Comparing and Combining Analysis-Based and Learning-Based Regression Test Selection

Jiyang Zhang, Yu Liu, Milos Gligoric, Owolabi Legunsen, August Shi

The University of Texas at Austin, Cornell University

AST 2022

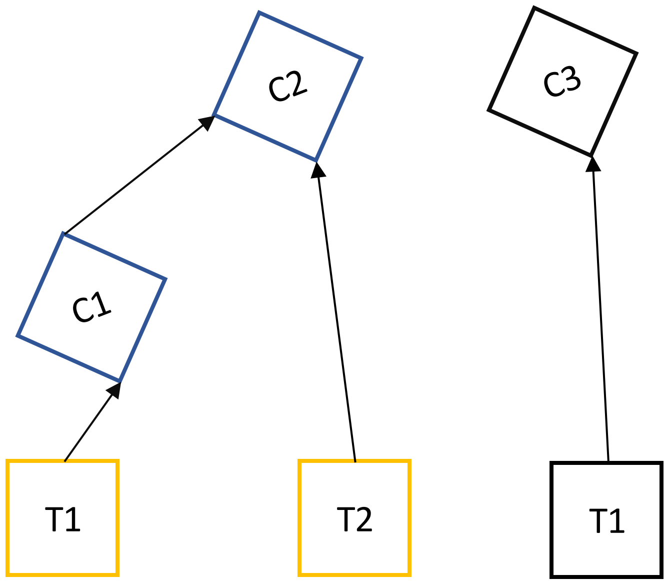
Partially supported by Google NSF TEXAS The University of Texas at Austin

Regression Test Selection (RTS)

- Regression testing is widely practiced by software developers
- Regression test selection (RTS) optimizes regression testing by only rerunning a subset of tests that can be affected by changes.
- Evaluate RTS:
  - Safe
  - Precise
  - Efficient

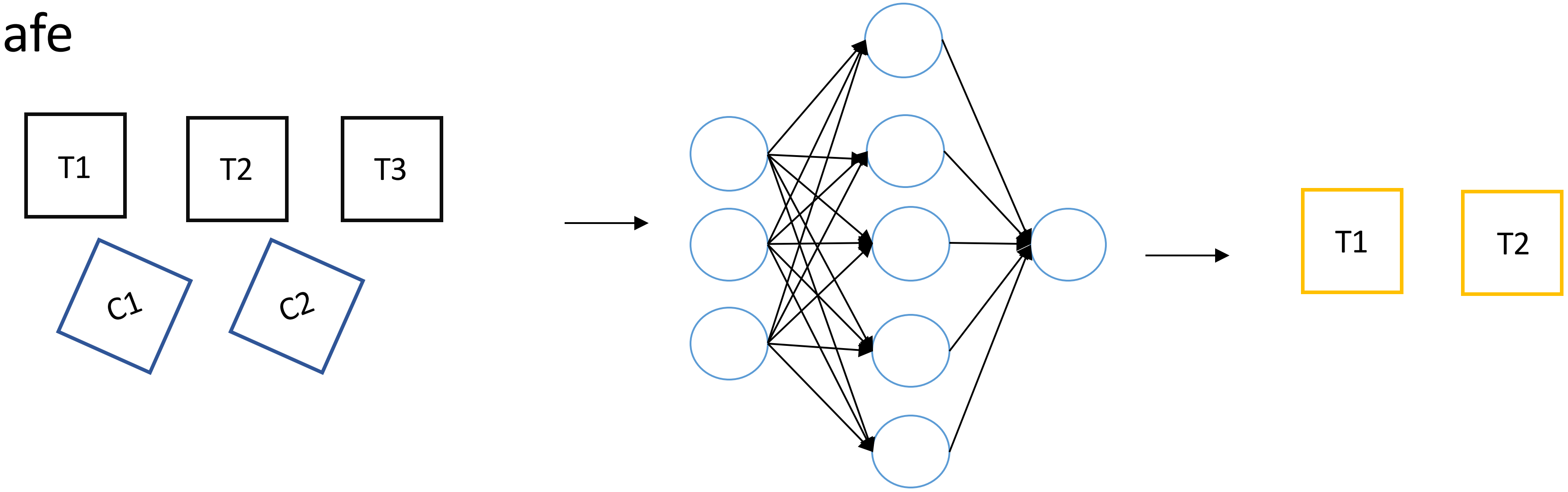
Analysis-based RTS

- RTS based on program analysis can save substantial testing time for (medium-sized) open-source projects.
  - Dynamic program analysis RTS (Ekstazi)
  - Static program analysis RTS (STARTS)
- Problems:
  - Costs
  - Imprecise
  - Large repository with multiple programming language

