TU Clausthal
Clausthal University of Technology

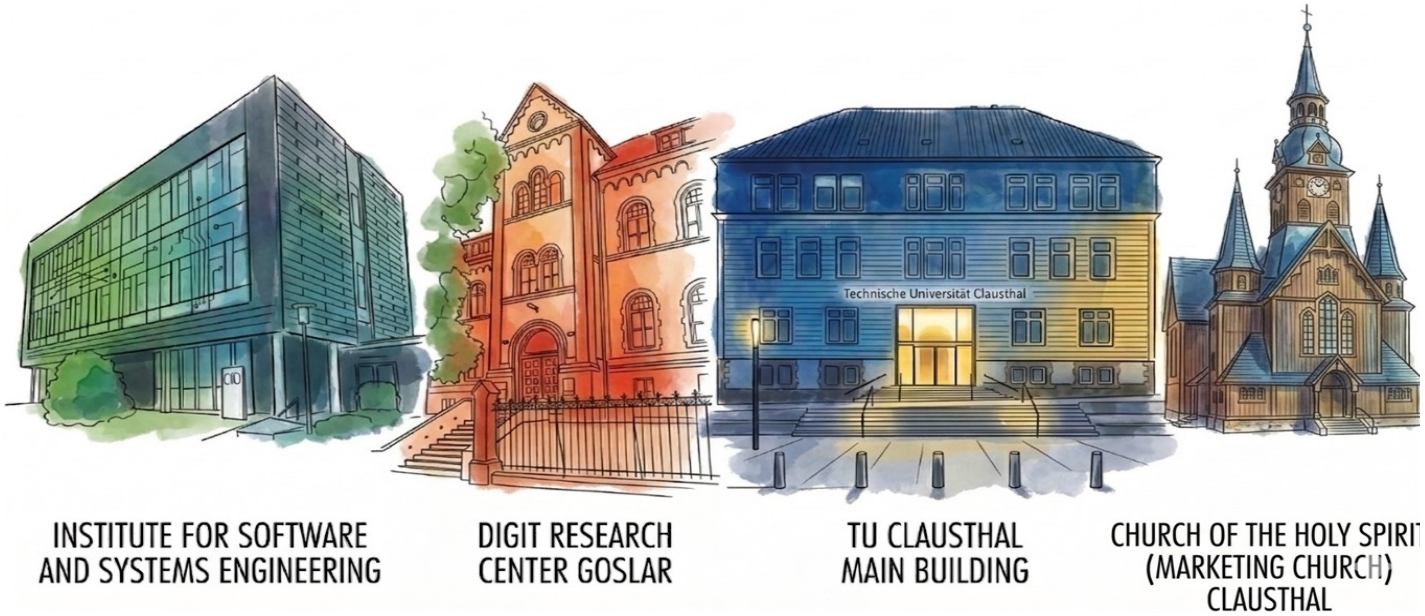
ISSE Institute for Software and Systems Engineering

AI-driven Identification of Security-Relevant Artifacts

Farnaz Soltaniani

PhD Viva Session @ Technische Universität Clausthal, Germany

7th of July, 2026



Chair of the Examiners: Prof. Dr. Benjamin Säfken

Thesis Supervisor: Prof. Dr. Mohammad Ghafari

2nd Reviewer: Prof. Dr. Christoph Knieke

3rd Reviewer: Prof. Dr. Mohammed Sayagh

Bridging the Gap: A Roadmap for AI-driven Security Analysis



1. What are the **main challenges** and **"semantic gaps"** in current security vulnerability detection?



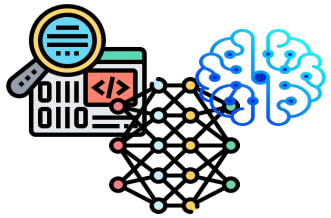
2. How do **non-generative** and **generative AI** compare for secret detection problem?

Data Leakge

3. How much do **code duplication** and **data leakage** affect different models validity?



4. Can we reliably predict security-vulnerabilities across **different software projects**?



5. **Which is more promising:** fine-tuned or prompted proprietary models?

Security **Vulnerability Detection Approaches**

Static Analysis Tools

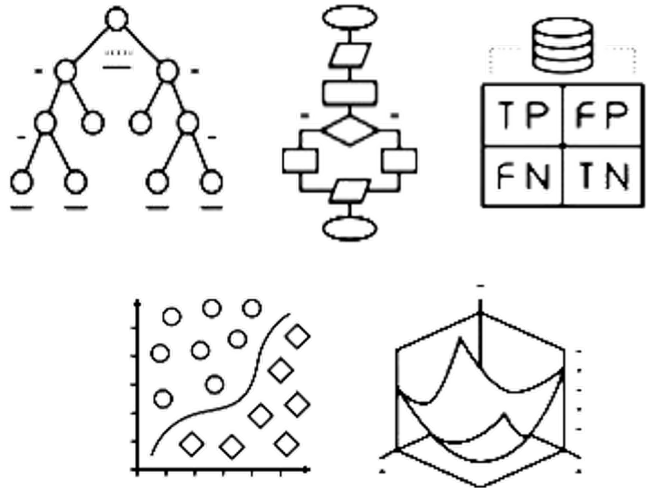
- Heuristic rules, pattern matching
- High numbers of FPs
- Developer Frustration
- Semantic Gaps
- Limited Language and PM Support
- Each Tool has its Own Language

```
1  app.post('/products', function (req, res) {
2    // Use a placeholder (?) for the user input
3    const sql = 'SELECT * FROM Products WHERE name LIKE ?';
4    const searchTerm = `%${req.body.product_name}%`;
5
6    db.query(sql, [searchTerm], (err, results) => {
7      if (err) throw err;
8      res.json(results);
9    });
10 });
```

Potential SQL Injection via string-based query concatination

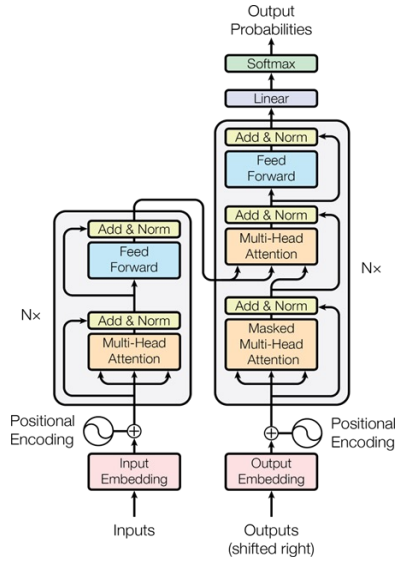
Traditional Machine Learning (ML) Deep Learning (DL)

- Manual Feature Extraction
- Large, task-specific labeled datasets
- Real-world Applicability
- Memorizing Terminology
- Data Leakage



Pre-trained Large Language Models

- Increased Complexity
- Inference Costs
- Poor Generalization
- Context Truncation
- Prompt Sensitivity



Security **Vulnerability Detection Approaches**

Static Analysis Tools

Heuristic rules, pattern matching

High numbers of FPs

Developer Frustratio

Semantic Gaps

Limited Language and PM S

Each Tool has its Own La

Traditional Machine Learning (ML)
Deep Learning (DL)

Manual Feature Extraction

Pre-trained
Large Language Models

Increased Complexity

Use of Hard-coded Credentials (CWE-798)



Secret Detection

Bug prioritization



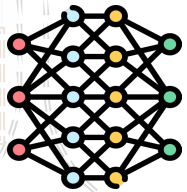
Security Bug Report prediction



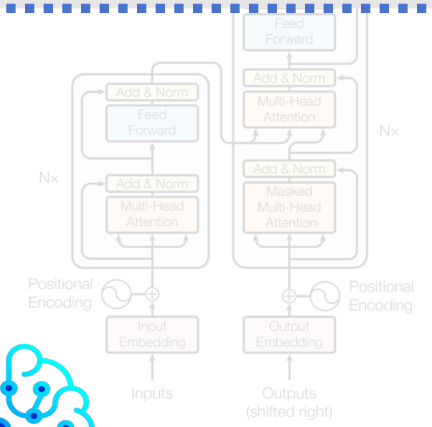
```
('/products', function (req, res
placeholder (?) for the user input
= 'SELECT * FROM Products WHERE name LIKE ?';
chTerm = `%${req.body.product_name}%`;

db.query(sql, [searchTerm], (err, results) => {
  if (err) throw err;
  res.json(results);
});
});
```

Potential SQL Injection via string-based query concatination







Research Questions

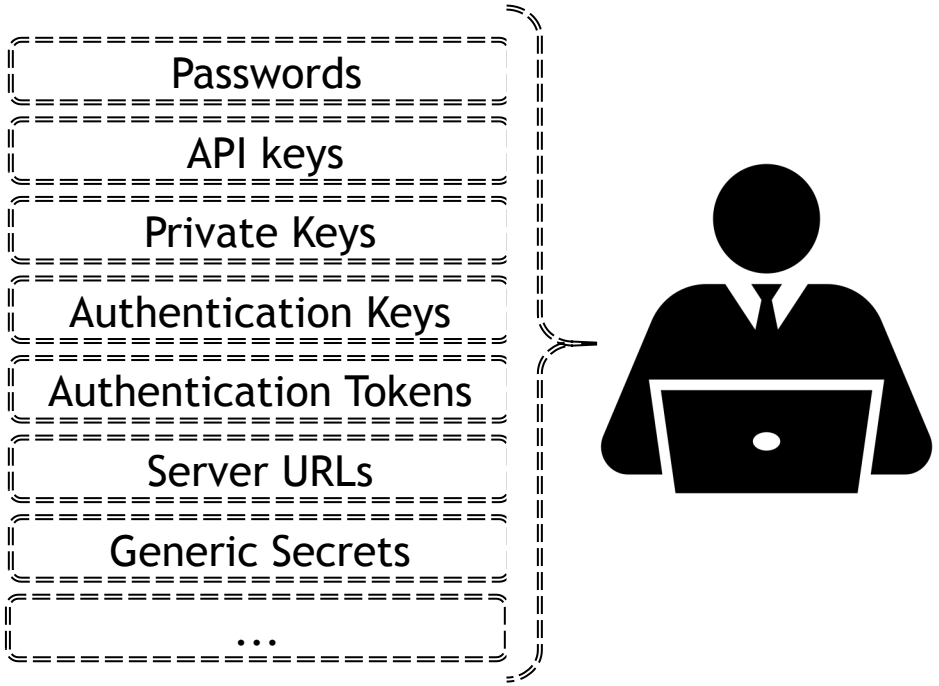
RQ_1: How do traditional ML and LLMs (non-generative and generative AI models) differ in what they learn?

RQ_2: To what extent do data quality and AI usage issues – including duplication, class imbalance, and cross-split leakage – compromise the validity and generalization of these models?

