and the code changes.

1.3 Results and Analysis

Bug-fixing commits found: 1358
File-level diffs extracted: 2041

Following are few examples of developer messages, LLM inferences, and my rectified messages:

Developer Message	**LLM Inference**	**Rectified Message**
A few fixes to initial point_thickness implementation	add missing constants	[constants.py]: Fix default point thickness to be 4 in DEFAULT_POINT_THICKNESS by specifying it explicitly.
A few fixes to initial point_thickness implementation	add nudge to displayer.py	[displayer.py]: Fix potential offset issue in point_thickness by adjusting the thickness of a plus-sign-shaped pixel arrangement to ensure correct rendering on high-quality displays.
A few fixes to initial point_thickness implementation	add missing docstring	[mobject.py]: Fix inconsistent point_thickness implementation in Mobject1D and Mobject2D by specifying a default value for the attribute.

RQ	**Message Type**	**Hit Rate (threshold = 0.9)**
RQ1	Developer messages	∼91%
RQ2	LLM inference	∼93%
RQ3	Rectified messages	100%

Developer messages were often short and lacked detail, lowering their alignment scores. The LLM inference was generally good but sometimes missed context that the rectifier captured.

My rectifier is able to consistently produce high-quality, precise messages. This is mainly because of 3 reasons:

1. **Better Large Language Model (LLM):** The use of a more advanced LLM for inference likely contributed to the improved message quality. The LLM was able to better understand the context and nuances of the code changes, resulting in more accurate and relevant messages.
2. **Better Context:** By using the specific file name and change type as part of the input, the rectifier can generate messages that are more closely aligned with the actual code changes being made. This helps to reduce ambiguity and improve precision.
3. **Structured Output:** The output format of the rectifier is designed to be concise and specific, which helps to ensure that the messages are clear and actionable.

1.4 Discussion and Conclusion

1.4.1 Challenges faced

During the lab, I faced a few challenges that slowed me down initially. PyDriller, for example, was a new library for me, and I needed some time to get comfortable with its API and how to extract the right commit-level information. Another difficulty came from the keyword-based heuristic I used to identify bug-fixing commits. While it worked reasonably well, it sometimes missed commits where developers didn't explicitly mention bug-related terms, and on the other hand it also pulled in a few extra commits that weren't true bug fixes. Finally, token limits posed a practical issue – some of the larger diffs had to be truncated before being passed to the model, and this occasionally hurt the accuracy of the LLM's generated summaries.

1.4.2 Lessons learned

Working through these issues taught me a few important lessons. I found that analyzing changes at the file level and then applying rectification added real value, because it made commit messages more precise and easier to interpret. I also realized that building a pipeline by combining several tools – PyDriller for mining, Hugging Face models for inference, Ollama for rectification, and CodeBERT for evaluation – can be powerful, but it also demands a lot of care in data handling. Even small oversights, like inconsistent CSV column names, can break later steps in the workflow.

1.4.3 Conclusion

Overall, the end-to-end pipeline – from mining commits to generating diffs, running LLM inference, rectifying the outputs, and finally evaluating them – proved to be quite workable. The rectifier in particular helped improve the quality of commit messages, often making them more specific and useful than both the original developer-written messages and the raw LLM predictions.

1.5 References

[1] PyDriller
[2] Hugging Face Transformers
[3] CodeBERT (microsoft/codebert-base)
[4] CommitPredictorT5 (mamiksik/CommitPredictorT5)
[5] Ollama
[6] Repository analyzed (manim)
[7] Lab Document: Google Doc