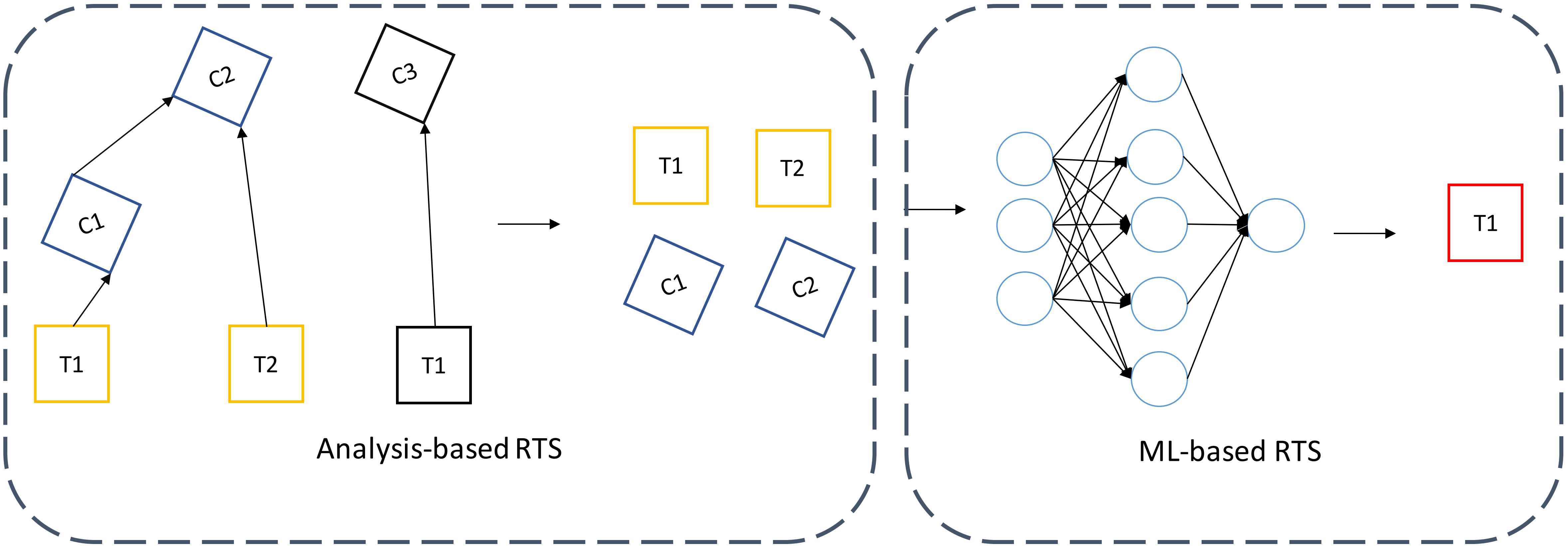


Machine Learning (ML) RTS

- ML-based RTS model learns from designed features
  - Predictive Test Selection (Machalica et al.)
- Problems:
  - Requires large training data
  - Unsafe



Combining



Our contributions

- Design and implement novel ML-based RTS models
  - Use mutation analysis to build training dataset
- Combine ML-based RTS with analysis-based RTS to improve precision
- Compare our approaches with prior analysis-based RTS and rule-based RTS

Outline

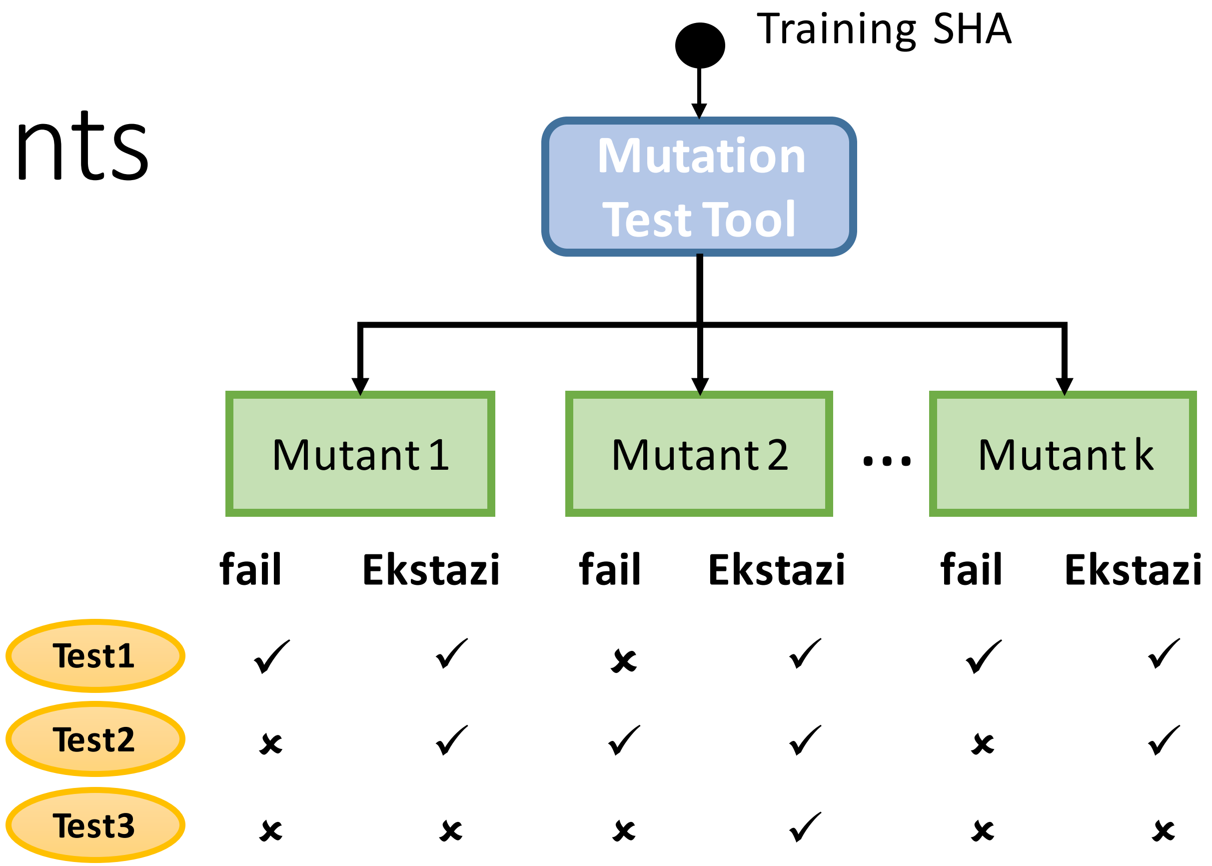
- Technique
  - Training Dataset from Mutans
  - Feature Extraction
  - Model Design
  - Model Variations
  - Combining Analysis-based RTS and ML-based RTS
- Empirical Study
  - Dataset
  - Experiments set up
  - Results and findings

