CMPT470

Winter 2025

Project Title: Fine-Tuning LLMs for Automated Bug Classification

Group: Lower Back
Members: Princess Tayab, Ardalan Askarian, Timofei Kabakov and Marmik Patel
Lead by: Ajmain Inqiad Alam

Related Papers

1. Javed, M. Y., & Mohsin, H. (2012, July). An automated approach for software bug classification. In *2012 sixth international conference on complex, intelligent, and software intensive systems* (pp. 414-419). IEEE.Link: https://ieeexplore.ieee.org/abstract/document/6245635

2. Otoom, A. F., Al-jdaeh, S., & Hammad, M. (2019, August). Automated classification of software bug reports. In *proceedings of the 9th international conference on information communication and management* (pp. 17-21). Link: https://dl.acm.org/doi/abs/10.1145/3357419.3357424

**** This plan is subjective to change. It can be modified based on the progress.****

Summary

This project develops a fine-tuned large language model (LLM) for automated bug classification using GitHub bug reports. Since GitHub labels issues only as "bug" without specifying the type of bug, we will:

1. Fetch open bug-tagged issues from various GitHub repositories using the GitHub REST API.
2. Manually classify data into specific bug categories, ensuring high-quality labels.
3. Evaluate inter-rater agreement to measure consistency between human annotators in the manual classification phase.
4. Train domain-specific LLMs (e.g., CodeBERT, GraphCodeBERT, CodeT5) to automate bug classification.

5. Analyze bug frequency patterns across different repositories to gain insights into recurring software issues.

Dataset

Data Source

- GitHub Issues from public repositories using the GitHub REST API.
- Filtering Criteria:
  - Issues labeled with "bug".
  - Exclude feature requests, enhancement discussions, or unrelated issues.
  - Focus on active/popular different types of repositories. For example, it can be AI, Front-end, Tool or other Software engineering related repos.
  - Let's first focus on open issues.

Target Size

- 1,000 – 5,000 bug reports collected.
- Time Frame: Extract issues from the past 3–5 years to ensure relevance (If we can impose this time frame).

Bug Categories (Manually Labeled for Training Data)

Category	Description	Common Indicators
Syntax Error	Compilation or syntax-related failures	`"SyntaxError"`, `"unexpected EOF"`, `"invalid syntax"`
Runtime Error	Code crashes unexpectedly	`"RuntimeError"`, `"TypeError"`, `"NullPointerException"`
Performance Issue	Slow execution, high memory/CPU usage	`"slow execution"`, `"memory leak"`, `"CPU usage 100%"`
Security Vulnerability	Exploitable flaws in authentication, access control, etc.	`"XSS"`, `"SQL Injection"`, `"unauthorized access"`

Logical Bug	Incorrect implementation or unexpected behavior	`"wrong output"`, `"unexpected result"`, `"logic error"`
Dependency Issue	Missing or conflicting dependencies	`"ImportError"`, `"module not found"`, `"dependency conflict"`
UI/UX Bug	Frontend interaction or accessibility problems	`"UI glitch"`, `"alignment issue"`, `"button not working"`

Preprocessing

- **Data Cleaning:**
    - Remove duplicates, incomplete issues, spam.
- **Feature Extraction:**
    - Metadata: Extract title, description, labels, timestamps, affected components and code snippets.

Methodology

1. Data Collection & Storage

- **Fetch GitHub Issues:**
    - Use the GitHub REST API to collect structured bug reports.
    - Extract title, description, labels, timestamps, comments count, reactions, and code snippets (if available).
- **Storage:**
    - Save the data in CSV and JSON formats for structured analysis.

2. Manual Bug Classification

- Manually classify bug reports into predefined categories.
- Inter-Rater Agreement Analysis:
    - Assign multiple annotators to label the same subset of bug reports.

- Use Cohen's Kappa to measure annotation consistency.
- If agreement is low, refine classification guidelines before labeling the full dataset.

3. Training a Bug Classification Model

3.1. Domain-Specific LLMs for Bug Classification

- **Fine-Tune Transformer Models**:
    - **CodeBERT** → Pretrained transformer on code + text.
    - **GraphCodeBERT** → CodeBERT variant incorporating data flow.
    - **CodeT5** → Code-aware model with text-to-code capability.

3.2. Model Evaluation Metrics

- **Accuracy & F1-score** → Overall classification performance.
- **Confusion Matrix** → Identify misclassification patterns.
- **Per-Class Precision/Recall** → Evaluate effectiveness for each bug type.
- **Error Analysis** → Identify failure cases and refine training.

4. Bug Analysis Across Repositories

- Analyze bug distribution across different types of projects.
- Identify trends in recurring software issues across different domains.

Evaluation Metrics

- **Inter-Rater Agreement:** Cohen's Kappa for manual labeling consistency.
- **Model Classification Performance:** F1-score, Precision, Recall, Accuracy.
- **Model Robustness:** Error rate on different bug types.

Expected Outcomes

- High-quality labeled dataset of GitHub bug reports.
- Fine-tuned LLM that automates bug classification with high accuracy.
- Insights into bug distribution across software projects.

- Potential integration of automated bug classification into developer workflows (e.g., GitHub Actions).

Refined Research Questions

1. **RQ1:** What is the accuracy of a fine-tuned domain-specific LLM in classifying bug reports into predefined categories?
2. **RQ2:** Which bug types (e.g., syntax errors, performance issues) are harder for domain-specific LLMs to classify accurately?
3. **RQ3:** How do different repositories compare in terms of bug frequency and type distribution?