1 Secret Detection in Source Code

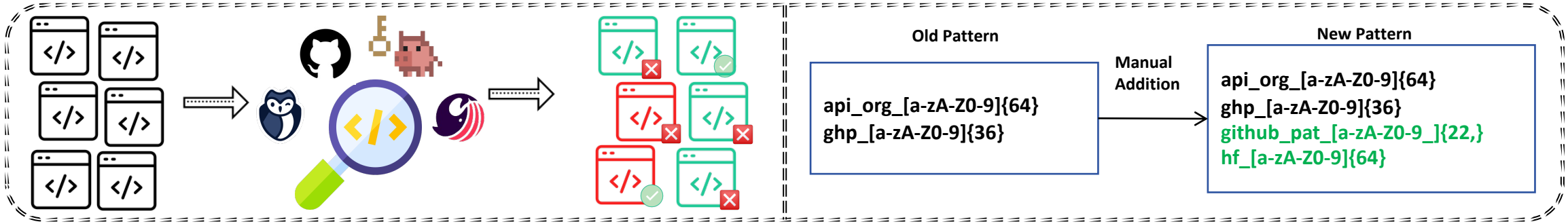
Secrets need to be safely stored and loaded from external sources (e.g., environment variables, configuration files, or secret management tools).

```
1  import boto3
2  import json
3  import sys
4
5  from typing import Dict
6
7  ACCESS_KEY = 'AKIAXJHR7VFNE7DRJF7'
8  SECRET_ACCESS_KEY = 'TnxfnQECFn8vCy9U7wuzdzlfmwPpWyTSuGXju887'
9
10 class StreamFromS3:
11     def __init__(self):
12         self.s3_url: str = 'https://testbucket10022022.s3.eu-west-
   1.amazonaws.com/very_short_stream_data.json'
13         self.s3 = boto3.client('s3', aws_access_key_id=ACCESS_KEY,
   aws_secret_access_key=SECRET_ACCESS_KEY)
14         self.iot_client = boto3.client('iot-data',
   aws_access_key_id=ACCESS_KEY, aws_secret_access_key=SECRET_ACCESS_KEY)
```



➢ Hard-coding secrets in source code is a bad practice that compromises the security of confidential resources.

1 Secret Detection: What are the Challenges?



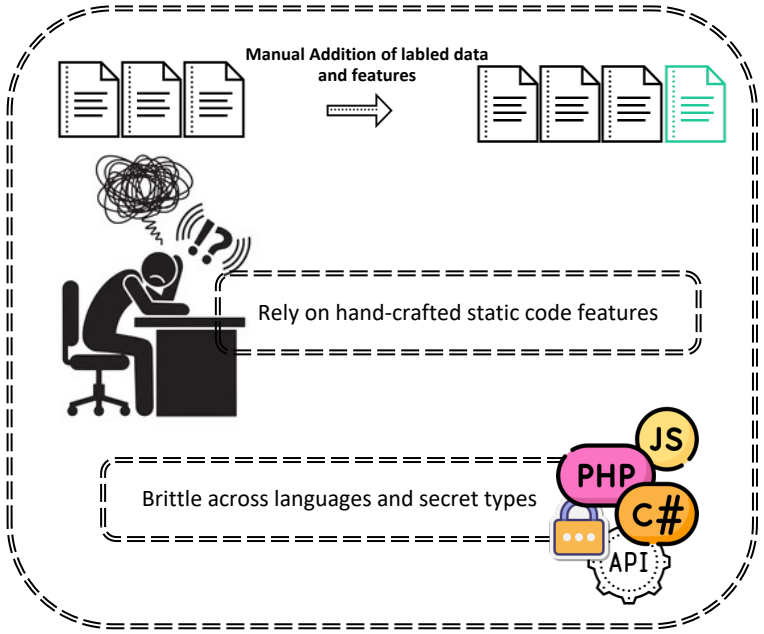


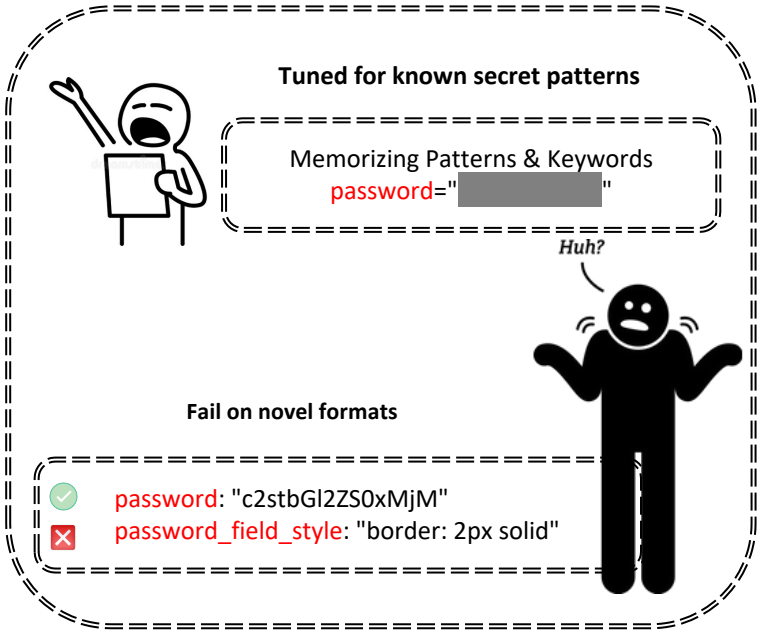
"Alert Fatigue"

- Detect many Benign strings
- Constrained by their rules
- Update detection rules periodically
- Skip specific file types

➢ Rule-Based Generic Regular Expression Tools have Insufficient Rulesets.

1 Secret Detection: What are the Challenges?

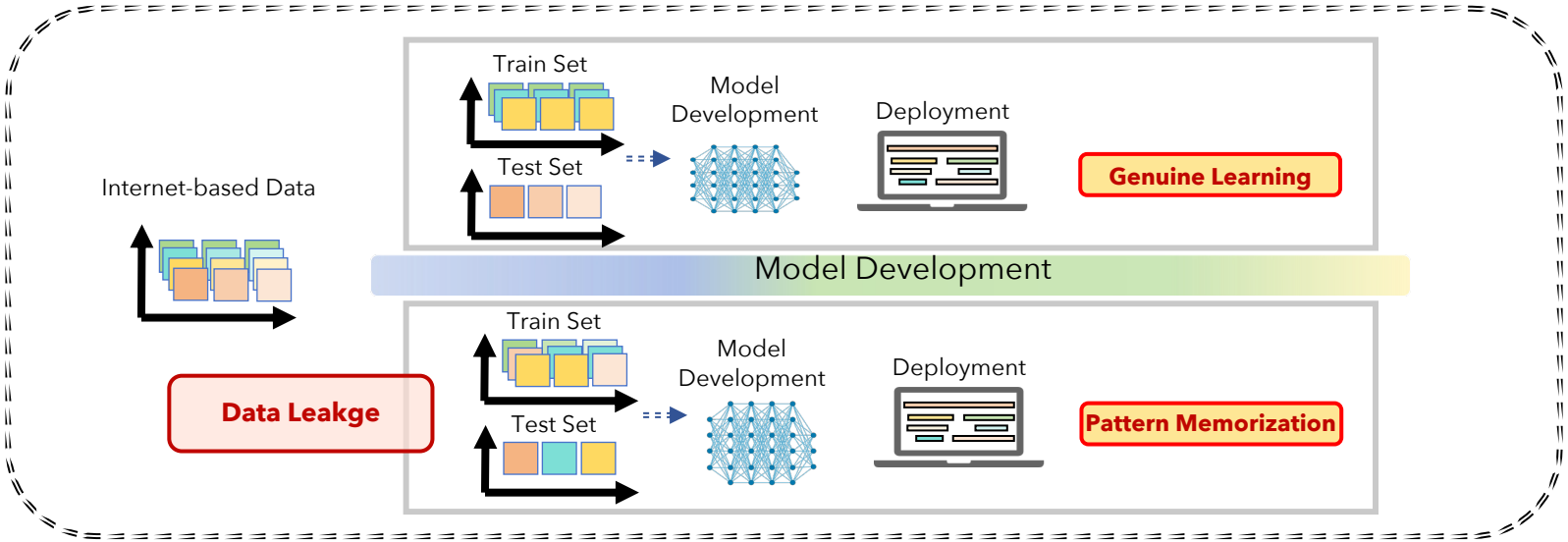




➢ Learning-Based Approaches Rely on lexical cues, use them as spurious features!

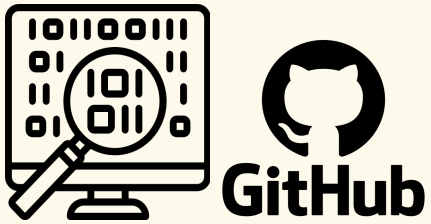
➢ Data Leakage: Enables models to memorize recurring patterns rather than generalize leading to inflated performance metrics.

Memorization ≠ generalization



1 Secret Detection: Proposed Framework

Static Analysis Tools



GitHub

1

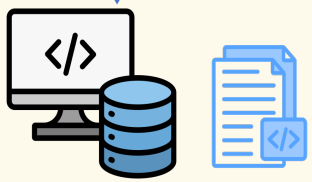
SecretBench: Extracting potential secrets using

- **761 regex patterns**

((xoxb|xoxp|xapp|xoxa|xoxr)[09]10,13[a-zA-Z0-9]***)**

\b(aio_[a-zA-Z0-9]{28})\b

- **818 GitHub repositories**
- **49 programming languages**



2

Code files with embedded secrets

3

Dataset Creation

	Description
1	Length of the potential secret.
2	Is a potential secret present in multiple lines?
3	Shannon entropy value of the secret.
4	Is potential secret a placeholder (such as "MY PASSWORD")?
5	Is a potential secret part of a URL (e.g., http://user:pwd@host.com)?
6	Does potential secret have parentheses? (possible function call)
7	Does potential secret have brackets? (possible variable declaration)
8	Does potential secret have period? (possible function call)
9	Does the secret have the word "Password"? (possible placeholder or variable)
10	Does potential secret have white spaces? (possible sentence/comment)
11	Does potential secret have -----BEGIN PRIVATE KEY-----?
12	Does secret begin with a $ sign? (possible variable)
13	Does secret contain null/undefined/None/true/false?
14	Does potential secret have an arrow (->, =>, ::)? (possible pointer variable)
15	Is a potential secret a numerical value?
16	Is config in the secret's file name (including path)?
17	Is test or example in the file name (including path)?

Key = "------Secret------"

```
client = DropletKit::Client.new(access_token: token)
chrome.storage.local.get.returns({
 [SB_LRU_CACHE_KEY]: JSON.stringify([
  {
key = " **********SECRET********** "
value: "adguard-malware-shavar" }
 ])
});
client.cdns.update_ttl(id: '19f06b6a-3ace')
await Safebr;
```

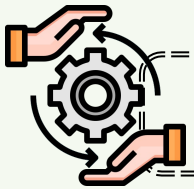
100 Characters

100 Characters

AI-based Approaches



Manual Inspection



6

Model Development

5

Preprocessing

Feature Engineering

Tokenization

Upsampe

Label Encoding

Deduplicated Dataset

4

Duplication Analysis

1 Secret Detection: Proposed Framework

Static Analysis Tools

AI-based Approaches

GitHub

1 **SecretBench**: Extracting potential secrets using

- 761 regex patterns

((xoxb|xoxp|xapp|xoxa|xoxr)[09]10,13[a-zA-Z0-9]*)

2 Code files with embedded secrets

3 Dataset Creation

	Description
1	Length of the potential secret.
2	Is a potential secret present in multiple lines?
3	Shannon entropy value of the secret.
4	Is potential secret a placeholder (such as "MY PASSWORD")?
5	Is a potential secret part of a URL (e.g., http://user:pwd@host.com)?
6	Does potential secret have parentheses? (possible function call)
7	Does potential secret have brackets? (possible variable declaration)
8	Does potential secret have period? (possible function call)
9	Does the secret have the word "Password"? (possible placeholder or variable)
10	Does potential secret have white spaces? (possible sentence/comment)
11	Does potential secret have -----BEGIN PRIVATE KEY-----?
12	Does secret begin with a $ sign? (possible variable)
13	Does secret contain null/undefined/None/true/false?
14	Does potential secret have an arrow (->, =>, ::)? (possible pointer variable)
15	Is a potential secret a numerical value?
16	Is config in the secret's file name (including path)?

```
client = DropletKit::Client.new(access_token: token)
chrome.storage.local.get.returns({
 [SB_LRU_CACHE_KEY]: JSON.stringify([
  {
key = " **********SECRET********** "
value: "adguard-malware-shavar" }
 ])
});
client.cdns.update_ttl(id: '19f06b6a-3ace')
await Safebr;
```

100 Characters

100 Characters

4 Duplication Analysis

5 Preprocessing

Feature Engineering

Tokenization

6 Model Development

Manual Inspection

Deduplicated Dataset

RQ_{21}: To what extent does code duplication impact the performance and generalization of machine learning models for secret detection?