Training Data from Mutants

- It is hard to obtain large data from open-source projects for training
- Utilize Mutation Testing (Pitest) to create Dataset

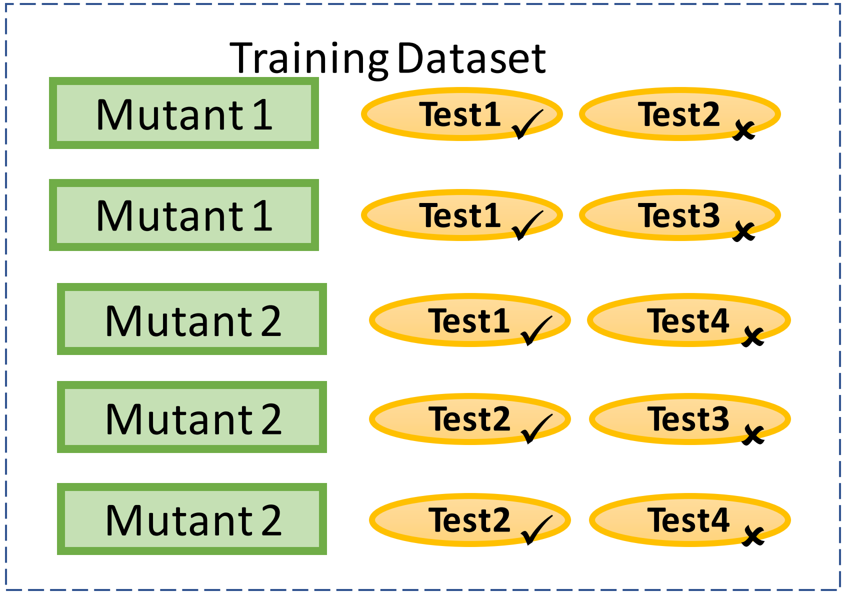
Training Data from Mutants

- Labeling the data in two ways:
  - A) Model predicts test failure:
    - Positive labels: test-mutants that are killed
  - B) Model predicts what Ekstazi selects
    - Positive labels: test-mutants that are selected by Ekstazi



Training Data from Mutants

- To overcome the data imbalance, training dataset instances are pairs of $\langle c, t^+, t^- \rangle$



Machine Learning Model

Feature Extraction

Feature Representation Learning

Likelihood Prediction

Feature Extraction

- Test:
  - split the test class name into tokens via *camelCase* and *snake_case*
  - convert to lower case

StrTokenizerTest.java → str | tokenizer | test

Feature Extraction

- Code diff:
  - Basic
    - Split changed class name into tokens via camelCase and snake_case and convert to lower case

StrTokenizer.java → str | tokenizer

  - Code
    - the sequence of tokens on added and modified lines

if (pos > 0) return true; → if | (| pos | > | 0 |) | return | true | ;

