Using Character-Based Git Blame Information to Enhance the Precision of Commit-Interaction Analysis

09.04.2024

Leonie von Mann

Sebastian Böhm

Florian Sattler

Prof. Dr. Sven Apel

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

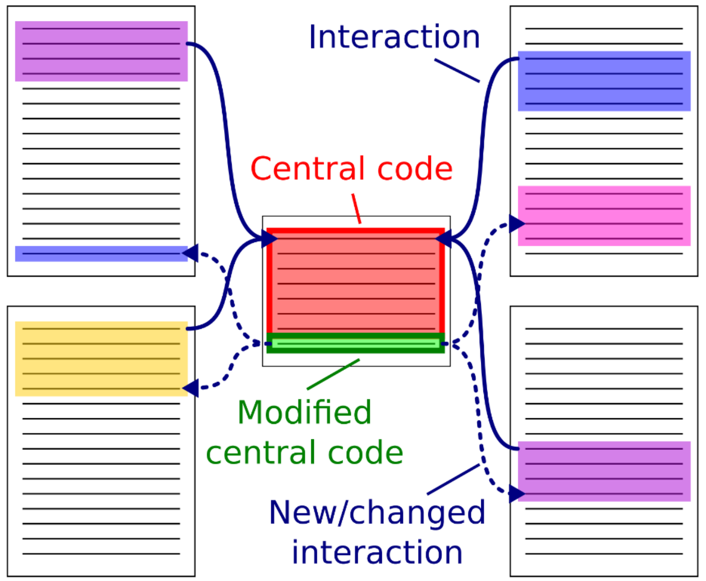
Motivation

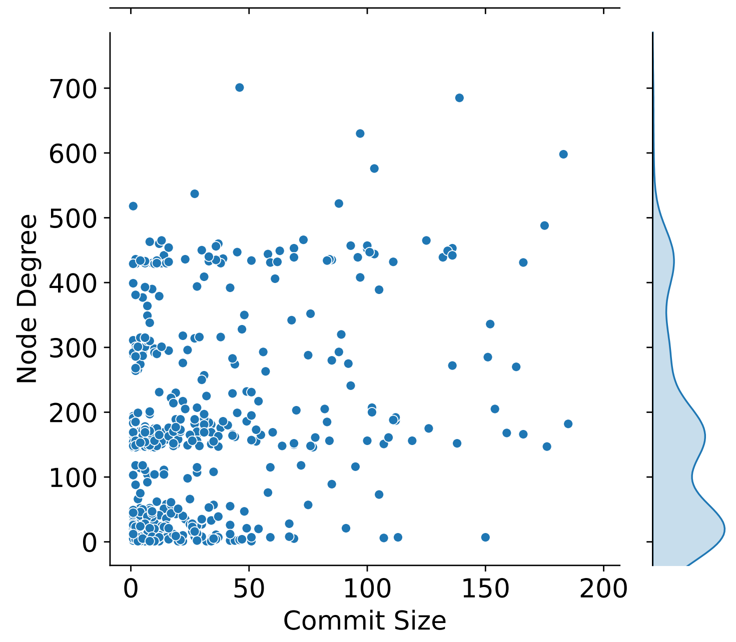
```
git blame:
```

- Which commit last modified each line
- Used in *commit-interaction analyses:*

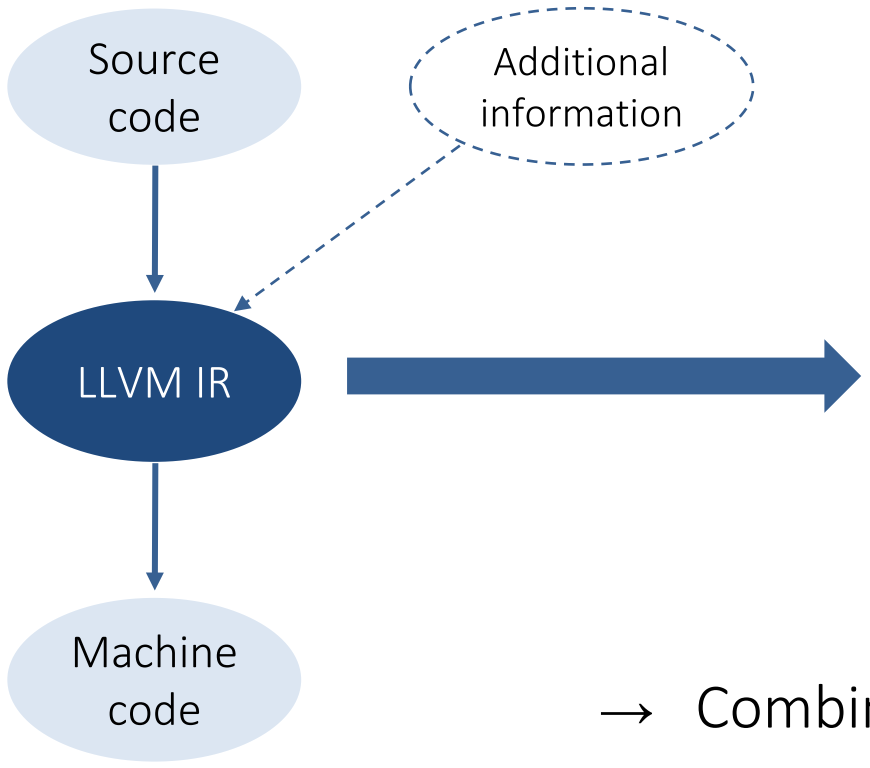
  Commit interactions within a programs data-dependency structure

Example: Central-code analysis





Motivation



Program analyses often work on

LLVM intermediate representation (IR)

→ Combine repository information with

low-level program information

Example: Arithmetic Expression

Commit A

```
+   int foo(int x) {
+       int result;
+       result = x + 42;
+       return result;
+   }
```

Commit B

```
    int foo(int x) {
        int result;
-       result = x + 42;
+       result = x + 42 - 1;
        return result;
    }
```

Example: Arithmetic Expression