1 Secret Detection: Motivation & Methodology

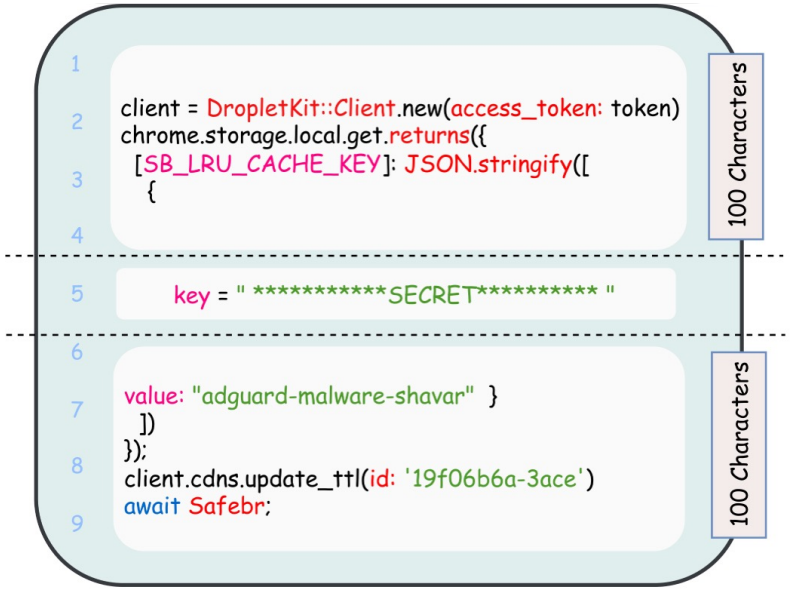
Let C = {c1, c2, . . . , cn} be the multiset of all n contexts in the dataset.

1. Exact Duplicates (C_{exact}): Contexts that are identical to one or more other contexts in the dataset.
   - We also define a set of representatives (R_{exact}) from the multiset C_{exact} containing one copy of every duplicated context and $R_{exact} \subseteq C_{exact}$.

2. Near Duplicates (C_{near}): Near duplicates are semantically similar contexts that are not identical and differ only in a few aspects.

3. Uniques (C_{unique}): A context is unique if it does not exactly match any other secret context. Uniques are sample contexts in $C_{\neg exact}$ that are neither exact duplicates nor near duplicates of any other context in the dataset.

Example of a secret code context



$$C = C_{unique} \cup C_{near} \cup C_{exact}$$

$$C_{dedup} = C_{unique} \cup C_{near} \cup R_{exact}$$

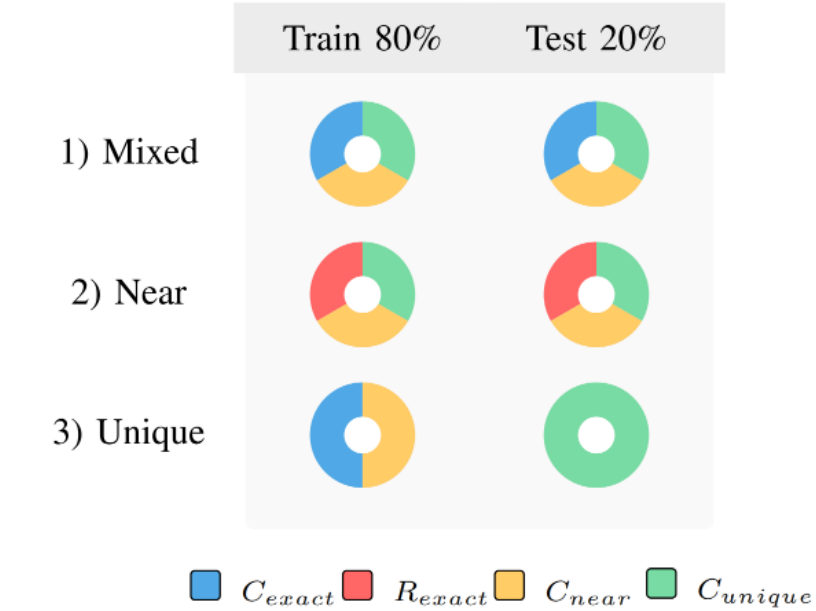
Secret Detection: Motivation & Methodology

⚠ **SecretBench contains 78% duplicate or near-duplicate samples — models memorize instead of generalizing.**

Duplications in the SecretBench Dataset.

Duplication Types	Count (%)
C_{unique}	21,484 (22.0%)
C_{exact}	67,523 (69.3%)
C_{near}	8,472 (8.7%)
Total	**97,479 (100.0%)**

1. **Mixed:** measuring model performance in the presence of all duplicates, simulating a naive evaluation on unpreprocessed data.
2. **Near Duplicate:** This scenario establishes the model's baseline performance without exact duplications. Training and Testing on datasets containing different, but semantically similar, instances.
3. **Unique:** measuring the model's ability to generalize from redundant data to entirely novel samples.



1 Secret Detection: Results

RQ_{21}: To what extent does code duplication impact the performance and generalization of machine learning models for secret detection?

Mixed Scenario Results

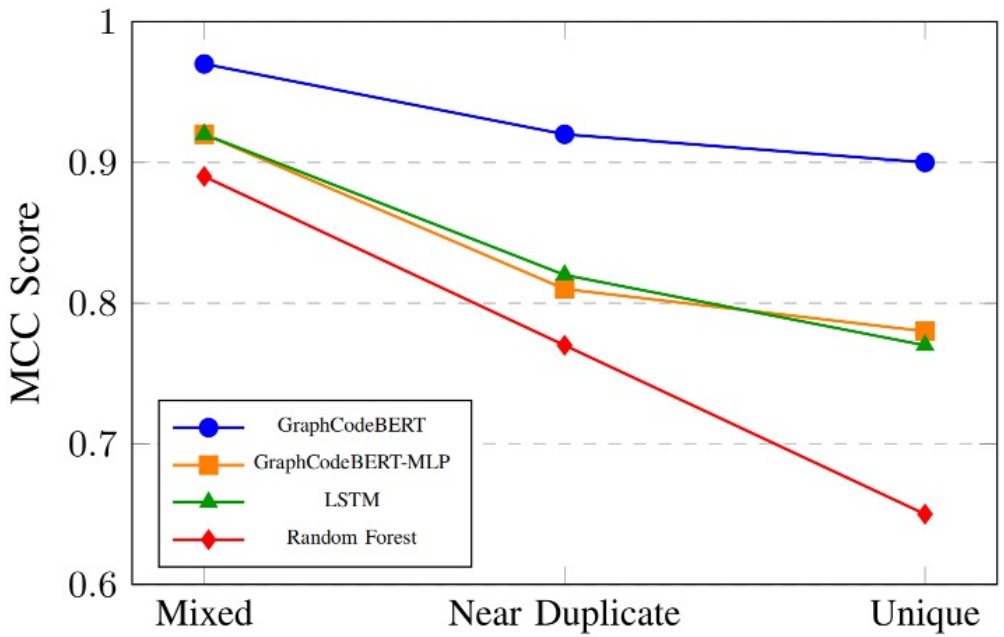
Classifier	Precision	Recall	F1-Score	MCC
GraphCodeBERT	0.97	0.98	0.97	0.97
GraphCodeBERT-MLP	0.92	0.93	0.93	0.92
Random Forest	0.96	0.86	0.91	0.89
LSTM	0.92	0.94	0.93	0.92

Duplicate Scenario Results

Classifier	Precision	Recall	F1-Score	MCC
GraphCodeBERT	0.91	0.96	0.93	0.92
GraphCodeBERT-MLP	0.85	0.83	0.84	0.81
Random Forest	0.90	0.71	0.79	0.77
LSTM	0.84	0.84	0.84	0.82

Unique Scenario Results

Classifier	Precision	Recall	F1-Score	MCC
GraphCodeBERT	0.91	0.92	0.91	0.90
GraphCodeBERT-MLP	0.82	0.80	0.81	0.78
Random Forest	0.85	0.56	0.67	0.65
LSTM	0.82	0.79	0.80	0.77



MCC Drop: Mixed → Unique

Random Forest	0.89 → 0.65	−27%
LSTM	0.92 → 0.77	−16%
GraphCodeBERT-MLP	0.92 → 0.78	−14%
GraphCodeBERT	0.97 → 0.9	−7%