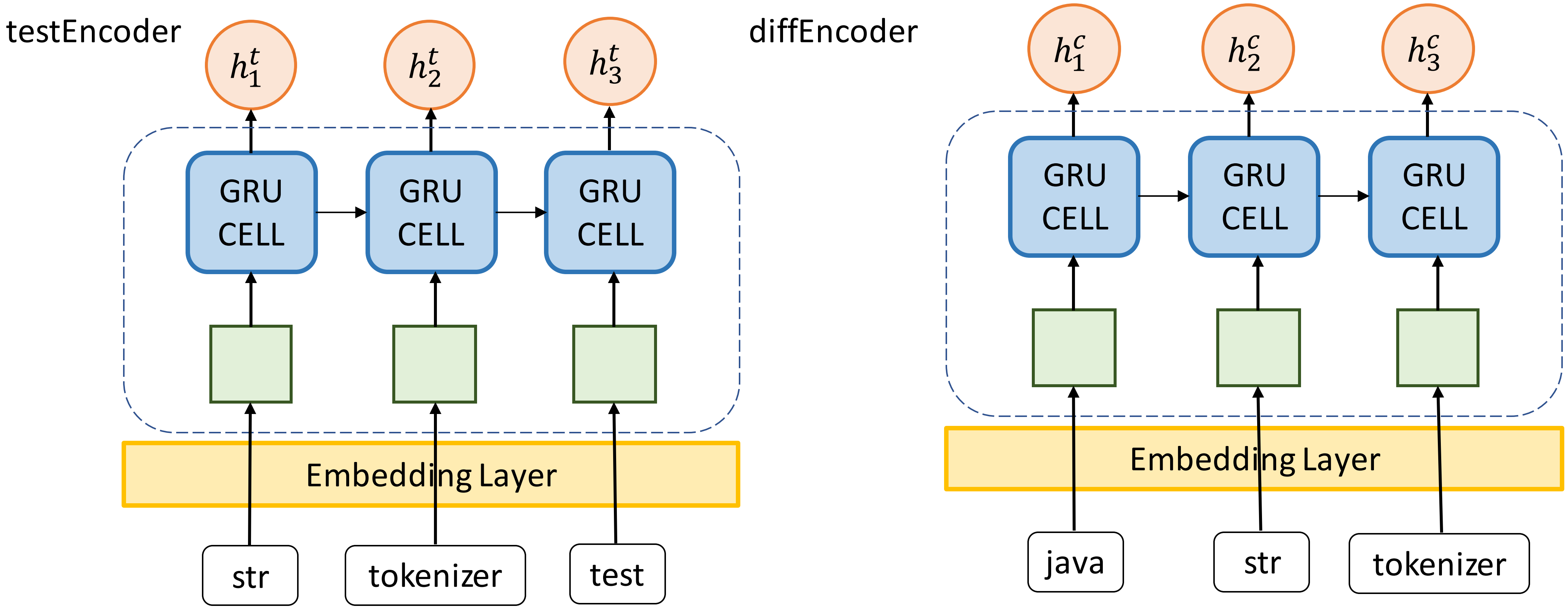
  - ABS
    - map parts of the changed lines of code to general operators

if (pos > 0) return true; → ConditionalBoundaryMutator | BooleanReturnValsMutator