Source Code	Line-Based Blame
`int foo(int x) {`	→ A
`    int result;`	→ A
`    result = x + 42 - 1;`	→ B
`    return result;`	→ A
`}`	→ A

Example: Arithmetic Expression

Source Code

```
int foo(int x) {
    int result;
    result = x + 42 - 1;
    return result;
}
```

LLVM IR

```
%1 = load i32, i32 %x.addr
%add = add nsw i32 %1, 42
%sub = sub nsw i32 %add, 1
store i32 %sub, i32* %result
```

Example: Arithmetic Expression

Source Code

```
int foo(int x) {
   int result;
   result = x + 42 - 1;
   return result;
}
```

LLVM IR

```
%1 = load i32, i32 %x.addr          → B
%add = add nsw i32 %1, 42           → B
%sub = sub nsw i32 %add, 1          → B
store i32 %sub, i32* %result        → B
```

Example: Arithmetic Expression

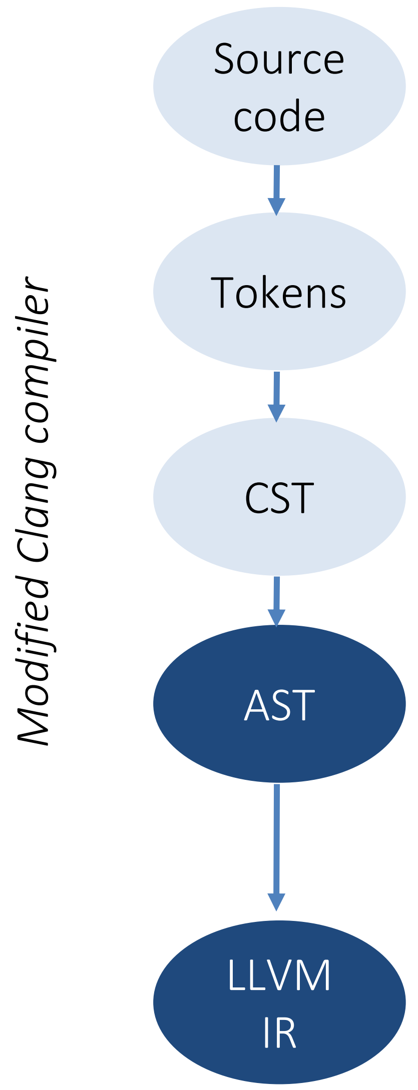
Source Code

```c
int foo(int x) {
    int result;
    result = x + 42 - 1;
    return result;
}
```

LLVM IR

```
%1 = load i32, i32 %x.addr          → A
%add = add nsw i32 %1, 42           → A
%sub = sub nsw i32 %add, 1          → B
store i32 %sub, i32* %result        → A
```

