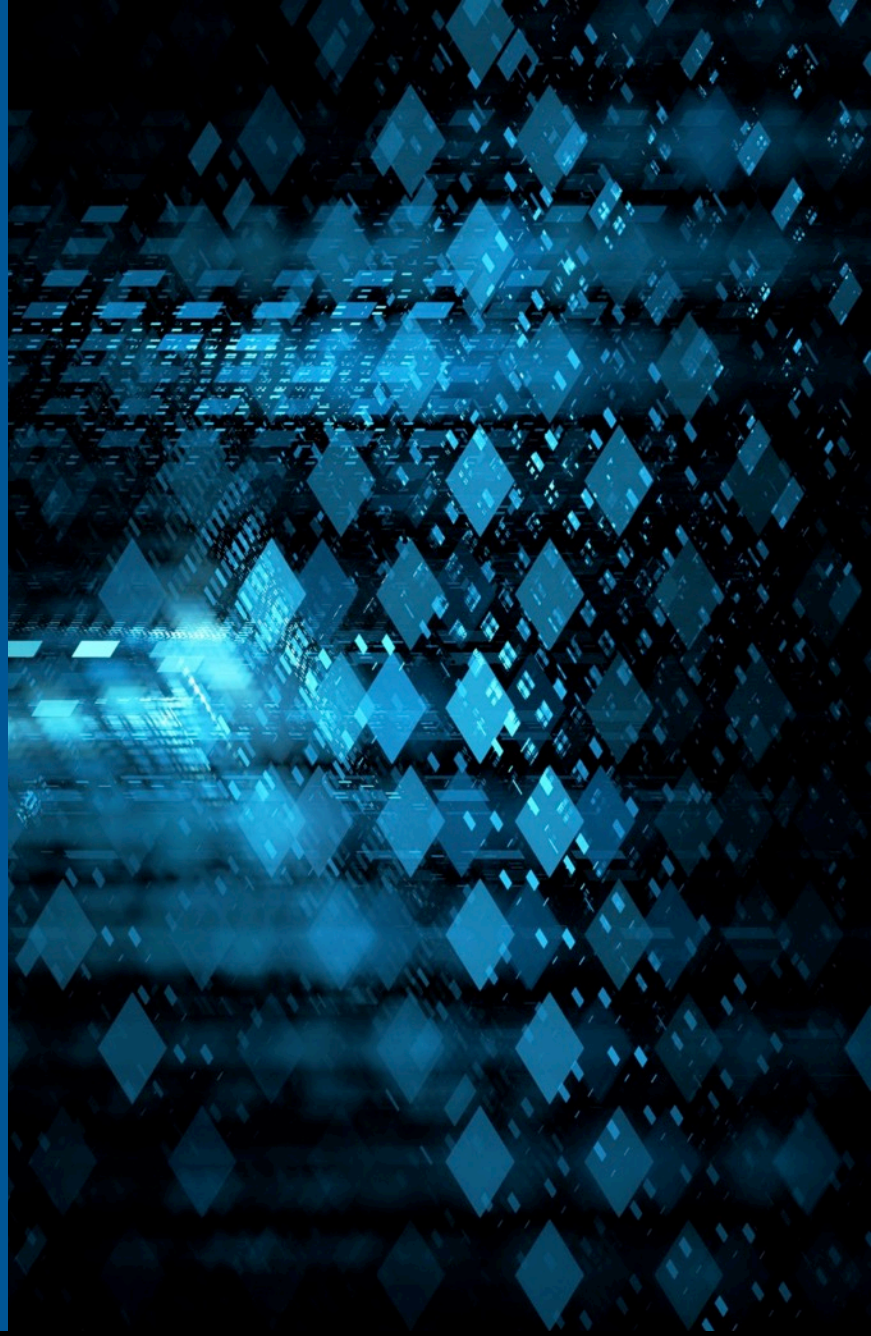
Prioritizing Alerts from Multiple Static Analysis Tools, using Classification Models

Lori Flynn

Team: Will Snavely, David Svoboda, Nathan VanHoudnos, Richard Qin, Jennifer Burns, David Zubrow, Robert Stoddard, Guillermo Marce-Santurio.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

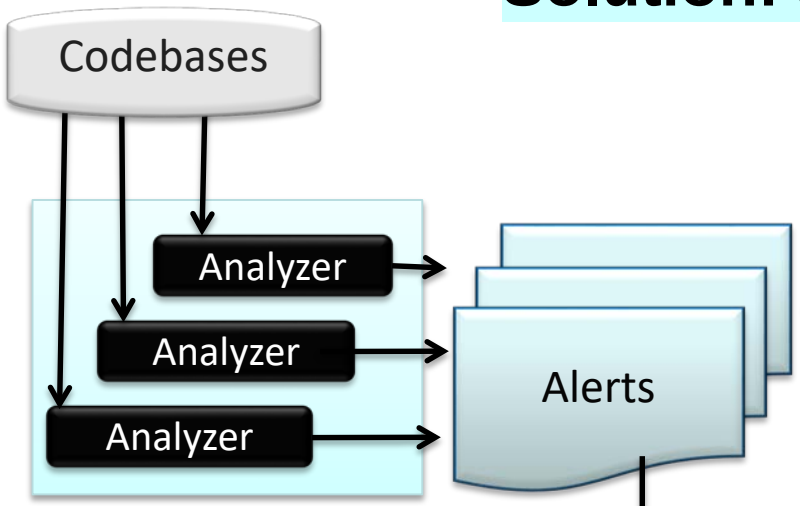


This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-0707

Overview

Problem: too many alerts
Solution: automate handling



Today

Goal

Classification algorithm development using "pre-audited" and manually-audited data, that

accurately classifies most of the alerts as:

Expected True Positive (e-TP) or
Expected False Positive (e-FP),
and
the rest as Indeterminate (I)