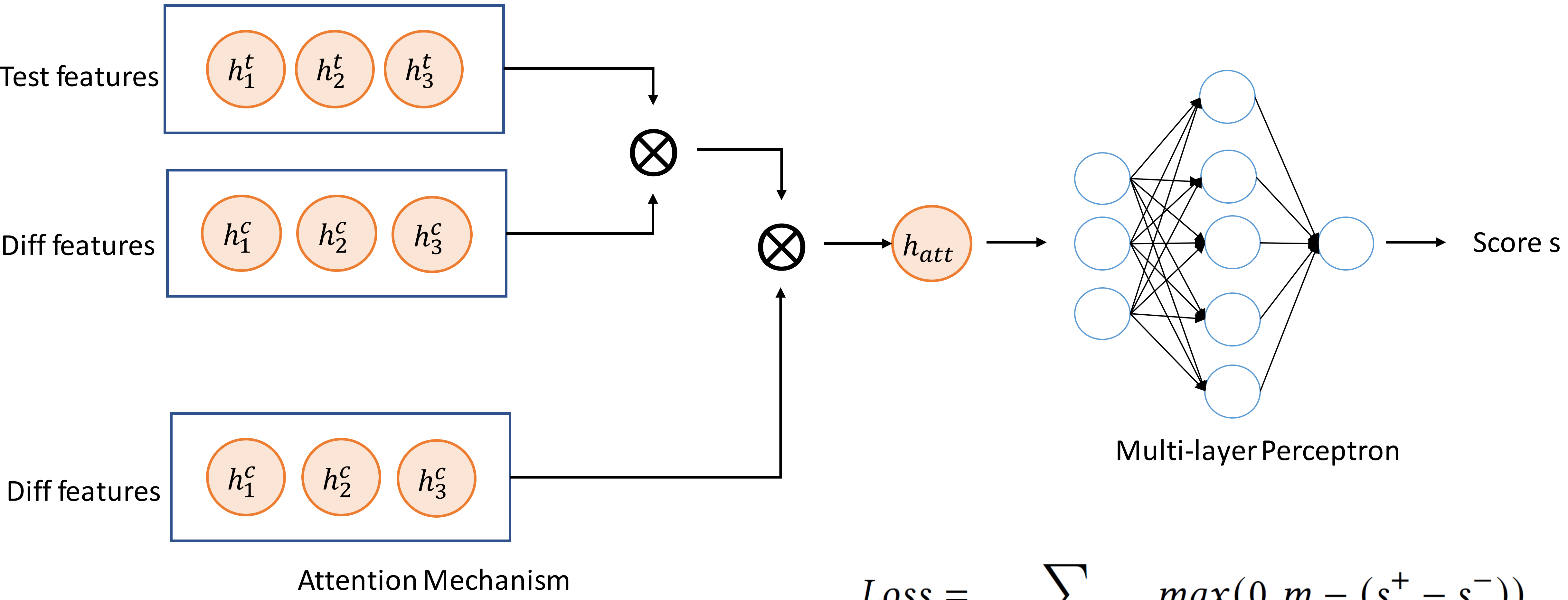
Feature Representation Learning

- Two Bi-drectional Gated Recurrent Unit encoders:
  - diffEncoder: encodes features from the code diff
  - testEncoder: encodes features from the tests

Feature Representation Learning

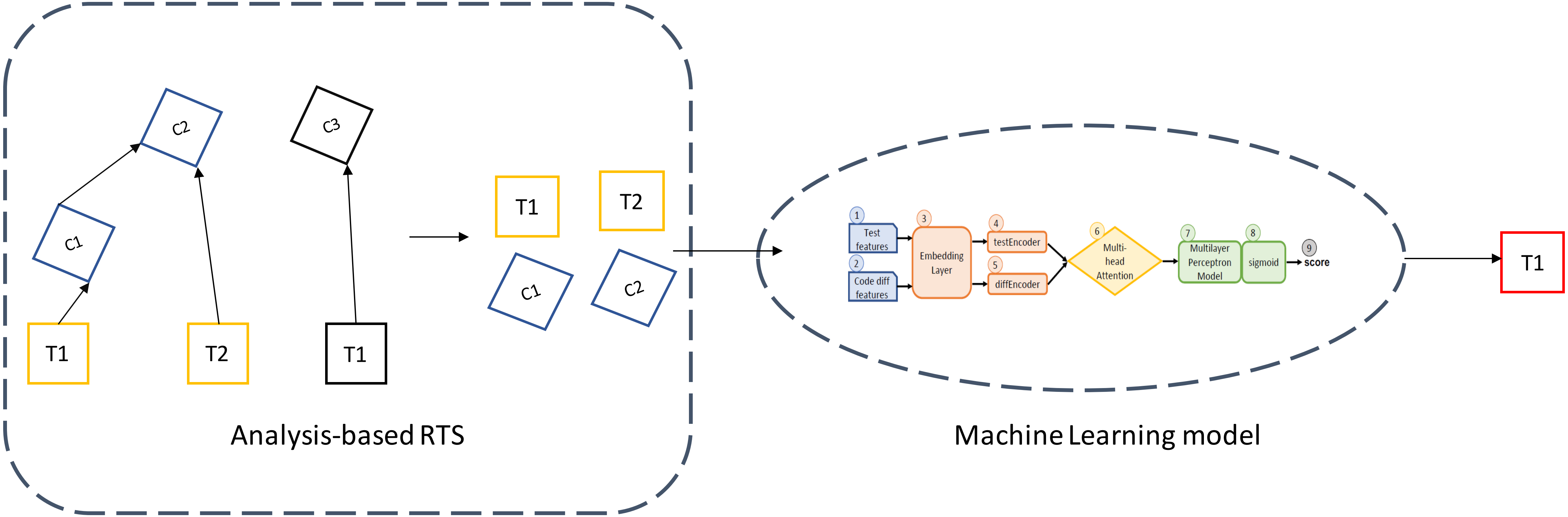


Feature Representation Learning



$$Loss = \sum_{\langle c, t^+, t^- \rangle \in D} max(0, m - (s^+ - s^-))$$

Combining Analysis-based RTS and ML-based RTS



Outline

- Technique
  - Overview
  - Training Dataset from Mutans
  - Feature Extraction
  - Model Design
  - Combining Analysis-based RTS and ML-based RTS
- Empirical Study
  - Baselines
  - Dataset
  - Experiments setup
  - Results and findings