Implementation



Make blame annotations to LLVM IR

Character-based blame computation

Example: Arithmetic Expression

Find blame for **x** (line 3, column 12)

```
1   int foo(int x) {              → A
2       int result;               → A
3       result = x + 42 - 1;      → B
4       return result;            → A
5   }                             → A
```

↳ CurrentBlame := B
↳ Parent(B) = A

↳ Compare commit B to A at line 3, column 12

```
A: result = x + 42;
B: result = x + 42 - 1;
```

↳ No change at the given location
↳ CurrentBlame := A

Evaluation

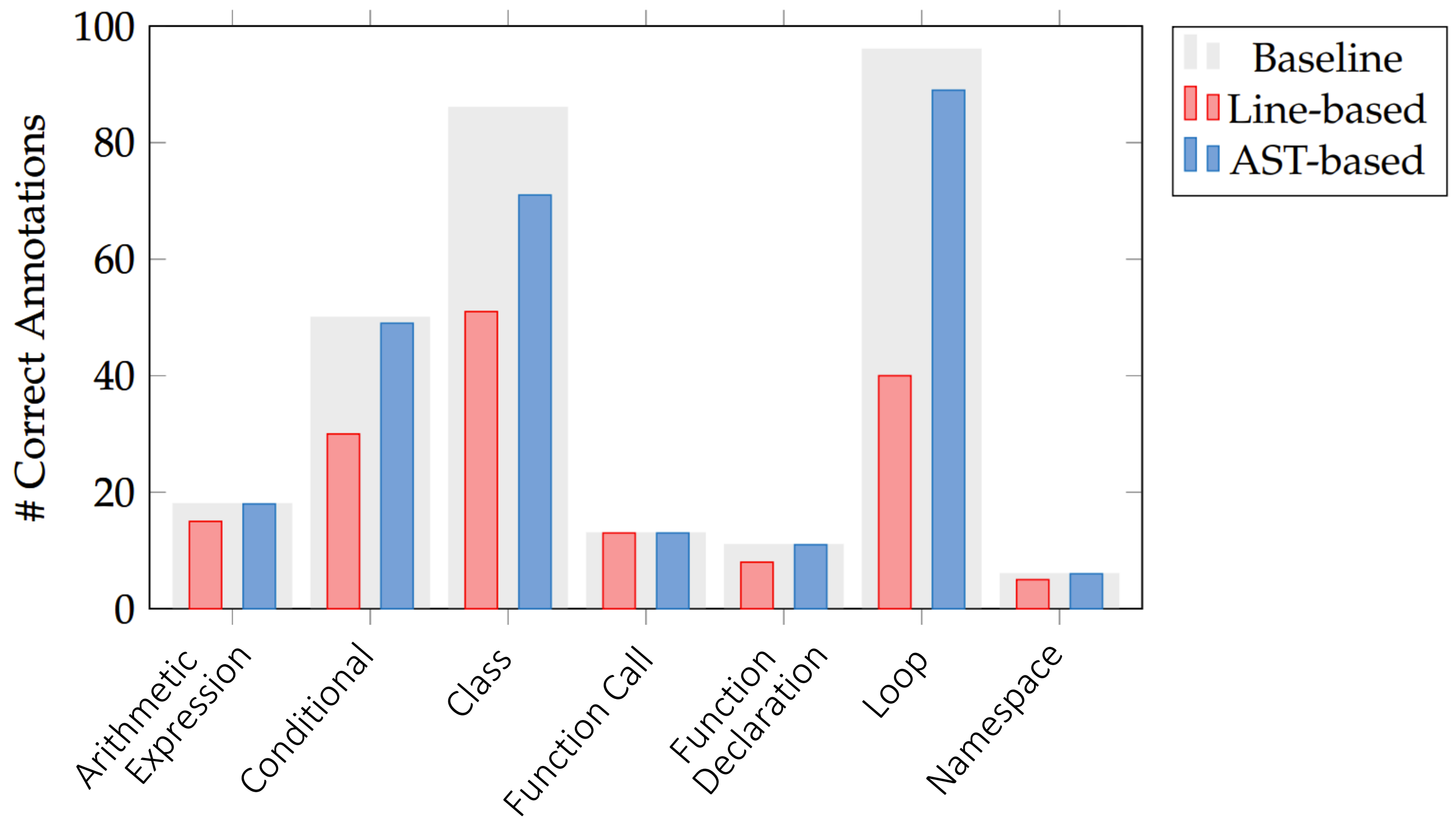
RQ1: How much do AST-based commit mappings improve blame information on an IR level?