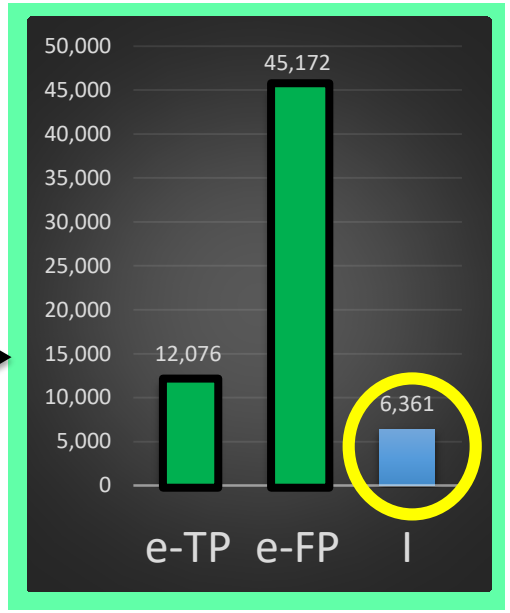
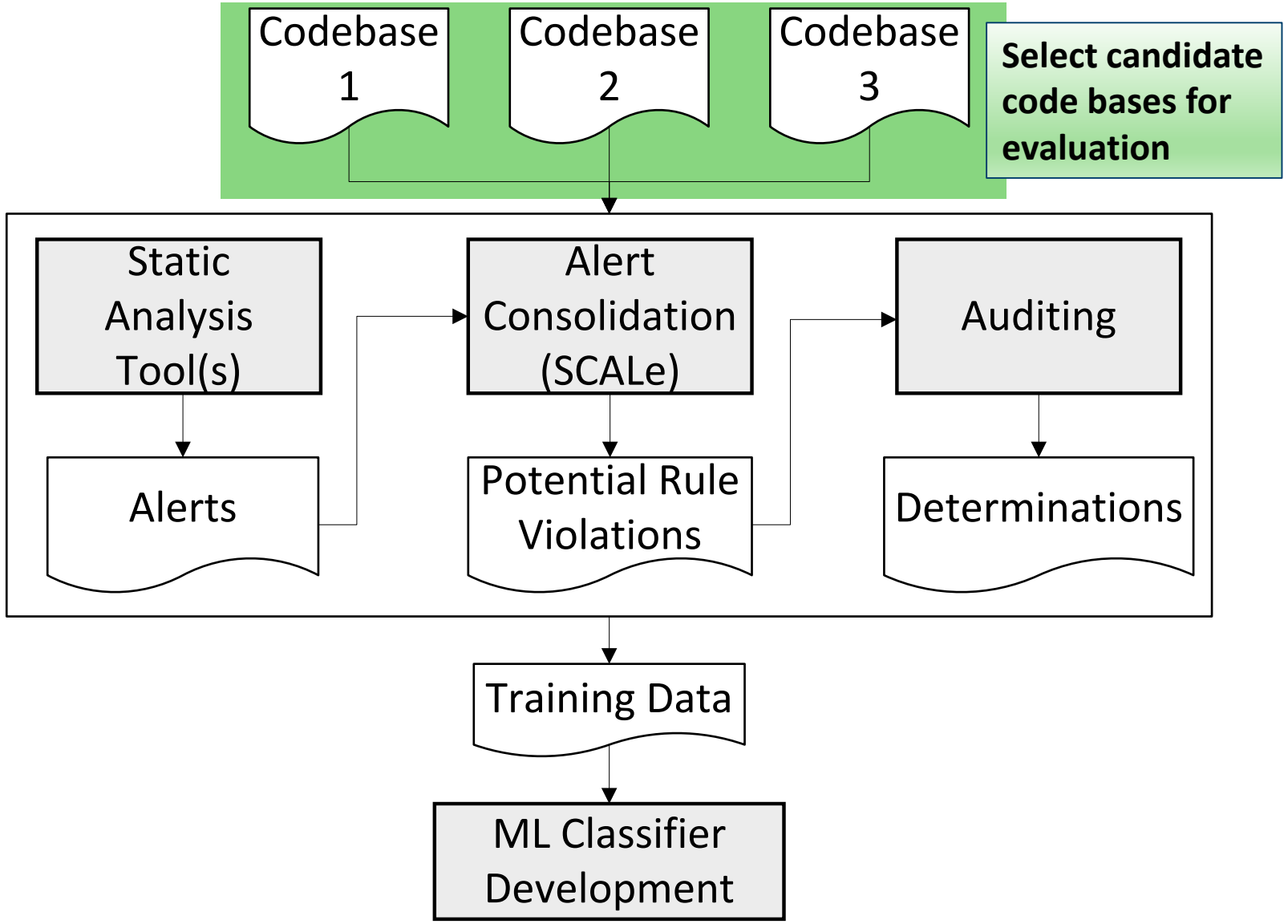
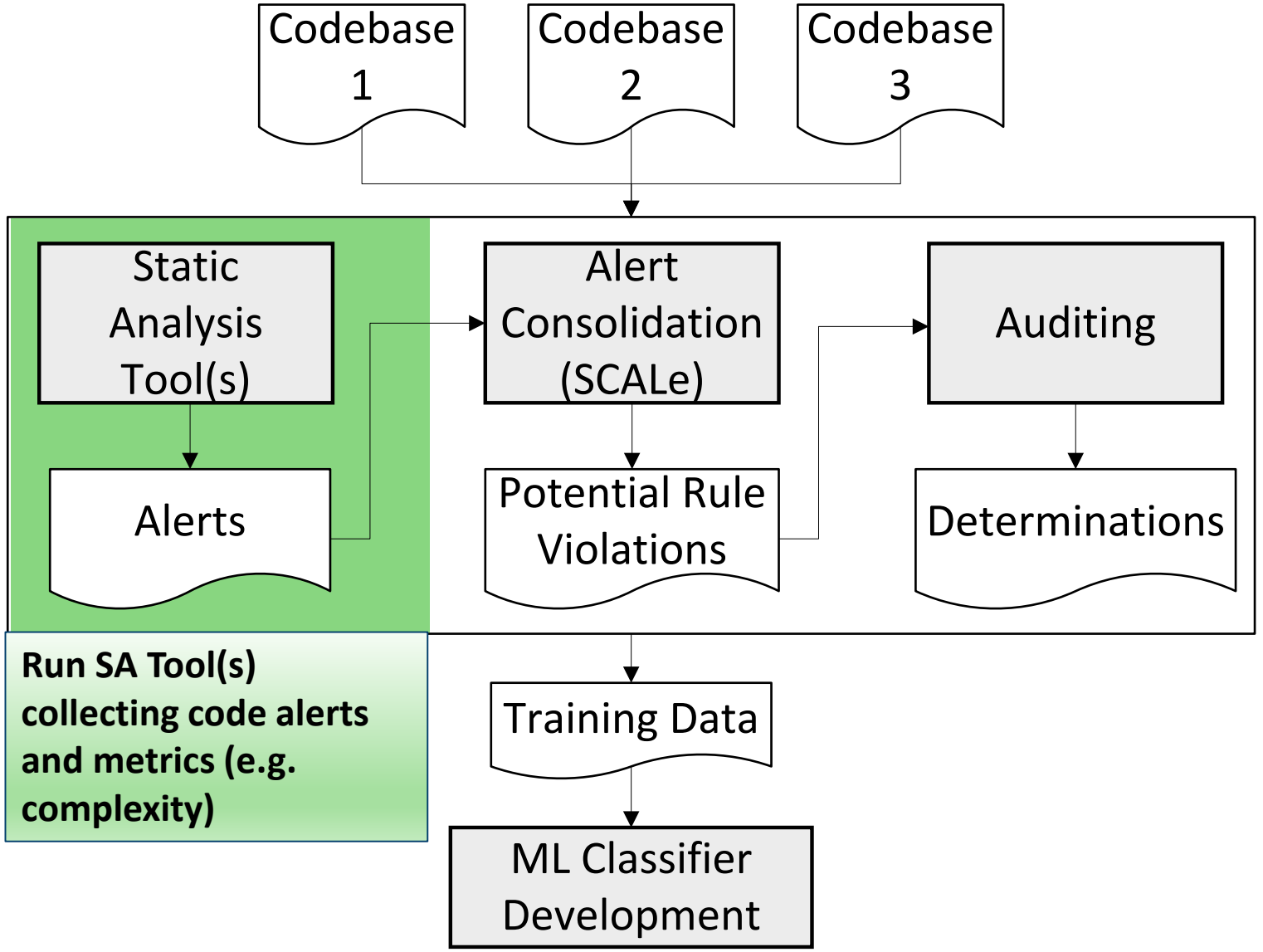