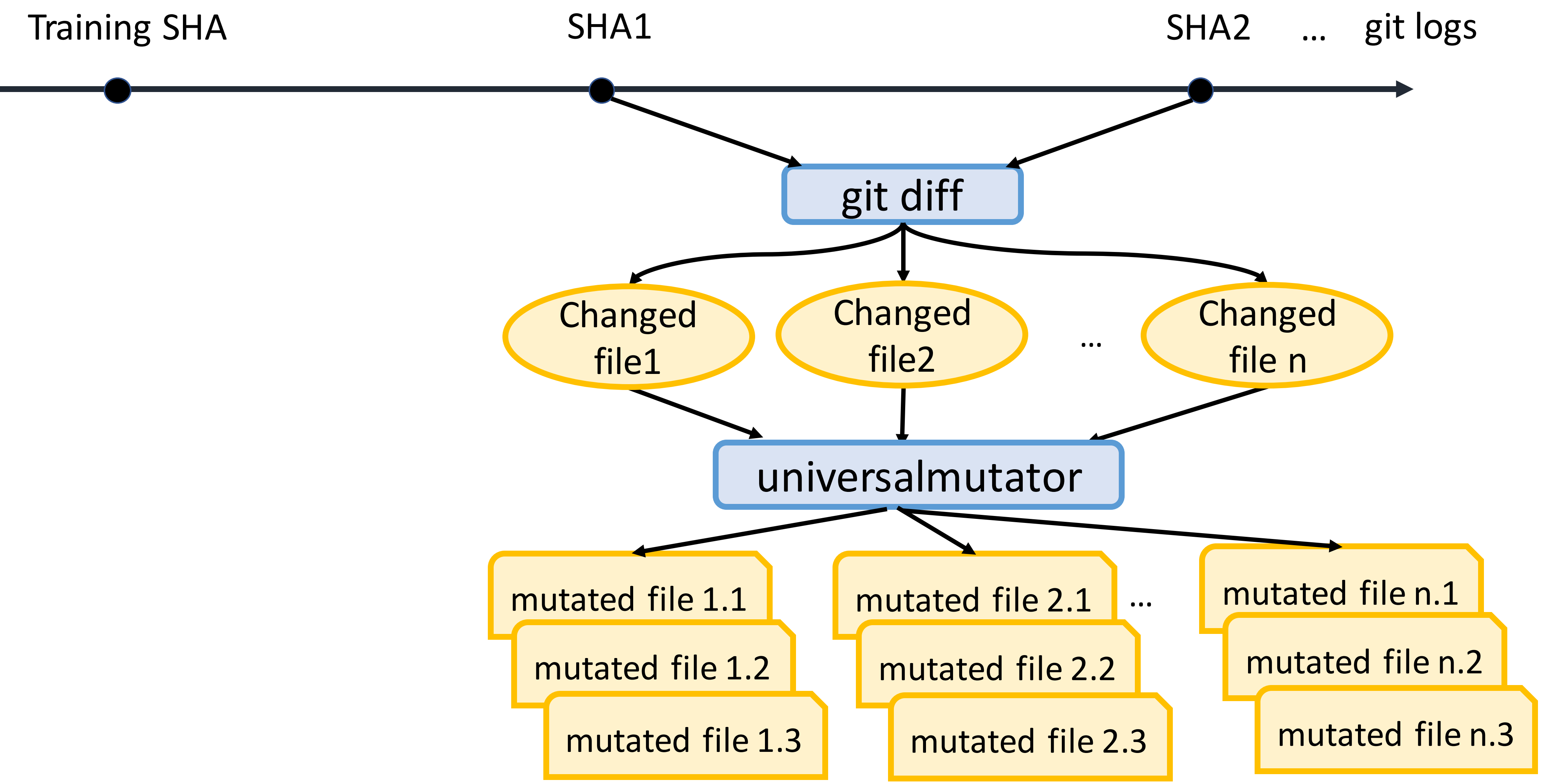
Baselines

- Information Retrieval-based RTS model: BM25
    - Rank tests based on their relevance to code changes by treating each test file as a document and the code changes as a query
    - The ranking is based on assigned scores to each test
- EALRTS
    - Features from a dependency graph extracted by STARTS and the project's Git commit history
    - Random Forest machine learning model

Dataset

- Training data
  - 10 open-source Java projects
  - Select a training commit without failures and can be run by Analysis-based RTS (Ekstazi, STARTS)
- Evaluation dataset:
  - The evaluation dataset for each project are collected by leveraging its real code evolution
  - Select commits that are after the training commit for evaluation
  - Failing tests are introduced by mutating the code at the evaluation commits

