ML in Static Analysis

Goal

This project aims to detect and filter false positive results of clang static-analyzer using ML methods to predict which warnings are FP and which are TP

In this presentation we will discuss our strategy to achieve this

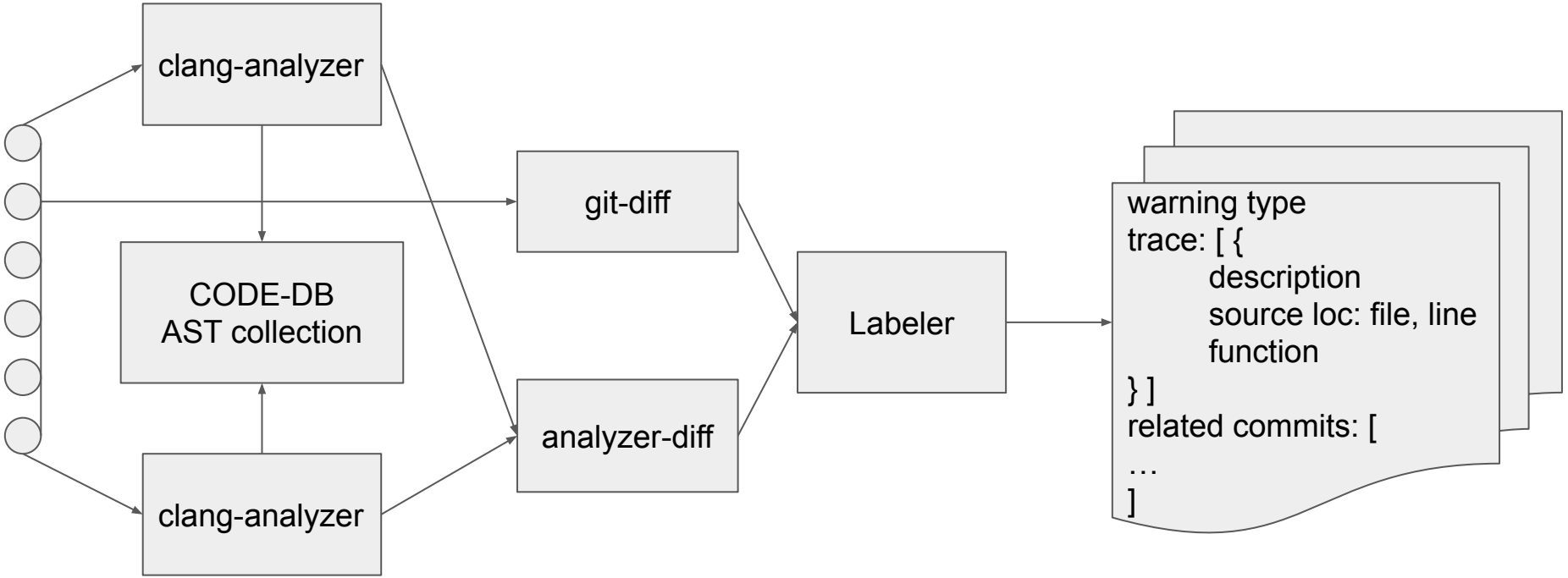
Data collection strategy

If we denote the issues found in the before-commit version as I_1 and the ones in the corresponding after-commit version as I_2, all issues can be classified into three groups: the fixed issues, that are detected in the before-commit version but disappear in the after-commit version, the pre-existing issues that are detected in both versions, and the introduced issues that are not found in the before-commit versions but detected in the after-commit version. We are particularly interested in the fixed issues because they are very likely to be bugs fixed by the commit. And those issues that remain are the potential FPs

Data collection strategy

So, we will take repository and do the following:

1. take released project
2. checkout to the middle of the development process
3. run clang static-analyzer on project and save all warnings
4. checkout to the subsequent release version and run static analyzer again


clang-analyzer
git-diff
CODE-DB
AST collection
Labeler
warning type
trace: [{
description
source loc: file, line
function
}]
related commits: [
…
]
analyzer-diff
clang-analyzer

What do we have

After these steps we will have all the info about each warning:

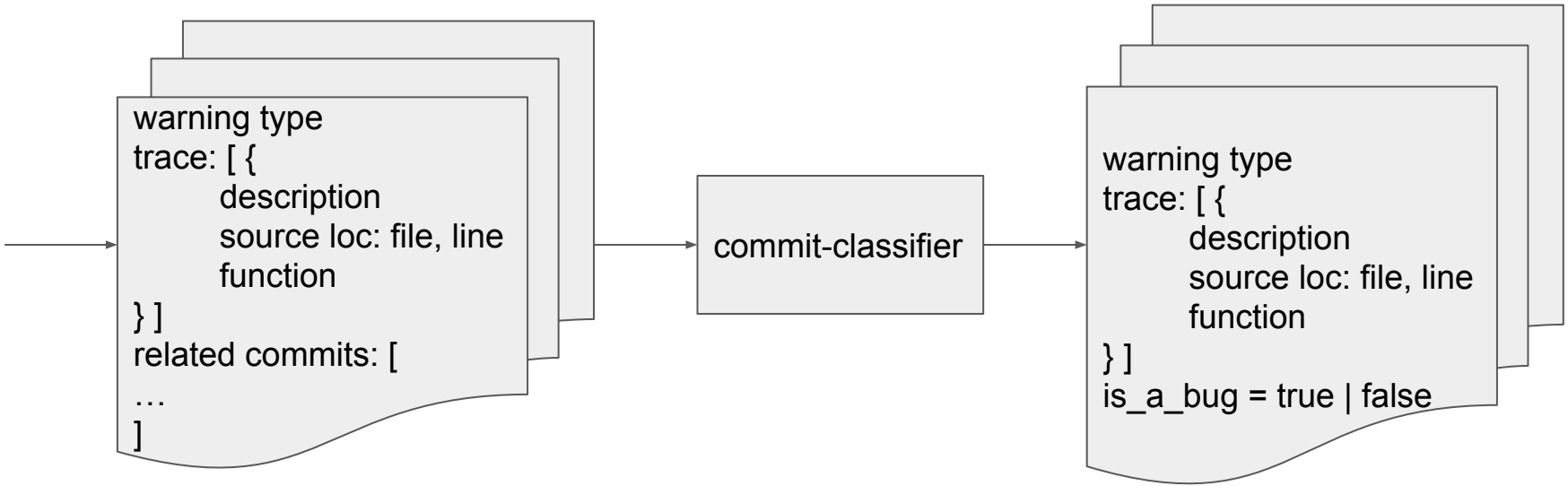
1. Warning type
2. Warning trace
3. Related commits id list

What will we do next

After obtaining dataset, we will need to train our model

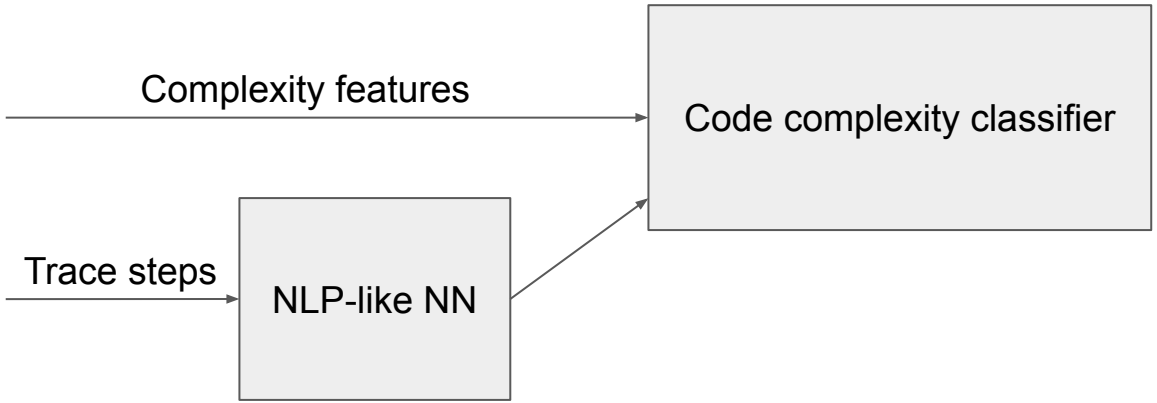
In order to work with dataset, we will need to mark warnings as FP or TP. We can do it manually, but it will be a long and tedious process. In order to avoid that, we suggest attempting to mark warnings using commit messages. F.ex. if there is commit that changes given block of code with “fix” in it’s text, and in latest version of project warning is missing, we will mark initial warning as TP. otherwise, it’s FP

Commit classifier will be a simple NN to find “bugfix” commit messages


warning type
trace: [{
description
source loc: file, line
function
}]
related commits: [
…
]
commit-classifier
warning type
trace: [{
description
source loc: file, line
function
}]
is_a_bug = true | false

Obtaining features

1. Code complexity metrics:
    a. line length, indentation depth, number of lines in file, number of operators in single line
2. Warning trace length, trace messages
3. Small AST pieces | nearest tokens


Complexity features
Code complexity classifier
Trace steps
NLP-like NN

Thanks for your attention