Image of woman and laptop from http://www.publicdomainpictures.net/view-image.php?image=47526&picture=woman-and-laptop "Woman And Laptop"

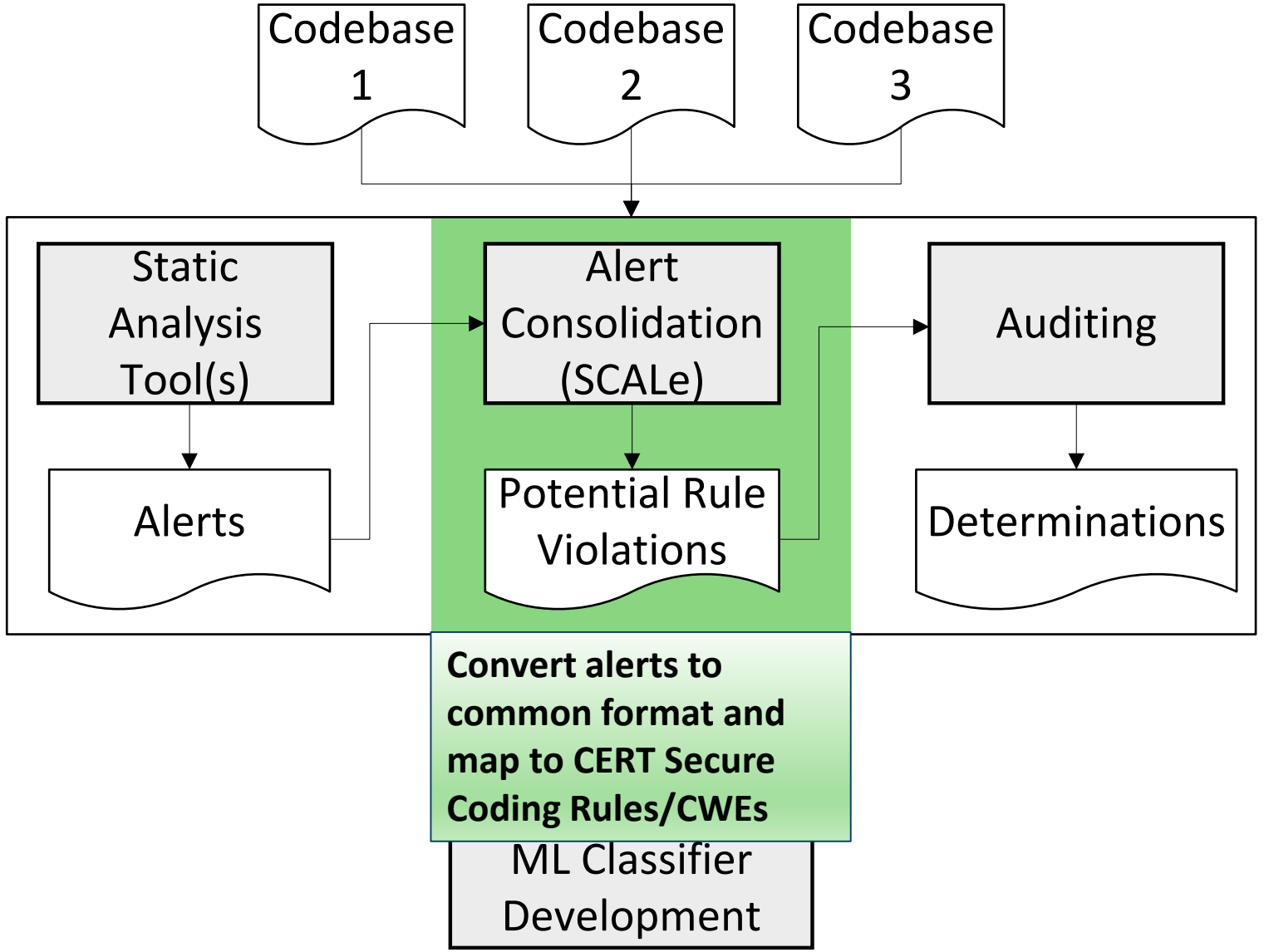
Background: Automatic Alert Classification



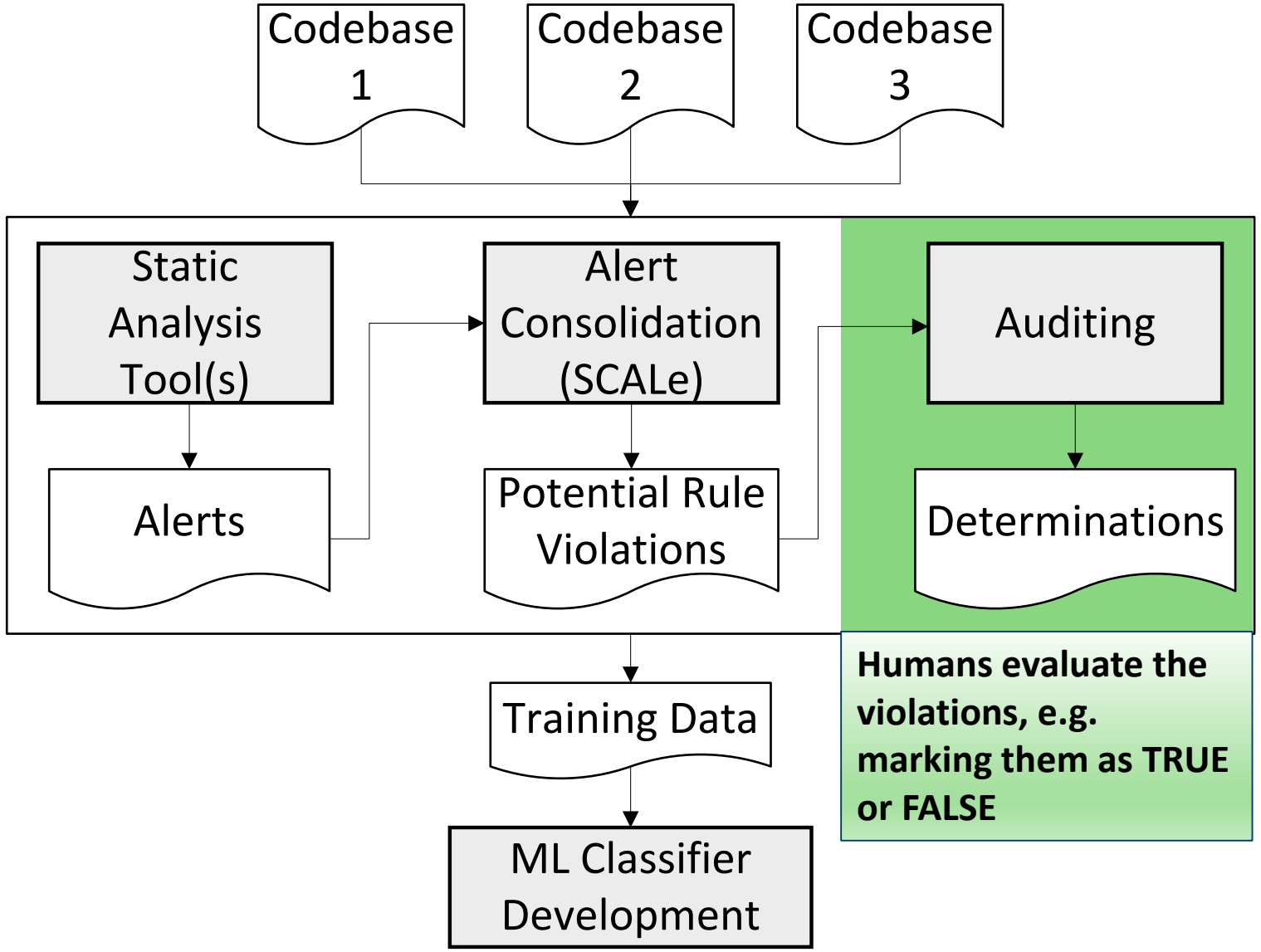
Background: Automatic Alert Classification