Evaluation Dataset Construction

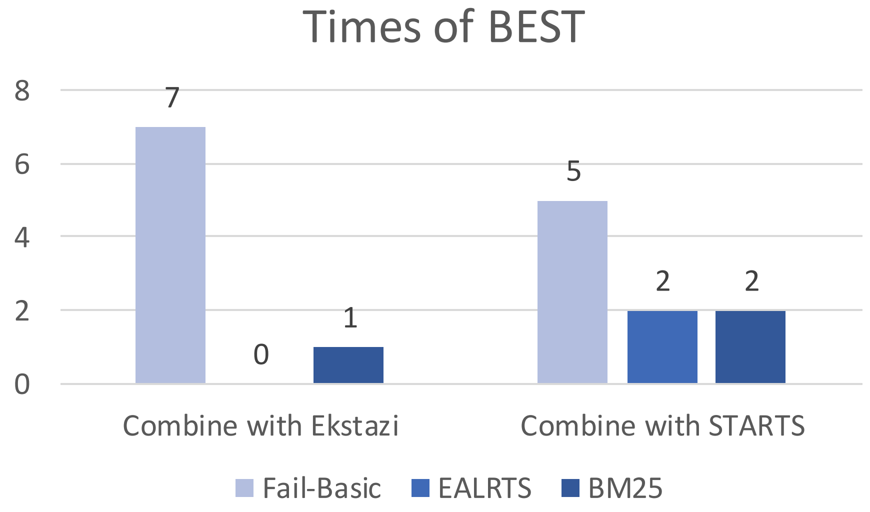


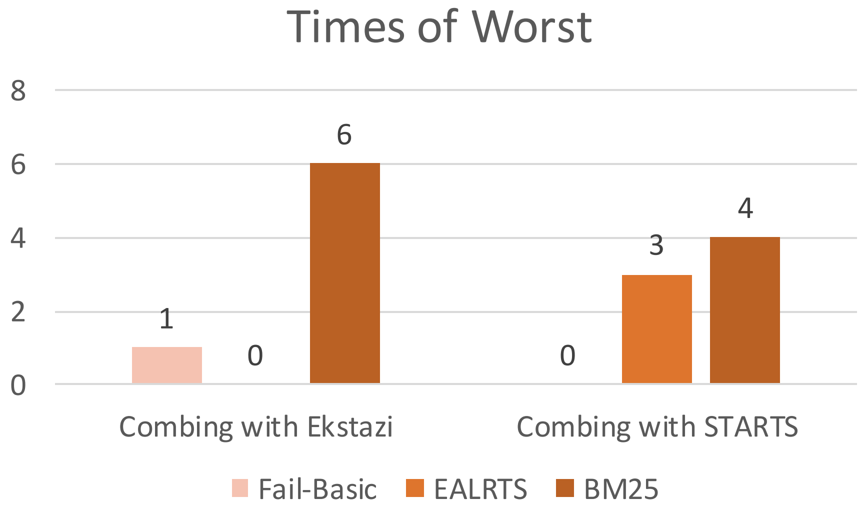
Experiments Setup

- Evaluation Setup
  - Run models to select the failing tests from the tests selected by analysis-based RTS tool
  - Compute the percentage of tests that the model would need to select to run all failing tests
  - Measure the overhead of combining Analysis-based RTS tool with ML-based RTS models
- Metrics
  - **Best safe selection rate:**
    - the largest selection rate needed to select all failing tests across all pairs of mutants and commits in each project (ensuring safe selection)
  - **End-to-end testing time:**
    - summation of the time to select tests and the time for running the selected tests such that all the failing tests are included in the selected test set

Results (1/3)

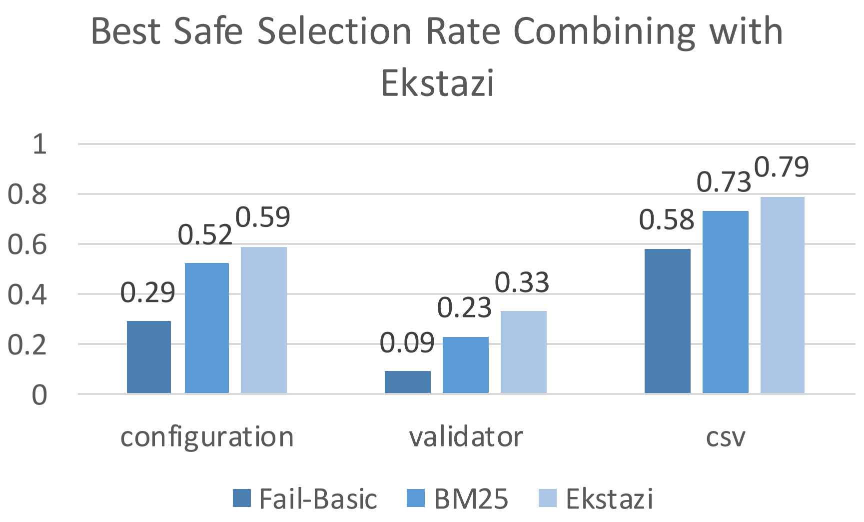
- There is no single ML-based RTS model that consistently outperforms or underperforms the rest
- Fail-Basic is the model with the highest number of **BEST**, better than all other ML-based RTS models and baselines when combining with analysis-based RTS models