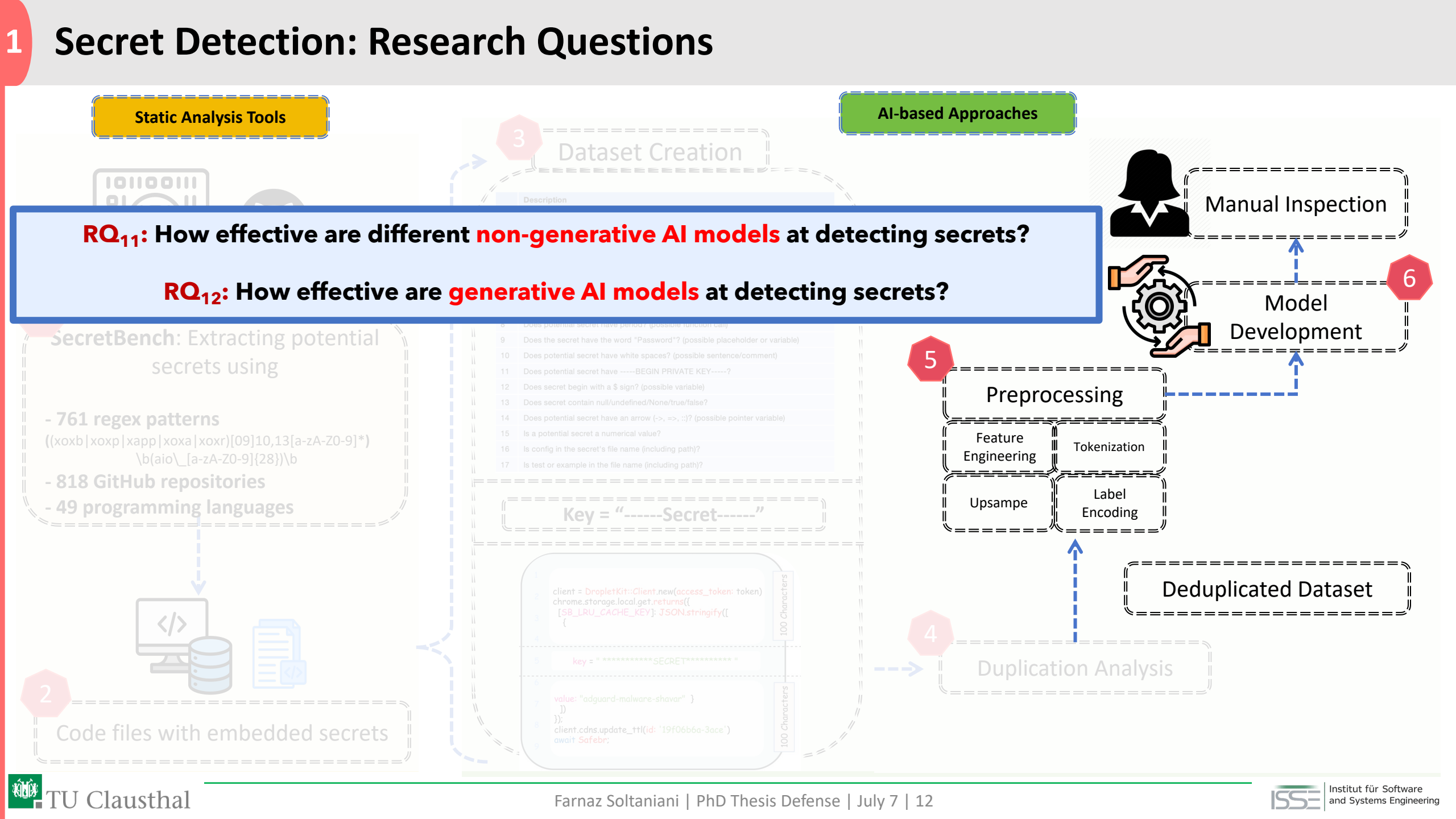
1 Secret Detection: Research Questions

Static Analysis Tools

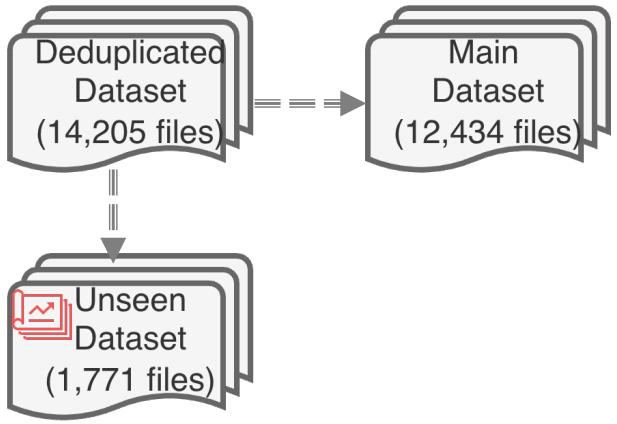
AI-based Approaches

RQ_{11}: How effective are different non-generative AI models at detecting secrets?

RQ_{12}: How effective are generative AI models at detecting secrets?



1 Secret Detection: Training & Testing Strategy



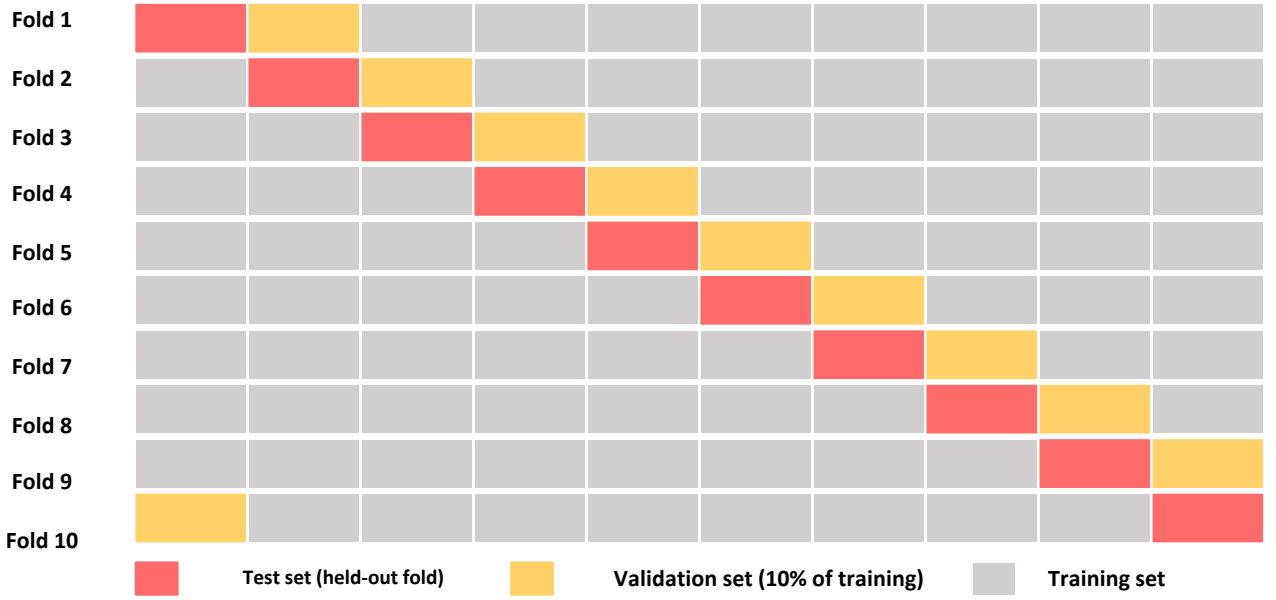
Category	Python	Java	Java Script	Ruby	Nix	C#	Total
API Keys	158 (138)	96 (46)	304 (267)	110 (105)	4 (4)	38 (34)	**710 (594)**
Authentication Key and Token	148 (64)	142 (71)	78 (55)	60 (38)	1 (0)	31 (19)	**460 (247)**
Database and Server URL	158 (14)	770 (7)	111 (12)	46 (2)	19 (0)	24 (1)	**1128 (36)**
Generic Secret	16 (9)	30 (9)	14 (11)	9 (5)	16 (16)	11 (9)	**96 (59)**
Password	13 (4)	21 (13)	11 (7)	24 (2)	3 (0)	7 (4)	**79 (30)**
Private Key	235 (168)	103 (50)	332 (220)	35 (16)	44 (44)	72 (38)	**821 (536)**
Other	627 (25)	1373 (25)	1856 (40)	2533 (4)	2935 (4)	1588 (9)	**10911 (107)**
Total	**1355 (422)**	**2535 (221)**	**2706 (612)**	**2817 (172)**	**3021 (68)**	**1771 (114)**	**14205 (1609)**

Secrets are rare:

- Class imbalance biases models and a single train-test split yields overly optimistic estimates.

10-fold cross-validation on the main dataset.

Splits are consistent across all experiments.



1 Secret Detection: Evaluated Models

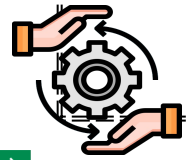
Traditional ML & Modern LLMs (proprietary, fine-tuned)

Models Evaluated

Random Forest	Ensemble	Traditional ML
Voting Classifier	LR + NB + SVM	Traditional ML
LSTM	Deep Learning	Sequential model
CodeBERT	Transformer	Semantic model
GPT-4.1	Generative LLM	Via API
Gemini 2.5	Generative LLM	Via API
DeepSeek-V3	Generative LLM	Via API

2 sampling variants × 3 feature sets × 4 learners × 10 folds = **240 models trained & evaluated**

Non-generative and Generative models are Evaluated on identical dataset splits.



The Prompt (standardised, applied to all three models)

Role Specification

```
You are a Senior Security Auditor.
  A lightweight static analysis tool has flagged a potential
  hardcoded secret in a given code snippet.
  Hardcoded secrets include API keys, tokens, passwords,
  credentials, or other sensitive values that should not
  be included in the source code.
```

Clarification

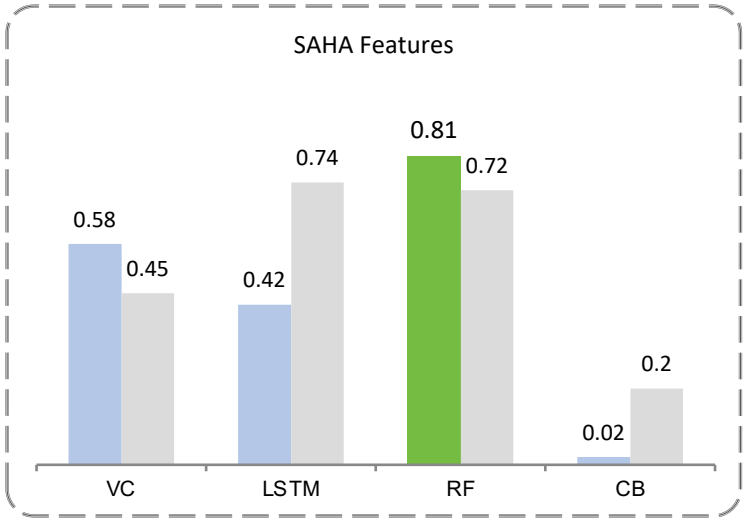
Guided Breakdown of Rules

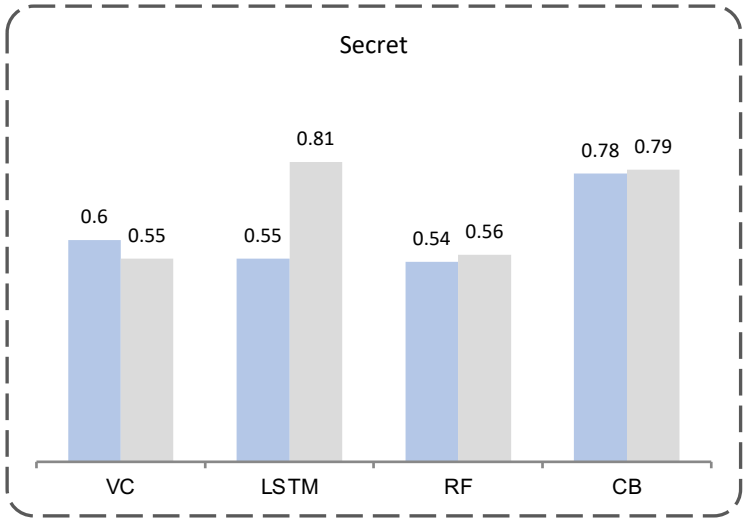
```
Tasks:
  Your task is to independently confirm or reject the
  presence of the secret in the code snippet.
  Only evaluate the reported secret and do not consider
  any other potential secrets in the code.
Input Format:
  { "CodeSnippet": "string",  "Hardcoded secret": "string" }
Response Format:
```

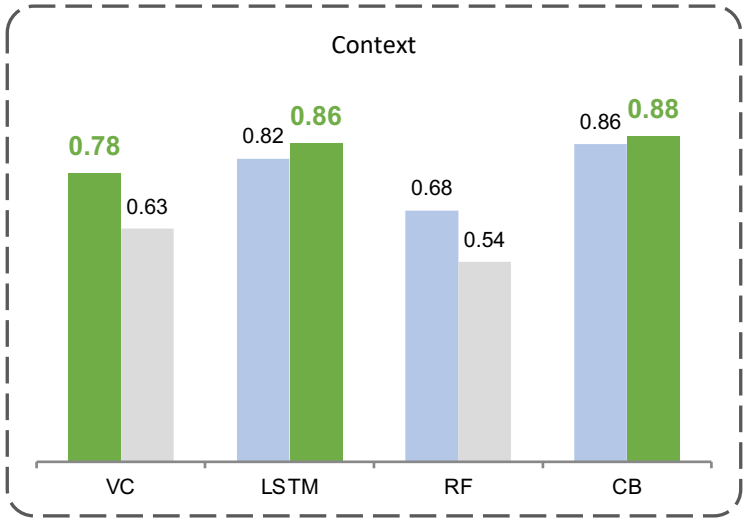
Control of Verbosity

```
  { "Response":
  {"IsSecret": "Yes" | "No",
  "Type": "string (e.g., API Key, Token, Password…)"}
  }
```

1 Secret Detection: Non-generative Model Comparison



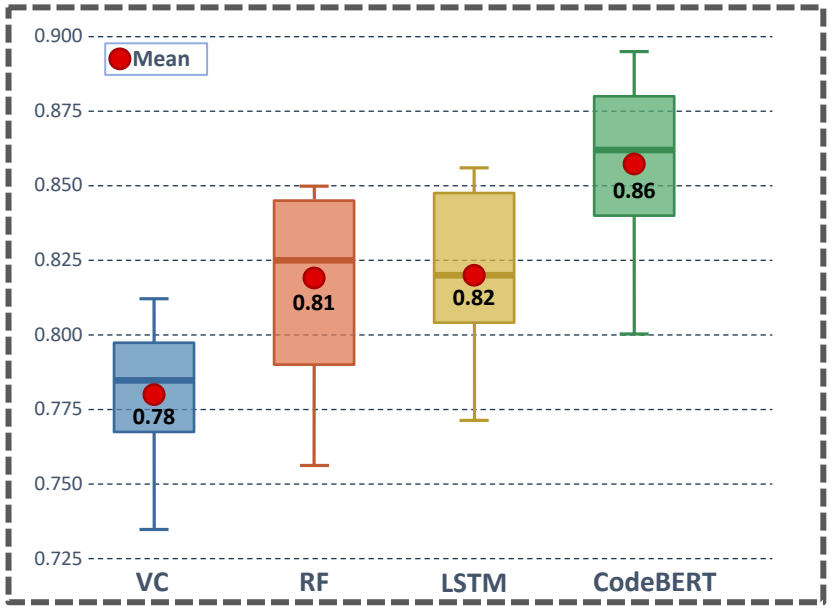




RQ_{11}: How effective are different non-generative AI models at detecting secrets?