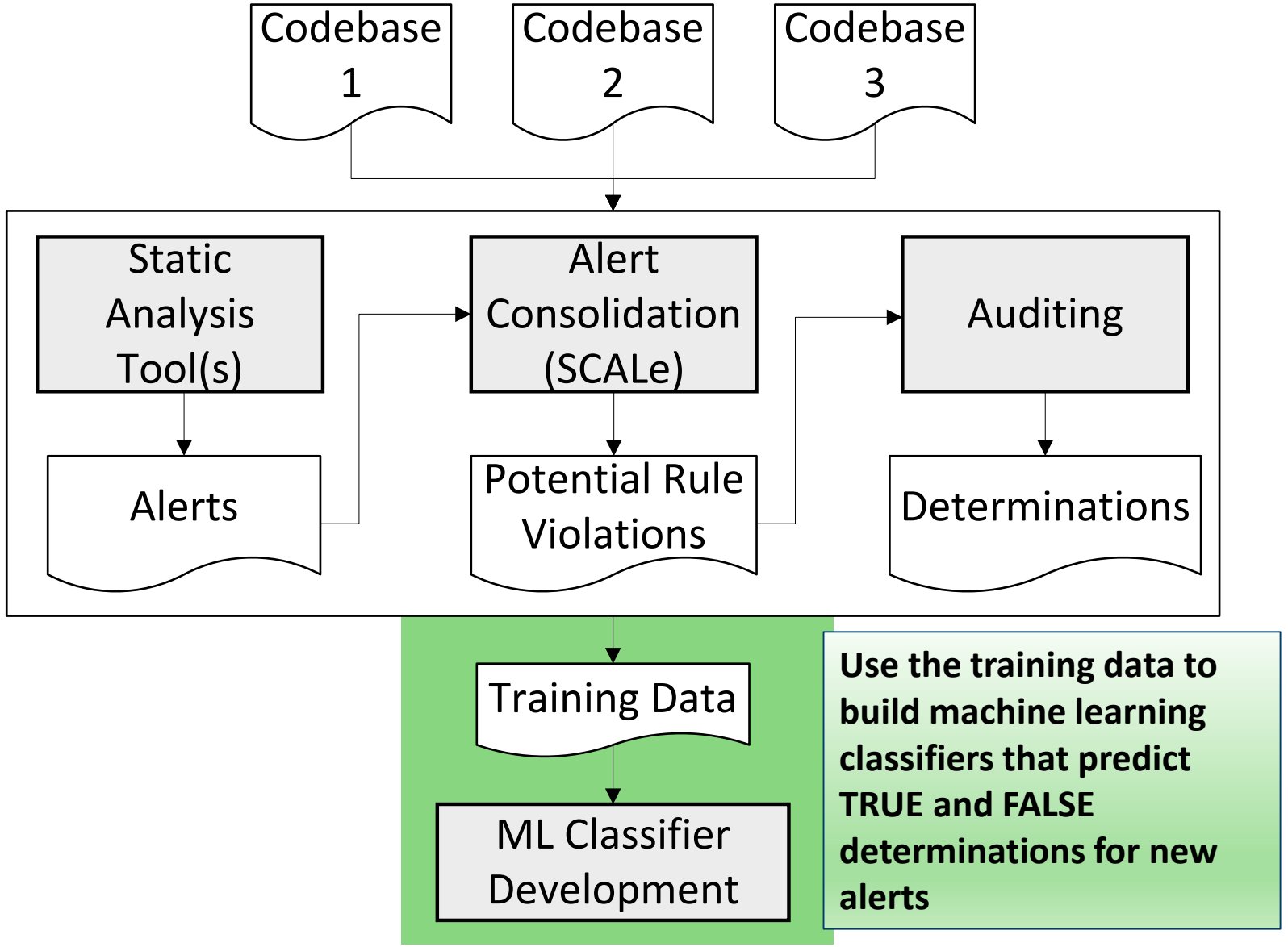
Background: Automatic Alert Classification



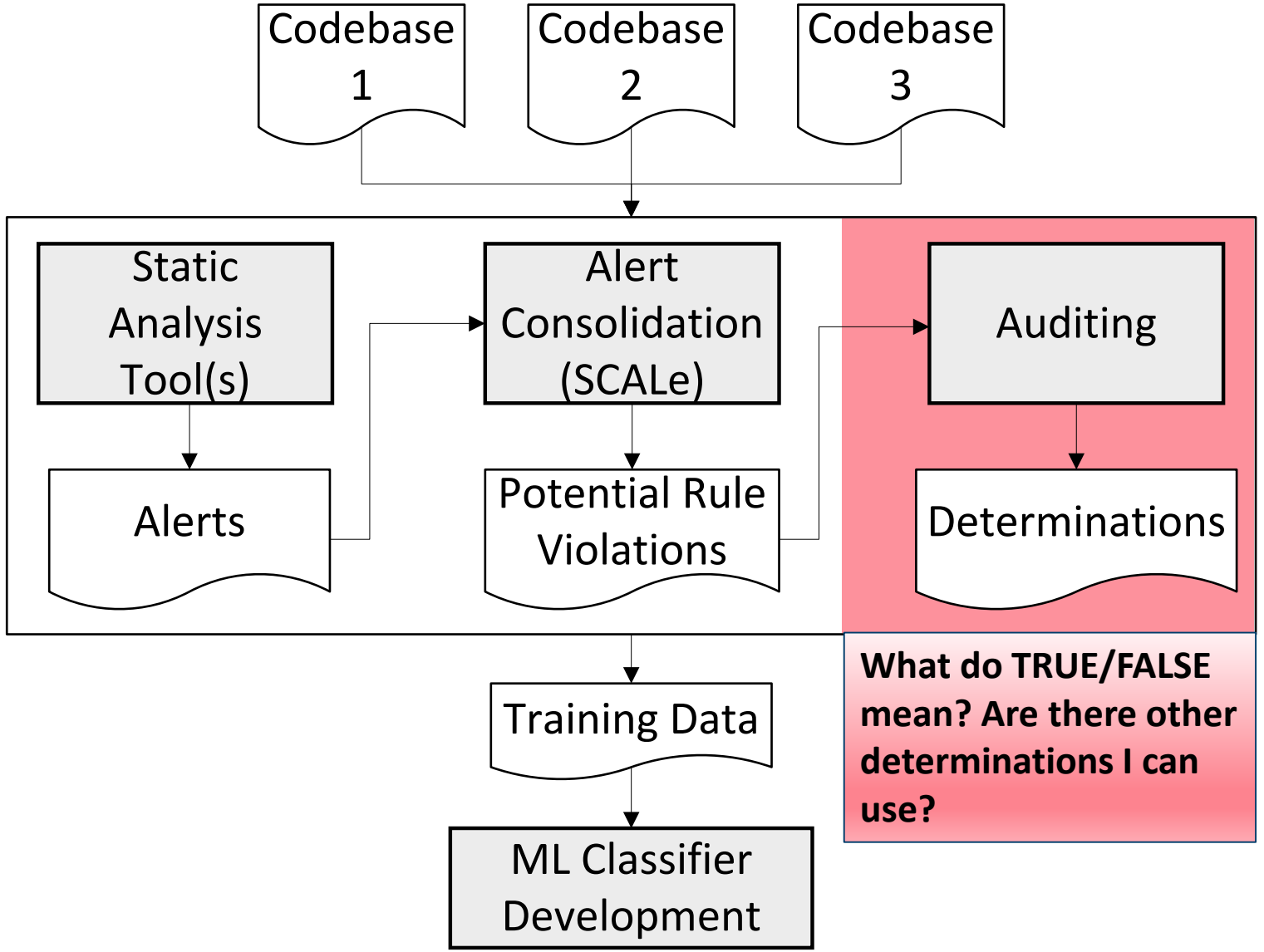
Background: Automatic Alert Classification