RQ1.1: Which common code development scenarios benefit from AST-based commit mappings?

RQ1.2: How many instructions are mapped differently in real-world projects with AST-based annotations compared to the line-based blame?

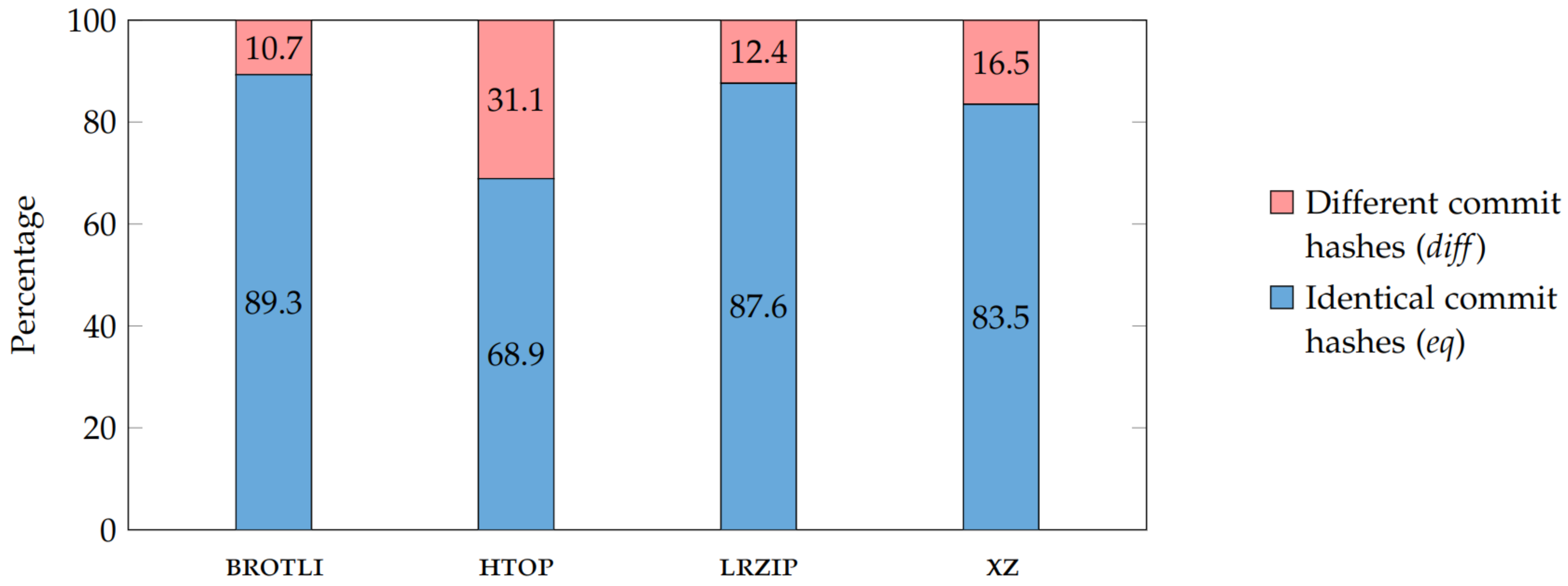
Evaluation

RQ1.1: Scenarios benefitting from AST-based commit mappings



Evaluation

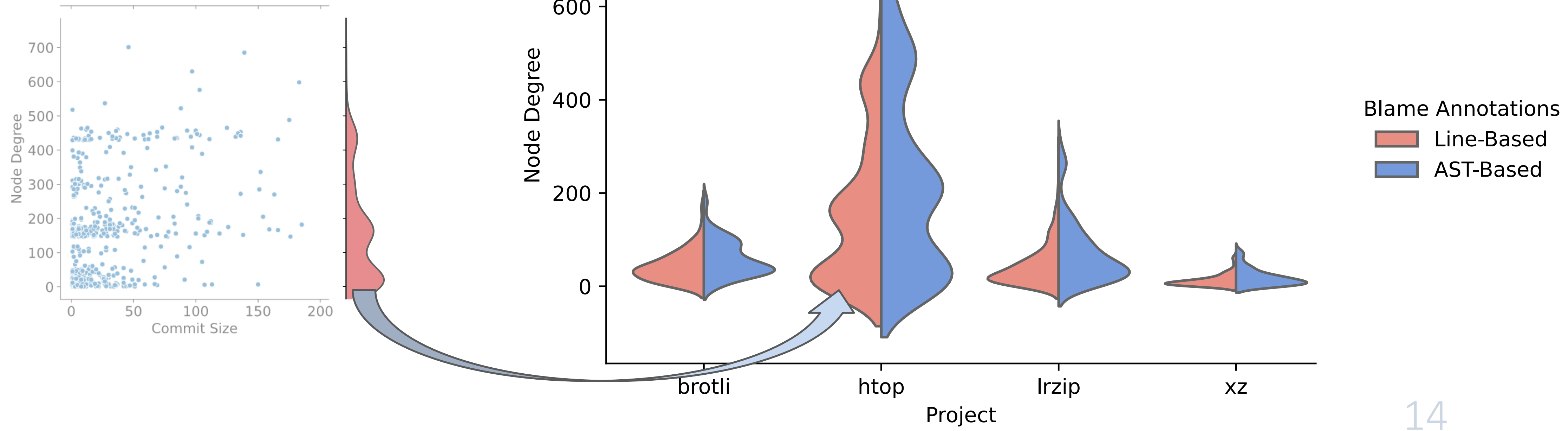
RQ1.2: Difference between AST-based and line-based blame



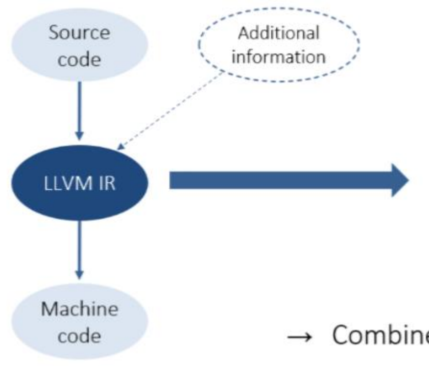
Evaluation

RQ2: What is the impact of AST-based blame information on commit-interaction analysis?

Central-code analysis



Motivation



Program analyses often work on

LLVM intermediate representation (IR)