- RF excels with SAHA features.
- Context boosts all DL models and CodeBERT outperforms all other models

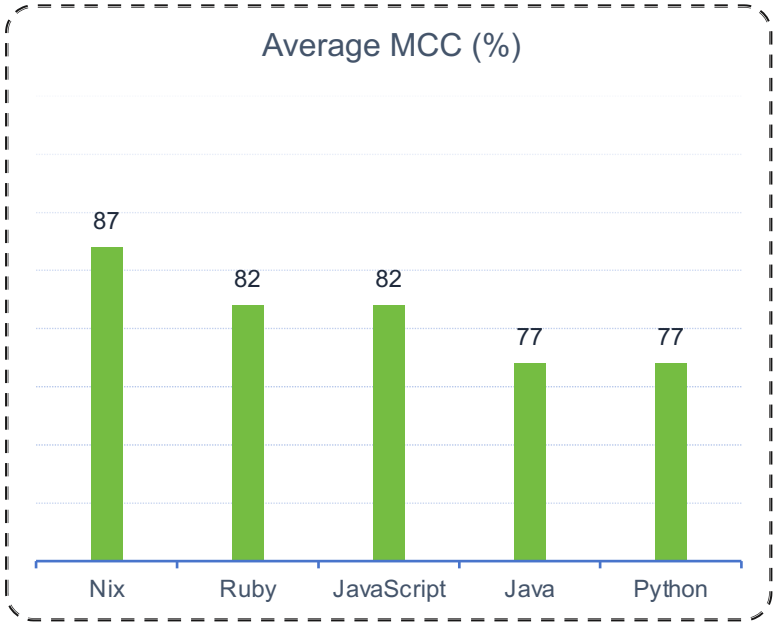
- Despite similar average MCC, the models are not consistent across folds.
- CodeBERT exceeds the median of every other model.

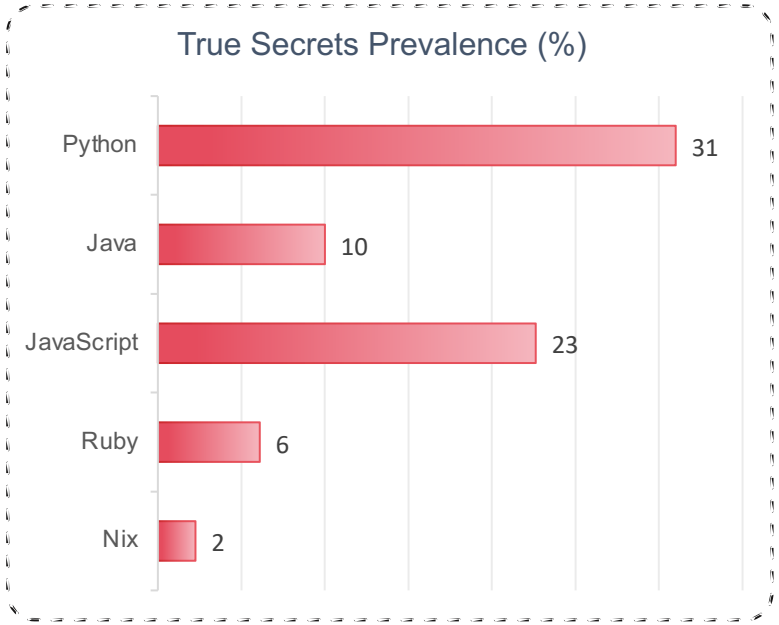


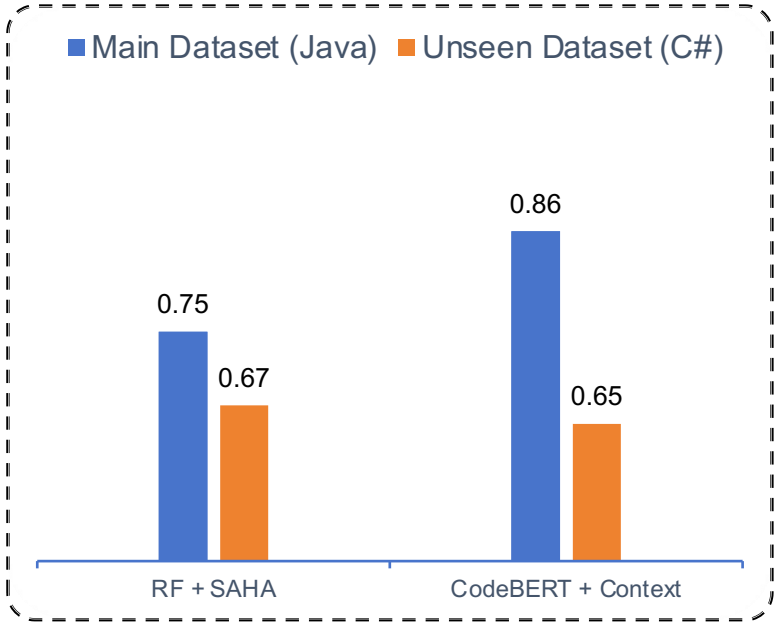
The distribution of the MCC across 10 folds for the top models.

1 Secret Detection: Generalisation to Unseen Language (C#)

- Language structure influences performance more than secret prevalence







C# vs. Java: C# is statically typed and object-oriented, whereas Python, Ruby, and JavaScript are **dynamically typed** and **scripting-oriented**.

- CodeBERT falls 21% (0.86 → 0.65)
- RF drops 8% (0.75 → 0.67)

1 Secret Detection: Generative Models — High Recall, Low Precision

RQ_{12}: How effective are generative AI models at detecting secrets?

- All three models are quite similar; they have very high recall but low precision.
- Test-Path Files: The #1 Source of False Positives

GPT		DeepSeek		Gemini	
MCC:	0.63	MCC:	0.61	MCC:	0.59
F1:	0.65	F1:	0.62	F1:	0.61
Precision:	0.49	Precision:	0.46	Precision:	0.45
Recall:	0.95	Recall:	0.97	Recall:	0.93

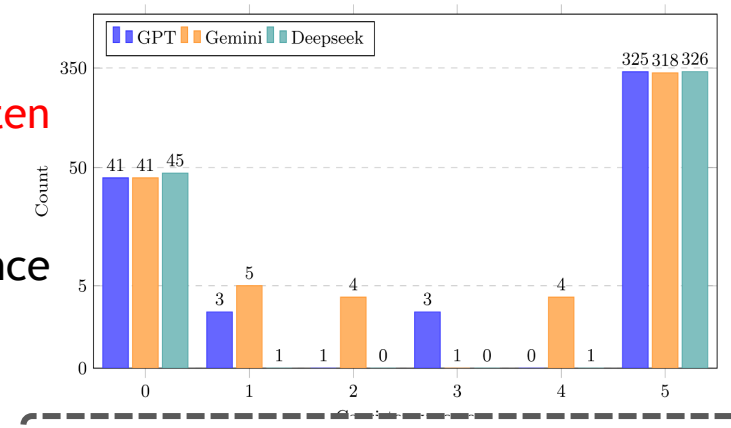
27%
Of samples are in test-path files

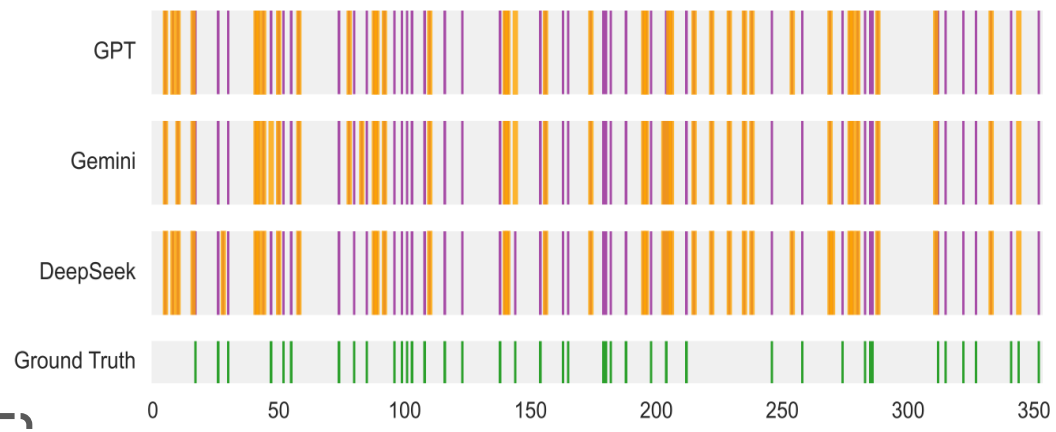
74.7%
of all errors from test-path files

76%
FP reduction when excluding test-path

0.77
GPT MCC excl. test-path (vs 0.63)

- Generative AI models are often consistent in secret prediction.
- They show comparable performance and make similar mistakes.





Determinism Analysis of Generative Models

1 Secret Detection: Manual Inspection of Incorrect predictions

A proportional stratified sampling of the misclassifications -> Yielding 298 FPs and 13 FNs

FALSE Negatives

13 cases (Total-FNs: 68 total)

(7) Idempotency Tokens / Public Identifiers
(4) Variable names didn't resemble secrets (e.g. named 'example')
(2) Incomplete PEM headers or connection URLs

FALSE Positives

298 cases (Total-FPs: 1,495 total)

(160) Demo/placeholder values flagged due to variable name bias
(78) Credentials embedded in URLs (treated as risky practice)
(51) Strings matching known secret patterns (PEM, AKIA, etc.)