Background: Automatic Alert Classification



Background: Automatic Alert Classification

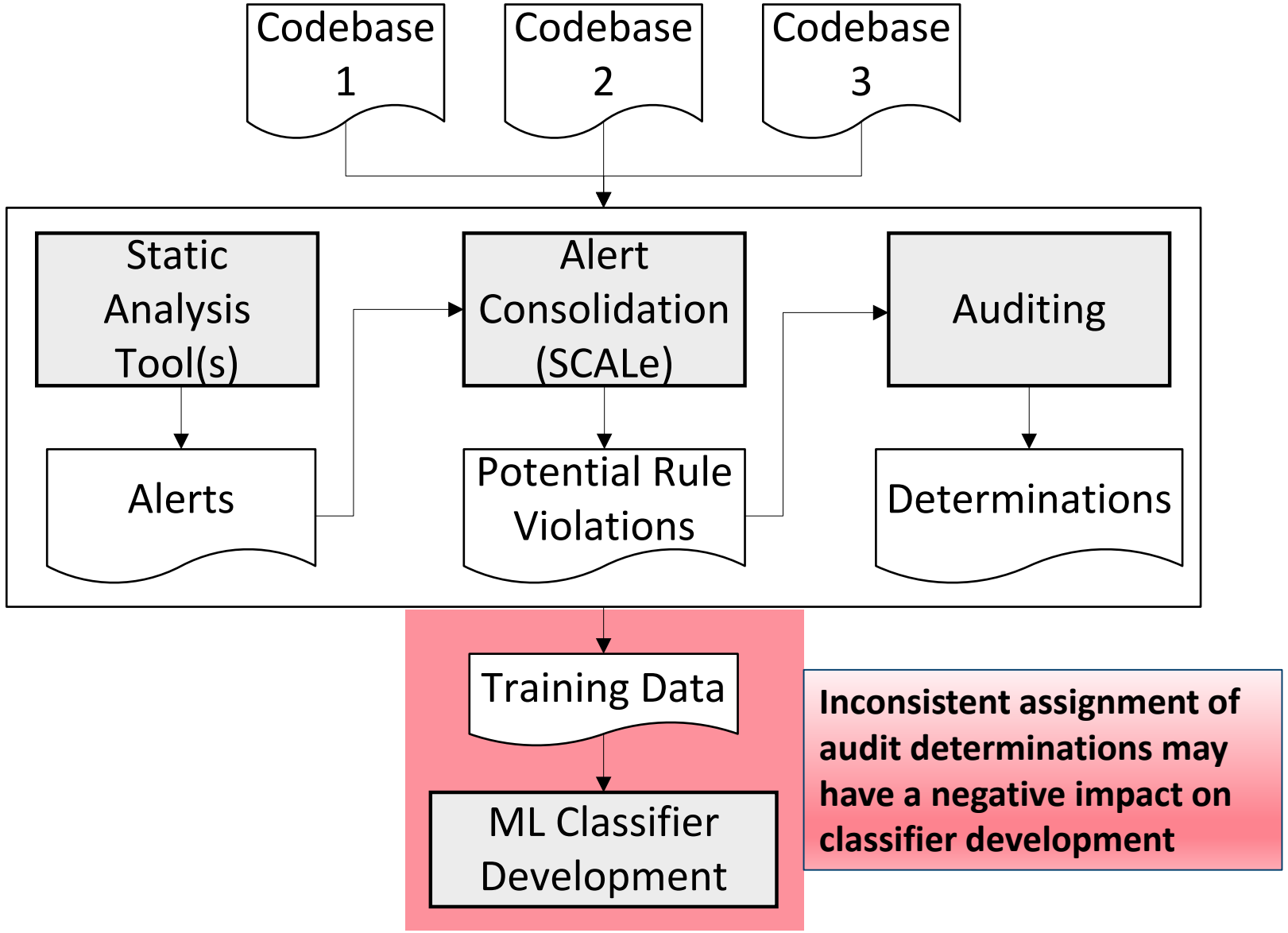


What is truth?

One collaborator reported using the determination **True** to indicate that the issue reported by the alert was a real problem in the code.

Another collaborator used **True** to indicate that *something* was wrong with the diagnosed code, even if the specific issue reported by the alert was a **false positive**!

Background: Automatic Alert Classification



Inconsistent assignment of audit determinations may have a negative impact on classifier development