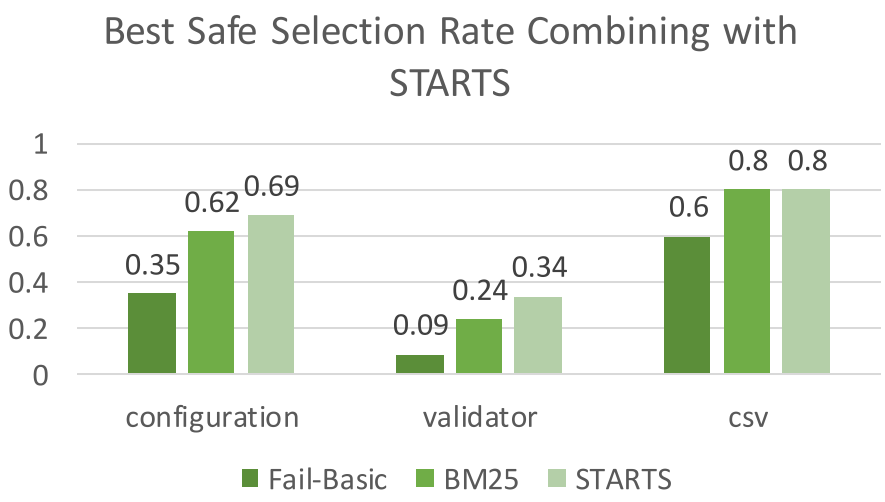




Results (2/3)

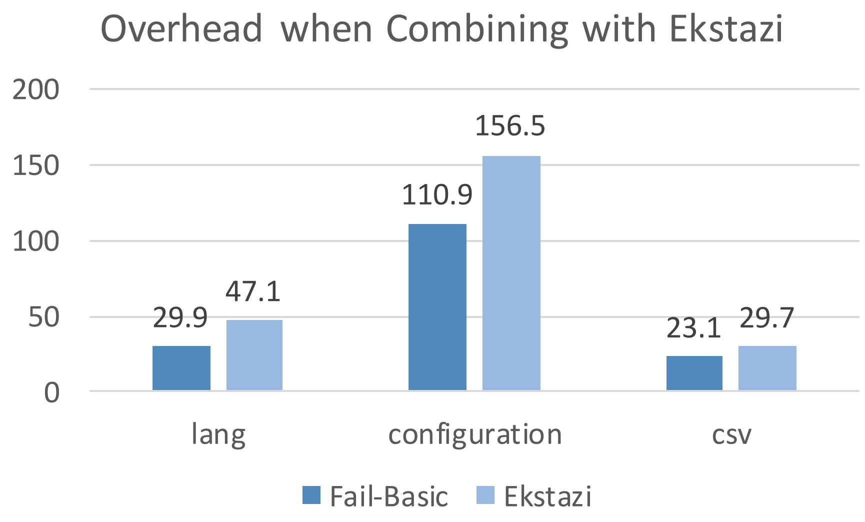
- Combining ML-based RTS with analysis-based RTS improves the precision of Ekstazi and STARTS

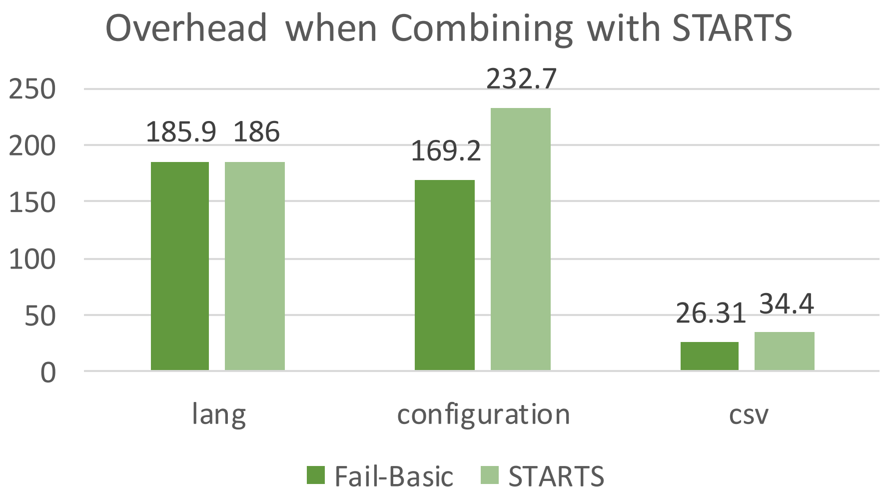




Results (4/4)

- ML-based RTS models combined with an analysis-based RTS technique for the most part outperform the corresponding analysis-based RTS technique





Conclusion

- Combining ML-based RTS and Analysis-based RTS improves the precision of analysis-based RTS
    - Our best ML-based RTS model reduces the average selection rate of two analysis-based RTS techniques, Ekstazi and STARTS by 25.34% and 21.44%
- Combining ML-based RTS models with an analysis-based RTS technique results in reduced end-to-end testing time
    - Overhead of the ML-based RTS models are small compared with the analysis-based RTS techniques
    - Substantial time savings result from running fewer tests than what the analysis-based RTS technique selects

Thank you!

Jiyang Zhang <jiyang.zhang@utexas.edu>