Client request token used to ensure the idempotency of API operations.

```
# This operation creates a new file system.

resp = client.create_file_system({
client_request_token: "a8ca07e4-61ec-4399-99f4-19853801bcd5",
file_system_type: "WINDOWS",
kms_key_id: "arn:aws:kms:us-east-1:012345678912:key/0ff3ea8d-130e
    -4133-877f-93908b6fdbd6",
security_group_ids: [
```

The modael found the secret value in a publicly available Microsoft documentation.

```
public void getKeyWithResponse() {
    KeyAsyncClient keyAsyncClient = createAsyncClient();
    // BEGIN: com.azure.security.keyvault.keys.KeyAsyncClient.
        getKeyWithResponse#String-String
    String keyVersion = "6A385B124DEF4096AF1361A85B16C204";

    keyAsyncClient.getKeyWithResponse("keyName", keyVersion)
    .contextWrite(Context.of("key1", "value1", "key2", "value2"))
```

URL containing dummy credentials

```
    @uri.scheme = "ftp"
    expect(@uri.scheme).to eq("ftp")
    expect(@uri.to_s).to eq(
      "ftp://user:pass@example.com/path/to/resource?query=x#
          fragment"
    )
    expect(@uri.to_str).to eq(
      "ftp://user:pass@example.com/path/to/resource?query=x#
          fragment"
```

1 Secret Detection: Manual Inspection of Incorrect predictions

A proportional stratified sampling of the misclassifications -> Yielding 298 FPs and 13 FNs

FALSE Negatives
13 cases (Total-FNs: 68 total)

(7) Idempotency Tokens / Public Identifiers

FALSE Positives
298 cases (Total-FPs: 1,495 total)

(160) Demo/placeholder values flagged due to variable name bias

We adopt Tree-of-Thought (ToT) prompting, which generalizes over chain-of-thought (CoT), and reduces reliance on superficial token-level features and promotes context-aware, semantics-driven decision making.

We observed an Average MCC of 0.71 with the engineered prompt versus 0.59 with the basic prompt.

```
kms_key_id: "arn:aws:kms:us-east-1:012345678912:key/0ff3ea8d-130e
    -4133-877f-93908b6fdbd6",
security_group_ids: [
```

The modael found the secret value in a publicly available Microsoft documentation.

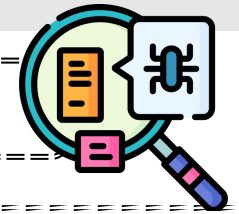
```
public void getKeyWithResponse() {
    KeyAsyncClient keyAsyncClient = createAsyncClient();
    // BEGIN: com.azure.security.keyvault.keys.KeyAsyncClient.
        getKeyWithResponse#String-String
    String keyVersion = "6A385B124DEF4096AF1361A85B16C204";

    keyAsyncClient.getKeyWithResponse("keyName", keyVersion)
    .contextWrite(Context.of("key1", "value1", "key2", "value2"))
```

```
    @uri.scheme = "ftp"
    expect(@uri.scheme).to eq("ftp")
    expect(@uri.to_s).to eq(
      "ftp://user:pass@example.com/path/to/resource?query=x#
          fragment"
    )
    expect(@uri.to_str).to eq(
      "ftp://user:pass@example.com/path/to/resource?query=x#
          fragment"
```

2 Security Bug Report Prediction

Monitoring issue tracker submissions is a crucial software maintenance activity



GitHub

Jira

bugzilla/**bugzilla**

Official repository for the Bugzilla bug tracking system. Report bugs to





Created by **developers**, **testers** & **users**

Camel / CAMEL-3343

CxfRsInvoker produces NullpointerException when no ContinuationProvider is set in the InMessage

Summary

Severity

Details

Type:	Bug	Status:	CLOSED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.5.0	Fix Version/s:	2.6.0
Component/s:	camel-cxf		
Labels:	None		
Patch Info:	Patch Available		

Description

Description

In my usage of CXF-RS with Camel through the camel-cxf component I ran into a NullpointerException, the specific case is described in some detail on StackOverflow: http://stackoverflow.com/questions/4198461/why-do-i-get-a-nullpointerexception-when-invoking-the-cxf-rs-endpoint-of-a-camel

Specifically I run into a NullpointerException:

```
Caused by: java.lang.NullPointerException
        at org.apache.camel.component.cxf.jaxrs.CxfRsInvoker.getContinuation(CxfRsInvoker.java:63)
        at org.apache.camel.component.cxf.jaxrs.CxfRsInvoker.performInvocation(CxfRsInvoker.java:52)
        at org.apache.cxf.service.invoker.AbstractInvoker.invoke(AbstractInvoker.java:89)
        ... 33 more
```

I was assuming I was misconfiguring something but didn't find any alternative way so I started digging in the camel-cxf source code.

I found that such a "getContinuation" method also exists for the `org.apache.camel.component.cxf.CxfConsumer` as well as for the CxfRsInvoker (where it was failing for me). However in the case of the CxfConsumer there is a guard on the ContinuationProvider so that if it is null, then null is returned:

```
return provider == null ? null : provider.getContinuation();
```

CxfRsInvoker does not have this guard, but it does have code to deal with a "null" Continuation at the calling site (line 52, in performInvocation). Therefore I assumed this was a bug and patched the guard into the CxfRsInvoker as well.

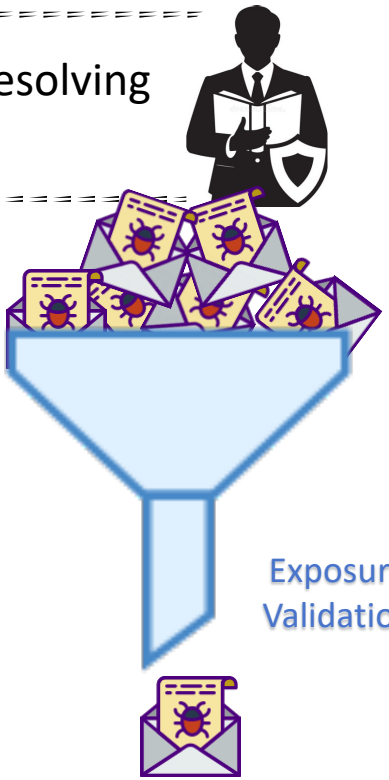
My program now seems to work correctly. I assume this is a bug.

Bug reports (BRs) are detailed documents including **what went wrong**, how to **reproduce** it, and the **environment** where it happened.

A key goal is the **prioritization** of high risk **security-related bugs**.

2 Security Bug Report Prediction: What are the Challenges?

1. Reporters Lack Security Expertise and usually only a small group of developers are involved in resolving security issues.



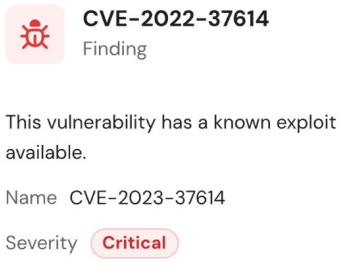
2. The scarcity of SBRs in large and diverse datasets of Bug reports leads to severe class imbalance, which limits the effectiveness of predictive models

Key Difference Keywords

Security	Non-Security
Vulnerability	UI glitch
Unauthorized	Performance
Exploit	Crash
Data breach	Typo
Privilege Escalation	Hang
XSS	Alignment
Auth failure	Incorrect Behaviour
...	...

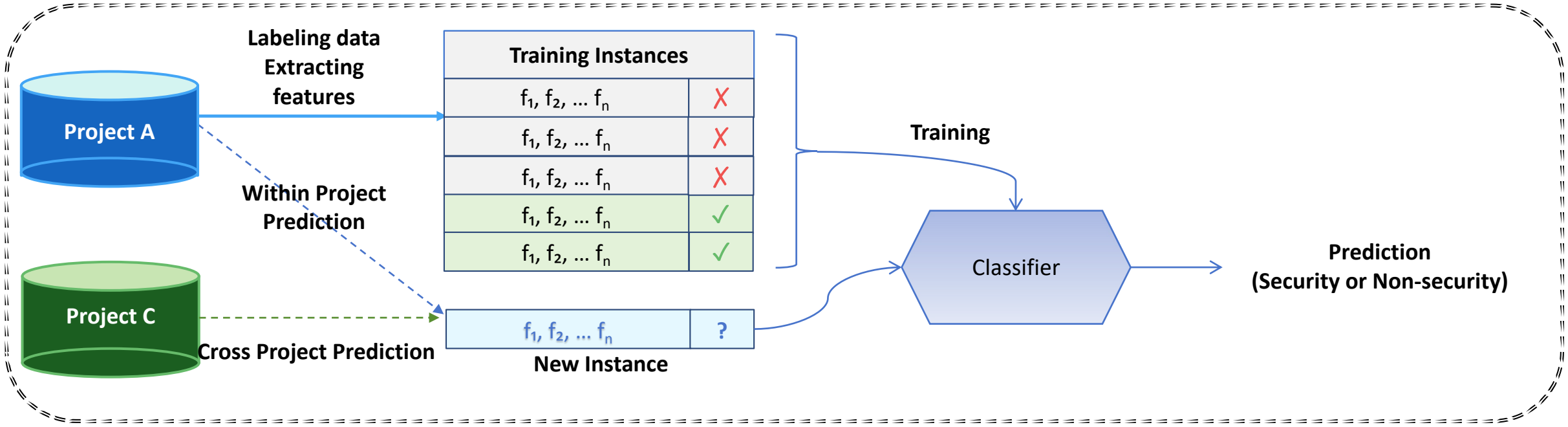
3. Implicit Security Signals: ML classifiers rely on lexical cues, using them as spurious features!

"This is a security concern."
"This is not a security mater."



4. Ambiguity & Unstructured Format; Reports range from well-documented issues to vague, incomplete descriptions

2 Security Bug Report Prediction: Methodology

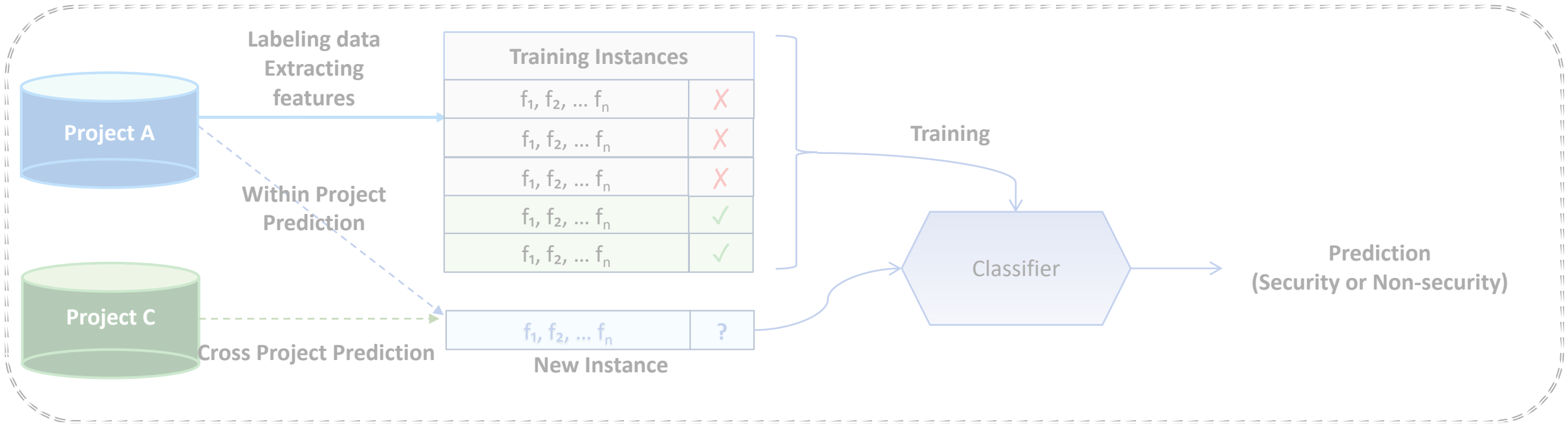


WPP: When a **substantial amount** of historical data is available for a project.

Augmentation: Expand the training dataset with additional SBRs or BRs.

Cross Project Prediction: Leveraging labeled BRs from one or more projects to predict unlabeled future bug reports for a target project.