Solution: Lexicon And Rules

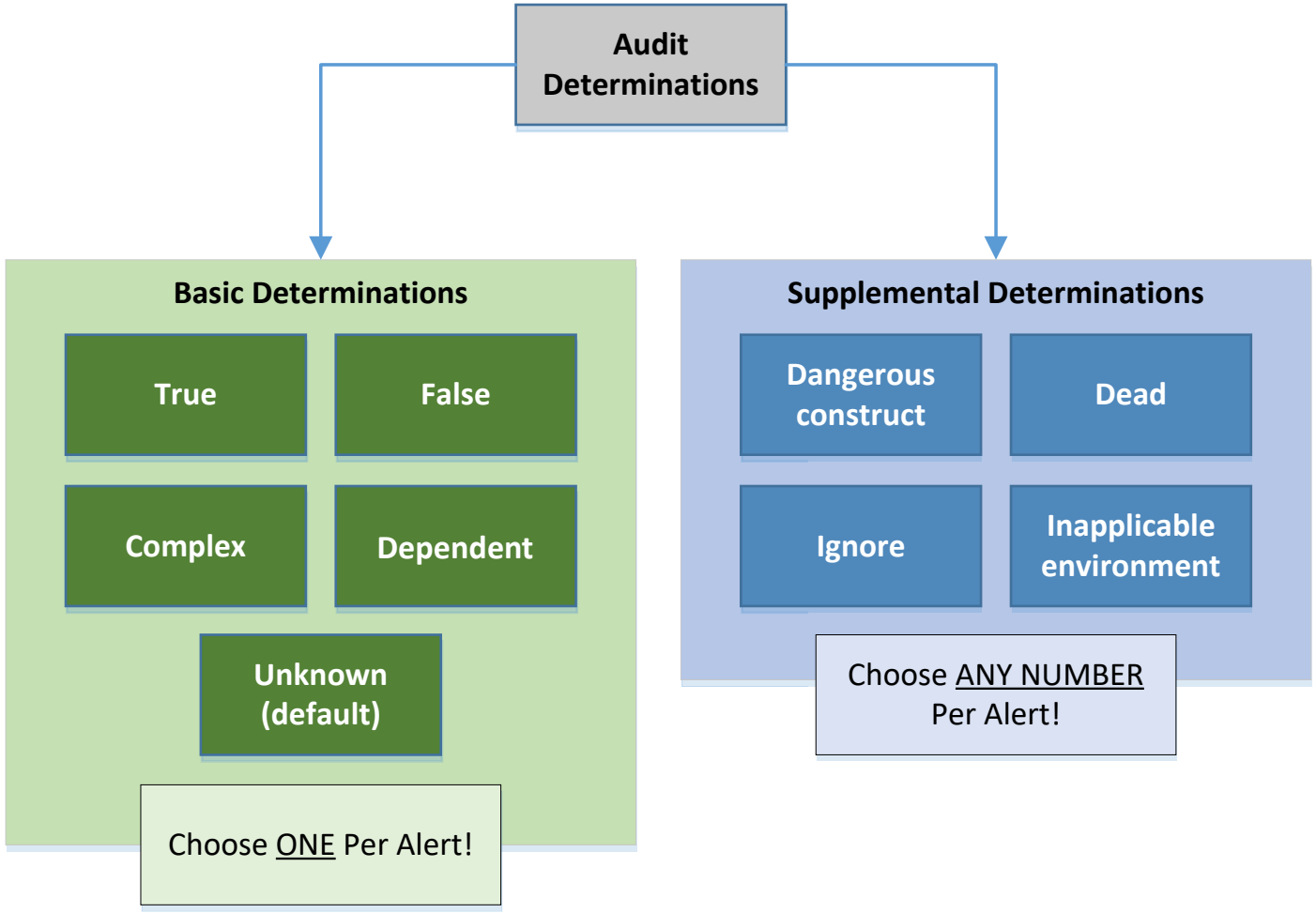
- We developed a **lexicon** and auditing **rule set** for our collaborators
- Includes a standard set of well-defined **determinations** for static analysis alerts
- Includes a set of **auditing rules** to help auditors make consistent decisions in commonly-encountered situations

Different auditors should make the **same determination** for a given alert

Improve the **quality and consistency** of audit data for the purpose of building **machine learning classifiers**

Help organizations make **better-informed** decisions about **bug-fixes**, **development**, and **future audits**.

Lexicon: Audit Determinations



Lexicon: Basic Determinations

True

- The code in question violates the **condition** indicated by the alert.
- A **condition** is a constraint or property of validity.
  - E.g. A valid program should not deference NULL pointers.
- The condition can be determined from the definition of the alert itself, or from the **coding taxonomy** the alert corresponds to.
  - CERT Secure Coding Rules
  - CWEs

Audit Rules

Goals

- Clarify **ambiguous or complex** auditing scenarios
- Establish **assumptions** auditors can make
- Overall: help make audit determinations **more consistent**

We developed **12 rules**

- Drew on our own experiences auditing code bases at CERT
- Trained 3 groups of engineers on the rules, and incorporated their feedback

Auditing Rules

Rule 1 Understand the language and the secure coding rule in question.

Rule 2 Some alerts are too complex to judge; they should be marked Suspicious

Rule 3 It is OK to mark an alert true even if you think the code maintainers will protest.

Rule 4 Assume that external inputs to the program are malicious.

Rule 5 Unless instructed otherwise, assume that code must be portable.

Rule 6 When auditing an alert, if you discover a second true violation, mark its alert as true.

Rule 7 Do not arbitrarily extend the scope of a CERT rule.

Rule 8 Code that behaves as expected might still violate a CERT rule.

Rule 9 An alert might indicate a true violation of the CERT rule, even if its message text is useless or incorrect.

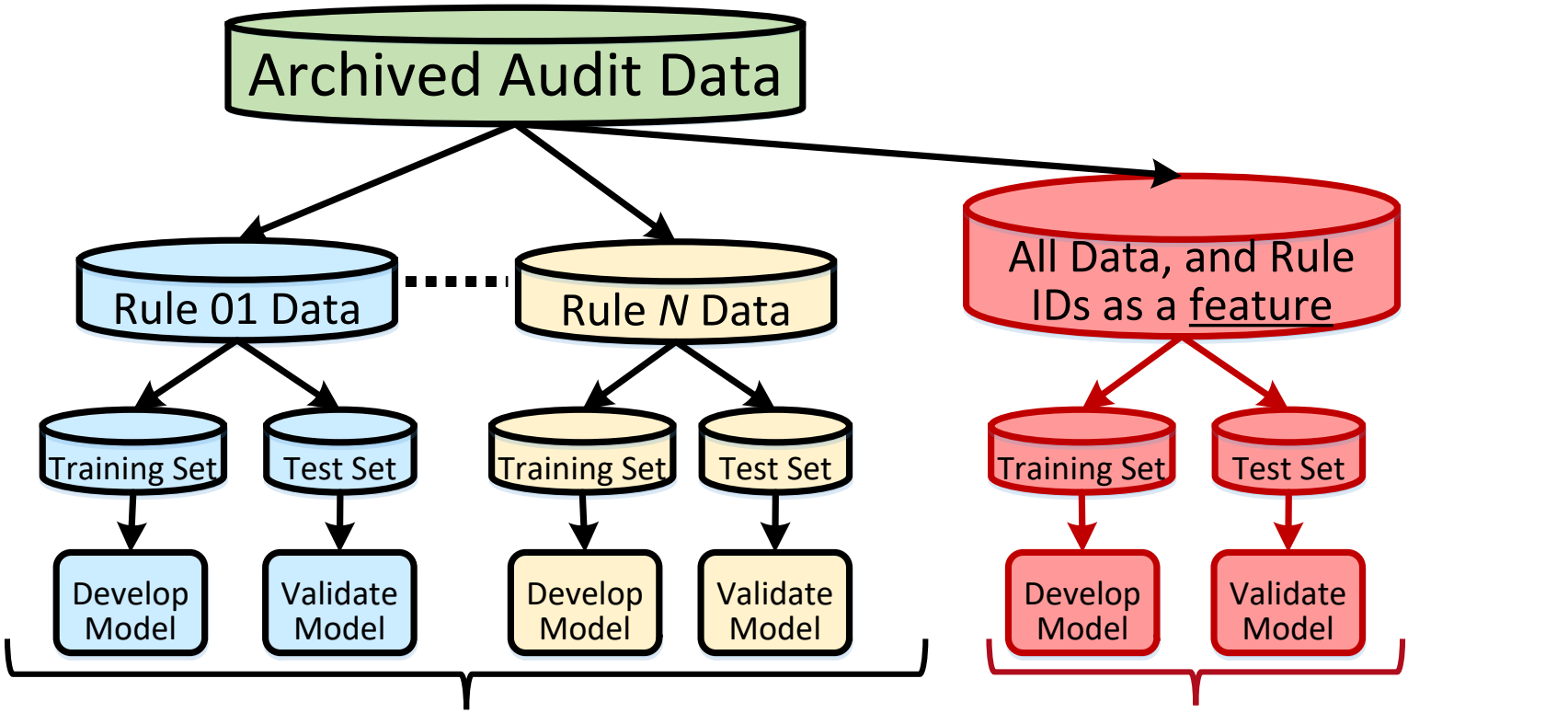
Rule 10 Multiple messages help in understanding an alert.