→ Combine repository information with

low-level program information

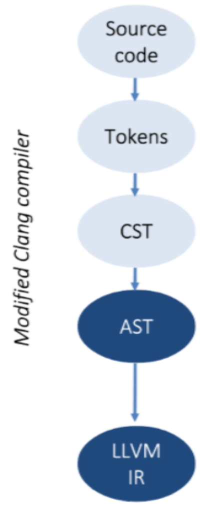
Example: Arithmetic Expression

Source Code	LLVM IR

```
int foo(int x) {
  int result;
  result = x + 42 - 1;
  return result;
}
```

```
%1 = load i32, i32 %x.addr
%add = add nsw i32 %1, 42
%sub = sub nsw i32 %add, 1
store i32 %sub, i32* %result
```

Implementation

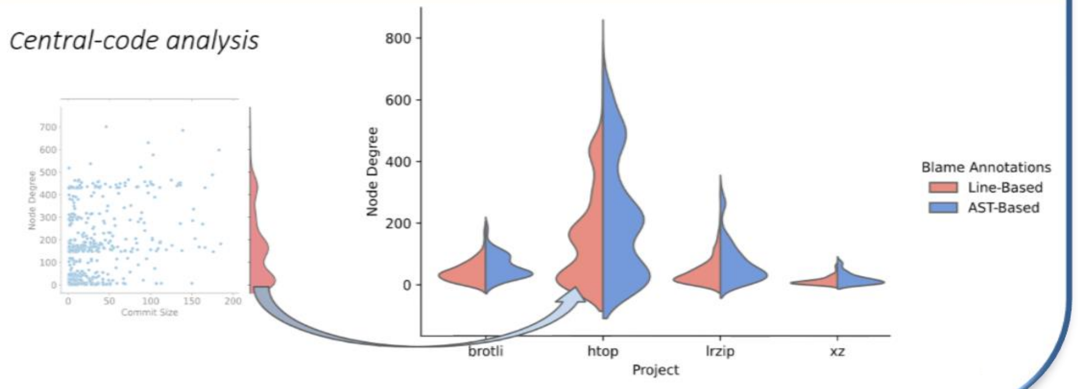


Make blame annotations to LLVM IR

Character-based blame computation

Evaluation

RQ2: What is the impact of AST-based blame information on commit-interaction analysis?


Central-code analysis
Node Degree
Commit Size
brotli
htop
lrzip
xz
Project
Blame Annotations
Line-Based
AST-Based