2 Security Bug Report Prediction: Methodology



RQ_{11}: How effective are different non-generative AI models at predicting SBRs?
RQ_{12}:: How effective are generative AI models at predicting SBRs?

RQ_{21}: What is the impact of augmenting data and cross-project predictions on the performance of security bug report predictors?

Security Bug Report Prediction: Datasets

WPP

Target Dataset	SBR Train	NSBR Train	SBR Test	NSBR Test
Chromium	371	20599	437	20533
Derby	82	418	97	403
Camel	28	472	46	454
Ambari	40	460	16	484
Wicket	24	476	23	477

5 datasets × 7 models = 35 models

Augmented SBRs

Target Dataset	SBR Train	NSBR Train	SBR Test	NSBR Test
Chromium	727	20599	437	20533
Derby	1067	418	97	403
Camel	1118	472	46	454
Ambari	1108	460	16	484
Wicket	1165	476	23	477

75 augmentation × 5 models = 375 models

Augmented BRs

Target Dataset	SBR Train	NSBR Train	SBR Test	NSBR Test
Chromium	727	24243	437	20533
Derby	1067	44373	97	403
Camel	1118	44322	46	454
Ambari	1168	44292	16	484
Wicket	1141	44299	23	477

75 augmentation × 5 models = 375 models

CPP

Target Dataset	SBR Train	NSBR Train	SBR Test	NSBR Test
Chromium	356	3644	437	20533
Derby	985	43955	97	403
Camel	1090	43850	46	454
Ambari	1108	43832	16	484
Wicket	1117	43823	23	477

20 configuration × 5 models = 100 models

2 Security Bug Report Prediction: Models for SBR Prediction

Traditional ML & Modern LLMs (proprietary, fine-tuned)

Models Evaluated		
RF	Ensemble	State-of-the-art
RF + FARSEC	Ensemble	State-of-the-art
BERT	Transformer	Transformer architecture
BERT + FARSEC	Transformer	Transformer architecture
DistilBERT	Encoder (distilled)	Transformer architecture
DistilGPT-2	Decoder (distilled)	Transformer architecture
Qwen2.5-Coder-0.5B	Decoder + LoRA	Transformer architecture
Gemini 2.5	Generative LLM	Via API
GPT-4.1	Generative LLM	Via API

Fine-tuned and **proprietary** models are evaluated on identical dataset splits.

Hyperparameters tuned via Differential Evolution.

The Prompt (standardised, applied to all two models)

```
- Role: You are a Security Vulnerability Classification Expert
(CWE/CVE/STRIDE/MITRE ATT&CK).

- Definitions:
1) Security: The bug describes a vulnerability (e.g., null
pointer, improper memory handling, privacy leakage,
permission, etc.).
2) Non-security: The bug is a functional error, a usability
issue, a UI glitch, a performance issue, or a feature request,
with no clear exploit path.

- Task: determine if the bug report is security-relevant.

- Output only this JSON (no extra text):
{
''explanation'': ''concise reason (<= 20 words)'',
''classification'': ''security'' | ''non-security''
}

- Bug Report:
{Text}
```

Role Specification

Clarification

Guided Breakdown of Rules

Control of Verbosity

2 Security Bug Report Prediction: Fine-tuned Models (WPP)

RQ_{11}: How effective are different non-generative AI models at predicting SBRs?

- After **RF**, **Encoder LLMs** are most reliable for WPP SBR prediction.
- **FARSEC** (Removes SBR-like information from NSBRs with a **Time Complexity:** O(N×W)

Models	Recall	Precision	F1 Score	FPR	G-Meausre
Bert	0.34	0.58	0.41	0.02	**0.47**
RF	0.47	0.80	0.57	0.01	**0.63**
BERT + FARSEC	0.50	0.34	0.36	0.18	**0.54**
RF + FARSEC	0.61	0.63	0.57	0.11	**0.70**
DistillBert	0.36	0.75	0.47	0.01	**0.51**
CodeBert + MLP	0.56	0.20	0.18	0.46	0.34
Distillgpt2	0.20	0.23	0.21	0.06	0.25
Qwen2.5 + LORa	0.30	0.51	0.35	0.03	0.42

Models	G-Meausre				
	BERT	DistilBERT	DistilGPT	Qwen	RF
Chromium	**0.83**	**0.85**	**0.88**	**0.87**	0.75
Derby	0.57	0.56	0.18	0.43	0.62
Camel	0.33	0.33	0.04	0.26	0.49
Ambari	0.40	0.32	0.12	0.22	0.60
Wicket	0.23	0.47	0.04	0.30	0.68
Average	0.47	0.51	0.25	0.42	0.63

- **Individual datasets** exhibit substantial variability in model effectiveness. All models achieve their best scores on **Chromium.**

2 Security Bug Report Prediction: Fine-tuned Models (Augmentation)

RQ_{21}: What is the impact of augmenting data and cross-project predictions on the performance of security bug report predictors?

Augmented SBRs:

Models	Recall	Precision	F1 Score	FPR	G-Meausre
Bert	0.45	0.48	0.45	0.05	0.59
RF	0.55	0.64	0.55	0.03	**0.70**
DistillBert	0.59	0.52	0.53	0.06	**0.72**
CodeBert + MLP	0.34	0.21	0.16	0.21	0.39
Distillgpt2	0.52	0.47	0..49	0.06	0.65
Qwen2.5 + LORa	0.38	0.41	0.38	0.06	0.51

➢ **Additional SBRs** improved the performance of all models.

➢ **Adding more BRs** improves performance for all models, but decreases for **RF**.

Augmented BRs:

Models	Recall	Precision	F1 Score	FPR	G-Meausre
Bert	0.52	0.72	0.60	0.02	0.66
RF	0.32	0.69	0.41	0.01	0.46
DistillBert	0.50	0.66	0.56	0.01	0.64
CodeBert + MLP	0.52	0.15	0.22	0.28	0.56
Distillgpt2	0.60	0.68	0.64	0.02	**0.72**
Qwen2.5 + LORa	0.51	0.89	0.61	0.00	0.64

G-measure for all models$_{\text{Augmented BRs}}$ > G-measure for all models$_{\text{Augmented SBRs}}$ > G-measure for all models$_{\text{WPP}}$

2 Security Bug Report Prediction: Fine-tuned Models (CPP)

RQ_{21}: What is the impact of augmenting data and cross-project predictions on the performance of security bug report predictors?

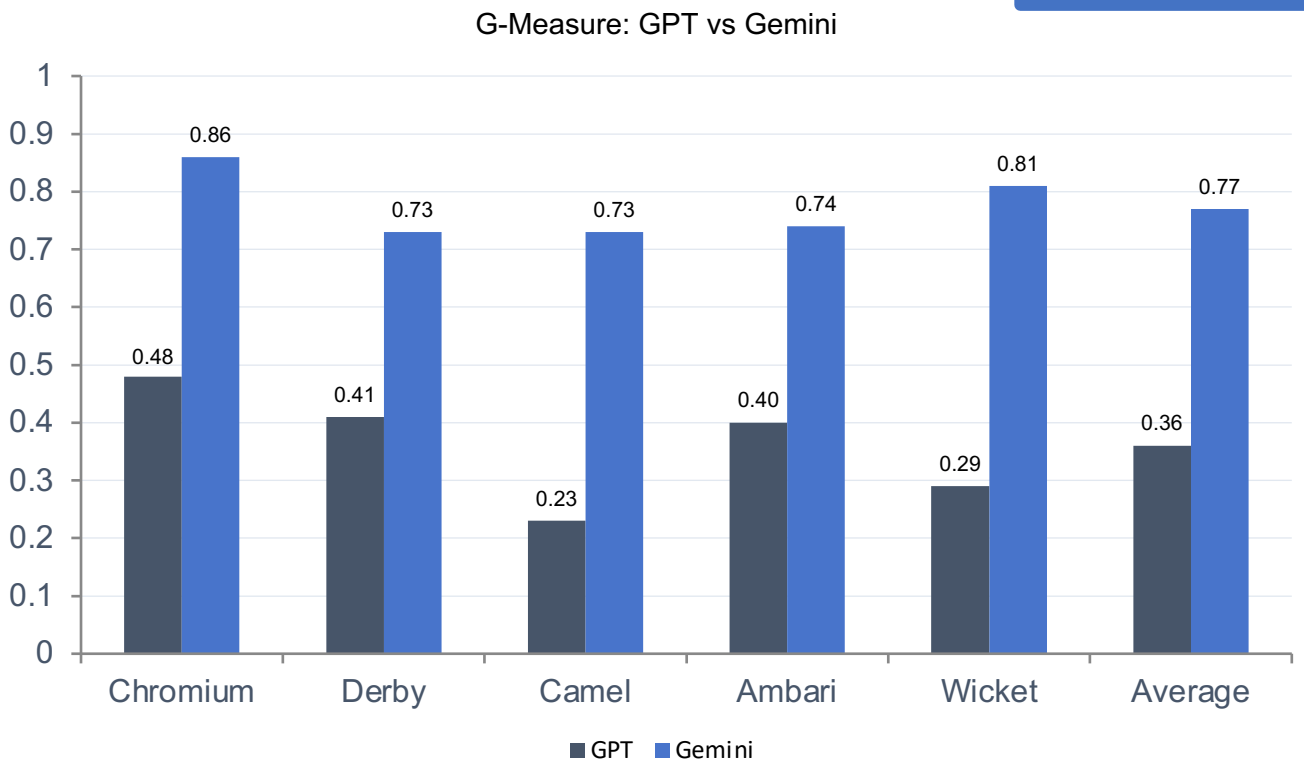
Cross Project SBRs Prediction:

Models	Recall	Precision	F1 Score	FPR	G-Meausre
Bert	0.48	0.62	0.49	0.02	0.62
RF	0.25	0.59	0.31	0.01	0.38
DistillBert	0.48	0.52	0.48	0.02	**0.63**
CodeBert + MLP	0.37	0.14	0.18	0.21	0.47
Distillgpt2	0.61	0.63	0.60	0.02	**0.74**
Qwen2.5 + LORa	0.66	0.48	0.47	0.00	0.60

- **Building a model based on one project may not effectively predict SBRs in another.**
- **Utilizing BRs from all other datasets for training resulted in higher performance than their WPP counterparts, except for RF, whose performance decreased from 0.63 (the best in WPP) to 0.38.**

2 Security Bug Report Prediction: Prompted Proprietary LLMs Results

RQ_{12}: How effective are generative AI models at predicting SBRs?