Rule 11 Assume no violations occur before the line in question.

Rule 12 Handle an alert in unreachable code depending on if it is in library or non-library code.

Scientific Approach

Novel combined use of:
1) multiple analyzers, 2) variety of features,
3) competing classification techniques



Competing Classifiers to Test
Lasso Logistic Regression
CART (Classification and Regression Trees)
Random Forest
Extreme Gradient Boosting (XGBoost)

Some of the features used (many more)
Analysis tools used
Significant LOC
Complexity
Coupling
Cohesion
SEI coding rule

Data Used for Classifiers

Data used to create and validate classifiers:

- CERT-audited alerts:
  - ~7,500 audited alerts
- 3 collaborators audit their own codebases with our auditing research prototype tool “enhanced SCALe”

We pooled data (CERT + collaborators) and segmented it:

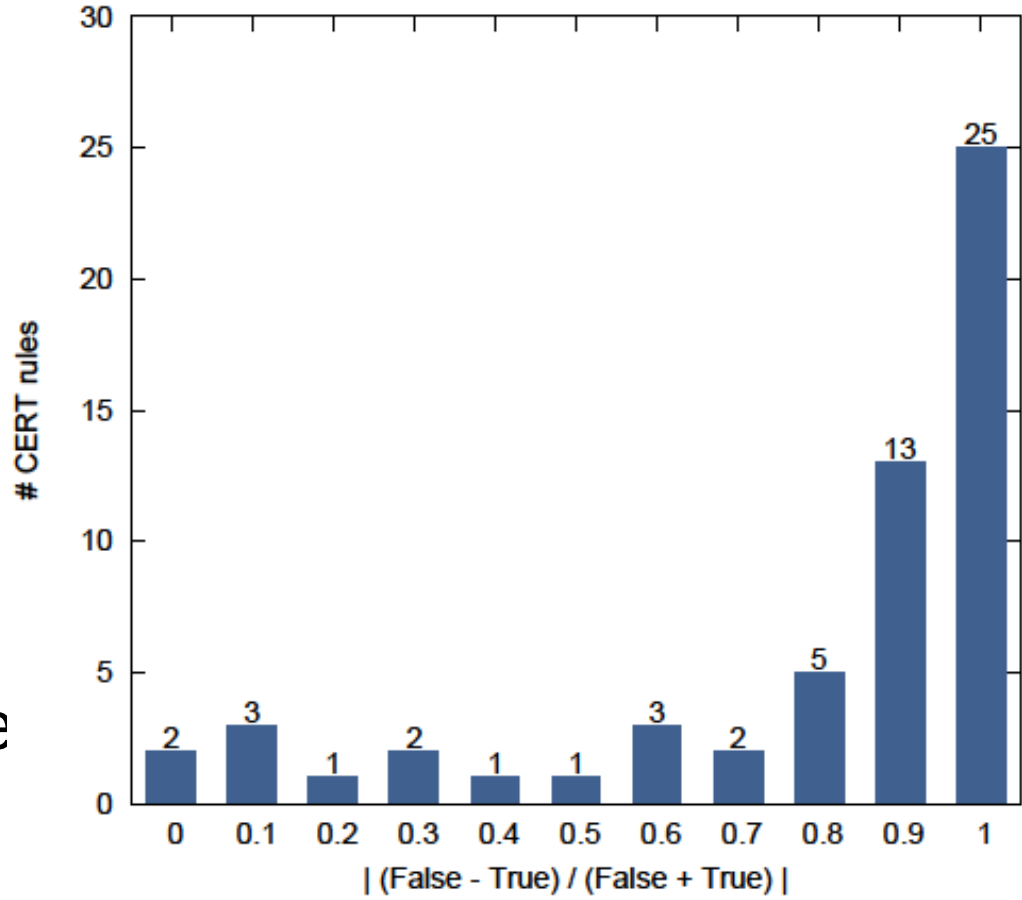
- Segment 1 (70% of data): train model
- Segment 2 (30% of data): testing

Added classifier variations on dataset:

- Per-rule
- Per-language
- With/without tools
- Others

CERT- Audited Archives Characterization

- 58 CERT coding rules with 20 or more audited (labeled) alerts
- 25 rules all (or nearly all) determined one way (True or False)
- Other 324 CERT rules have little or no labeled data
- Labeled data for 158 of 382 CERT rules
- 30% of the CERT-audited archives is for rule INT31-C
- 2,487 True and 4,980 False



Archive sanitizer: enabled collaborator data use

Added data sanitizer to “enhanced SCALe”

- Anonymizes sensitive fields
- SHA-256 hash with salt
- Enables analysis of features correlated with alert confidence

Audit archive for project is in a database

- DB fields may contain sensitive information
- Sanitizing script anonymizes or discards fields
  - Diagnostic message
  - Path, including directories and filename
  - Function name
  - Class name
  - Namespace/package
  - Project filename

Collaborator Data

354 audited alerts

- 144 False and 210 True

8 targeted rules

- CERT archives had at least one true and one false

Plus more audited rules

Classifier Result Highlights: Data All Sources

Classifiers made from all data, pooled:

All-rules (158) classifier accuracy:

- Lasso Logistic Regression: 88%
- Random Forest: 91%
- CART: 89%
- XGBoost: 91%

Single-rule classifier accuracy:

Rule ID	Lasso LR	Random Forest	CART	XGBoost
INT31-C	98%	97%	98%	97%
EXP01-J	74%	74%	81%	74%
OBJ03-J	73%	86%	86%	83%
FIO04-J*	80%	80%	90%	80%
EXP33-C*	83%	87%	83%	83%
EXP34-C*	67%	72%	79%	72%
DCL36-C*	100%	100%	100%	100%
ERR08-J*	99%	100%	100%	100%
IDS00-J*	96%	96%	96%	96%
ERR01-J*	100%	100%	100%	100%
ERR09-J*	100%	88%	88%	88%

Also, 15 one-way “classifiers”.

General results (not true for every test)

- Classifier accuracy rankings for all-pooled test data:
  XGBoost ≈ RF > CART ≈ LR
- Classifier accuracy rankings for collaborator test data:
  LR ≈ RF > XGBoost > CART
- Per-rule classifiers generally not useful (lack data), but 3 rules (INT31-C best) are exceptions.
- With-tools-as-feature classifiers better than without.
- Accuracy of single language vs. all-languages data:
  C > all-combined > Java

* Small quantity of data, results suspect

Classifier Result Highlights: CERT-only data

Classifiers made from all data, pooled:

All-rules (158) classifier accuracy:

- Lasso Logistic Regression: 87%
- Random Forest: 90%
- CART: 89%
- XGBoost: 92%

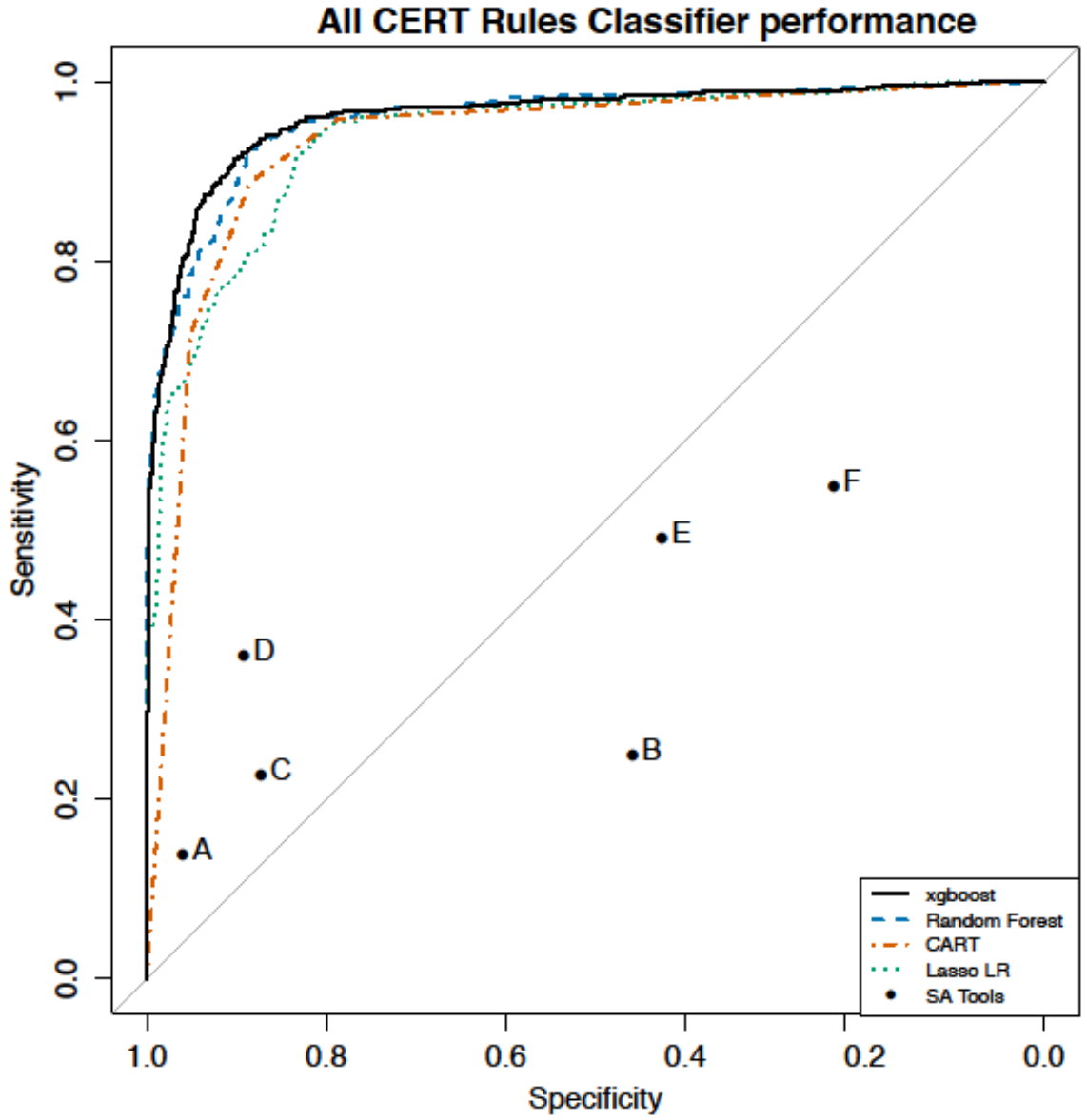
Similar results for CERT-only data

Single-rule classifier accuracy:

Rule ID	Lasso LR	Random Forest	CART	XGBoost
INT31-C	95%	96%	96%	95%
EXP01-J	68%	83%	89%	87%
OBJ03-J	73%	86%	86%	83%
FIO04-J*	75%	75%	83%	71%
EXP33-C*	90%	100%	90%	90%
EXP34-C*	74%	84%	87%	81%
DCL36-C*	100%	100%	100%	100%
ERR08-J*	99%	99%	97%	97%
IDS00-J*	97%	94%	94%	88%
ERR01-J*	100%	100%	100%	100%

* Small quantity of data, results suspect

Tool as Feature Helped



Using toolname as a feature improved classifier performance

Dots show performance of tool alone

Classifier Results: Per-Language, Fully Pooled Data

All Java data, pooled:

- All-Java-rules classifier accuracy:
  - Lasso Logistic Regression: 83%
  - Random Forest: 98%
  - CART: 86%
  - XGBoost: 90%

All C data, pooled:

- All-C-rules classifier accuracy:
  - Lasso Logistic Regression: 93%
  - Random Forest: 95%
  - CART: 94%
  - XGBoost: 93%

Built classifiers using 70% of data for single language

- All-rules (rules as feature)

Tested classifiers on remaining 30% data

Too little Perl data to create classifiers

All C++ data, pooled:

- All-C++-rules classifier accuracy:
  - Lasso Logistic Regression: 92%*
  - Random Forest: 92%*
  - CART: 100%*
  - XGBoost: 100%*

* C++ classifiers suspect (little data, ROC graph)

Challenges

Lack of data

- Missing labeled data (true/false) for many conditions

Sensitive data

- Helped: sanitized audit archives
- Can share aggregated classifier performance results
  - Remaining issue: Even if sanitized, not raw data

Proprietary tool performance and output

- Anonymized results
- Remaining issue: Cannot share this classifier development data

Cross-project prediction, adaptive heuristics: not addressed

Future Work

Goal: improve accuracy for more conditions

- Different mix of features:
  - Semantic features
  - Code repository logfile features
- More audit archive data needed
  - Additional data welcome! Potential collaborators, please contact me
  - **Current follow-on research focuses on using a labeled dataset to jump-start classifiers**
- Publish open-source data (only open-source programs + static analysis tools)

Contact Information

Lori Flynn
Software Security Researcher
Telephone: +1 412.268.7886
Email: lflynn@sei.cmu.edu

Discussion