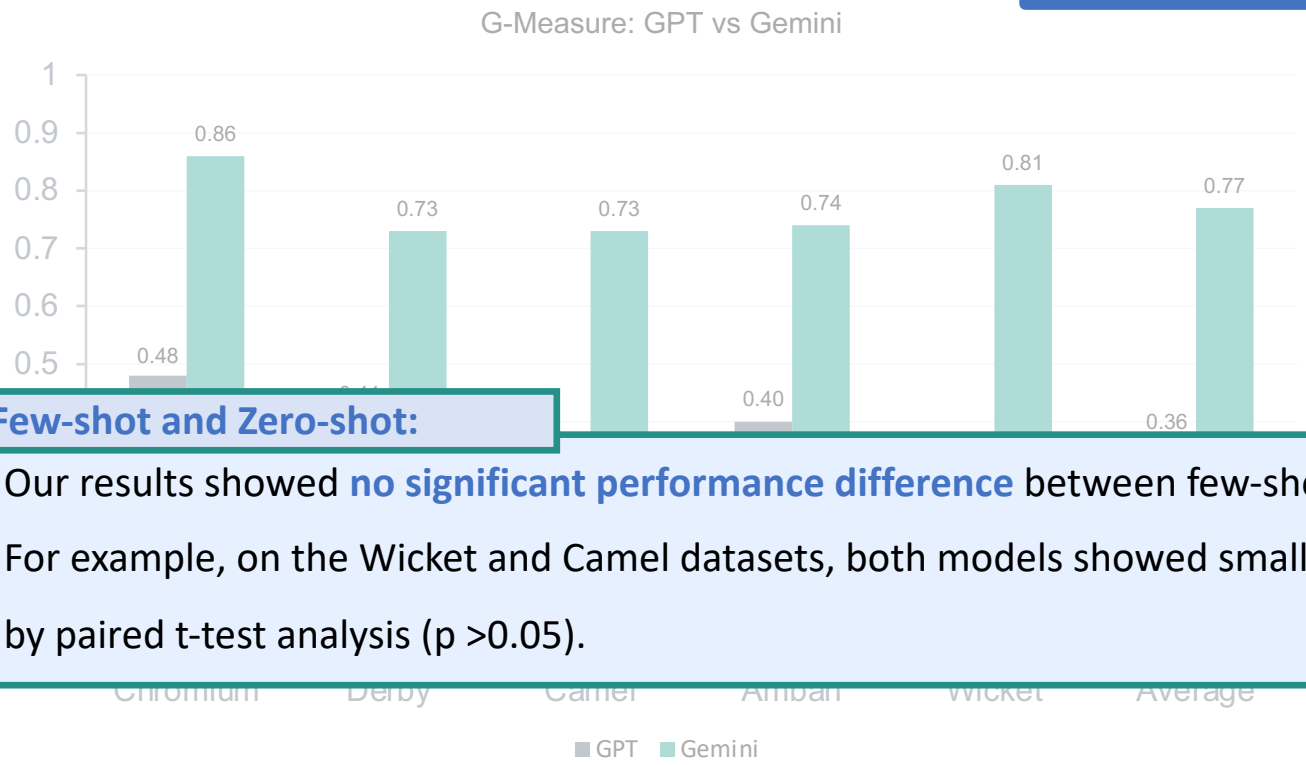


	GPT-4.1	Gemini 2.5
Avg Recall	0.23	0.74
Avg Precision	0.26	0.22
Avg FPR	0.05	0.18
Avg G-Measure	0.36	0.77
Cost (5 datasets)	~$25	~$71
Chromium time	19.5h	50h+

- **Gemini** detects **3× more** SBRs than GPT.
- Gemini's 77% G-measure surpasses all fine-tuned models, but its precision of 22% means most flagged reports are false alarms — **impractical at scale**.

2 Security Bug Report Prediction: Prompted Proprietary LLMs Results

RQ_{12}: How effective are generative AI models at predicting SBRs?



	GPT-4.1	Gemini 2.5
Avg Recall	0.23	0.74
Avg Precision	0.26	0.22
Avg FPR	0.05	0.18

Few-shot and Zero-shot:

Our results showed **no significant performance difference** between few-shot and zero-shot settings.

For example, on the Wicket and Camel datasets, both models showed small, statistically non-significant changes in **F1-scores** as confirmed by paired t-test analysis ($p > 0.05$).

- **Gemini** detects **3× more** SBRs than GPT.
- Gemini's 77% G-measure surpasses all fine-tuned models, but its precision of 22% means most flagged reports are false alarms — **impractical at scale**.

Security Bug Report Prediction: Agreement Distribution per Dataset

To assess the level of **agreement between the two prompted models**, we computed a consensus score for each sample across all datasets.

Score-2: Both models are in consensus

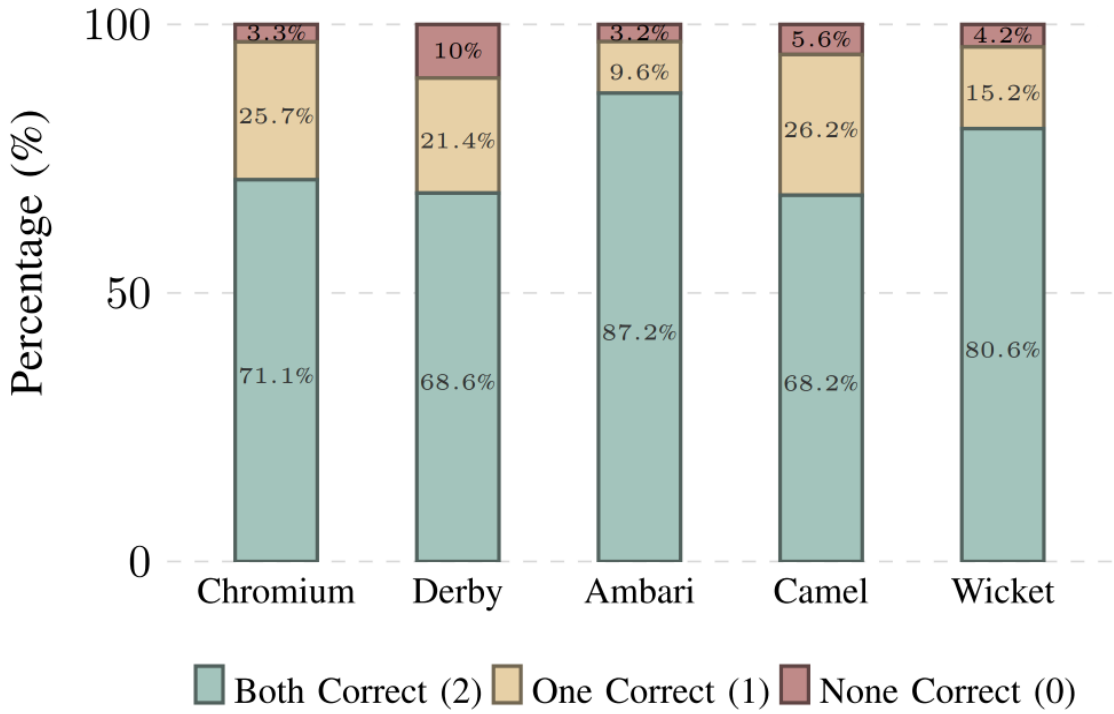
The **dominant** category. **Ambari** is the easiest dataset.

Score-1: One model is correct,

- Notable in **Camel** and **Chromium**.

Score-0: Neither model is correct.

- Notable in **Derby**.



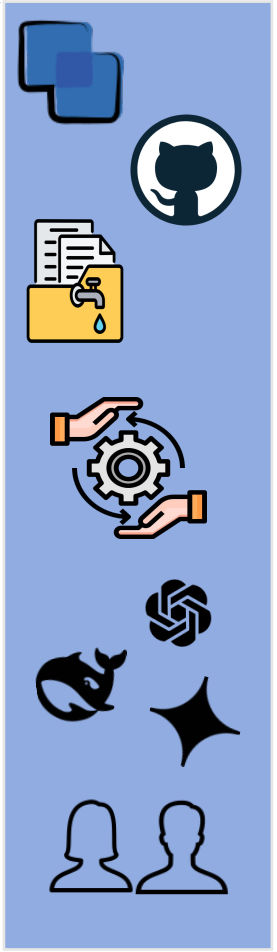
➢ Overall, although cases where **both models** were incorrect are **rare**, it was not Uncommon for only **one model** to be correct.

2 Security Bug Report Prediction: Prompted vs. Fine-tuned Models

Dimension	GPT-4 (Prompted)	Gemini (Prompted)	Fine-tuned LLMs
Speed per row	≈ 1.7 sec (faster per-row processing)	≈ 3–4 sec	10–47 min (full - initial computational investment for training)
Total (Chromium)	~19.5 hrs	~50 hrs	2–7 hrs train + inference
API Cost	≈ $25 (The total expenditure)	≈ $71	Local / Free
Training needed	None	None	Yes (one-time)
Data Privacy	⚠ 3rd-party API	⚠ 3rd-party API	✓ Local / On-prem

- **GPT** consistently achieved faster per-row processing and substantially lower total execution times, but **Fine-tuned Models** require an initial computational investment for training.
- **The total expenditure** for GPT was approximately **$25**, whereas Gemini incurred a cost of **$71.**
- **Fine-tuned Models:** Training fine-tuned small LLMs is possible on a local machine, requiring only moderate computational resources.

Conclusion



- **Duplications** can significantly **inflate** the reported performance of AI models.
- The exceptional performance reported in the literature may reflect **memorization** rather than **true learning** and we call for the reevaluation of secret detection models to address **data leakage** and ensure that models are reliable and applicable in **real-world settings.**
- When the distribution shift between the train and test data increases, more data is required and fine-tuning only the last layer is **not sufficient.** With sufficient data, training a model can yield better performance, but the choice of **feature set** greatly influences the performance of models.
- When **data is scarce**, generative models with prompt engineering provides a competitive alternative, but human oversight remains required to reliably identify vulnerabilities.
- We proposed the Augmentation and Cross-Project Learning Framework for SBR Prediction. Building a model based on one project may not effectively predict SBRs in another.

G-measure for all models$_{\text{Augmented BRs}}$ > G-measure for all models $_{\text{Augmented SBRs}}$ > G-measure for all models $_{\text{WPP}}$

- Synergy between the **ML/LLMs** and the **security analyzers**

- **Human-in-the-loop (persistent human feedback)**

Publications:

Emapirical Software Engineering Journal - EMSE '26

Learning to Detect Hardcoded Secrets

Farnaz Soltaniani · Mohammad Ghafari

Software Analysis, Evolution and Reengineering Conference - SANER '26

From Data Leak to Secret Misses: The Impact of Data Leakage on Secret Detection Models

Farnaz Soltaniani
Technische Universität Clausthal, Germany
farnaz.soltaniani@tu-clausthal.de

Mohammad Ghafari
Technische Universität Clausthal, Germany
mohammad.ghafari@tu-clausthal.de

Predictive Models and Data Analytics in Software Engineering (PROMISE '25), (Co-located with FSE)

Security Bug Report Prediction Within and Across Projects: A Comparative Study of BERT and Random Forest

Farnaz Soltaniani
farnaz.soltaniani@tu-clausthal.de
Technische Universität Clausthal
Germany

Mohammad Ghafari
mohammad.ghafari@tu-clausthal.de
Technische Universität Clausthal
Germany

Mohammed Sayagh
mohammed.sayagh@etsmtl.ca
ETS - Quebec University
Canada

Software Analysis, Evolution and Reengineering Conference - SANER-C '26

Evaluating Large Language Models for Security Bug Report Prediction

Farnaz Soltaniani
Technische Universität Clausthal
Germany
farnaz.soltaniani@tu-clausthal.de

Shoaib Razzaq
Technische Universität Clausthal
Germany
shoaib.razzaq@tu-clausthal.de

Mohammad Ghafari
Technische Universität Clausthal
Germany
mohammad.ghafari@tu-clausthal.de

Warm Thank You to My Committee Members:

TU Clausthal
Clausthal University of Technology

ISSE Institute for Software and Systems Engineering

Thesis Committee



Prof. Dr. Mohammad Ghafari



Prof. Dr. Mohammed Sayagh



Prof. Dr. Christoph Knieke



Prof. Dr. Benjamin Säfken

Future Directions

More than 10,000 Docker Hub container images expose data that should be protected, e.g., LLM model keys (ASIA CCS '25)



Developing CI/CD Architectures for container registries using code context before docker push and revoke mechanism.

Quantifying "**Toxic Knowledge**" in Code LLMs. Investigation of the extent to which current SotA coding models have memorized secrets.

How Credentials Are Leaked by LLM Agents? How prevalent is credential leakage in agents? To what extent are the identified credential leakage patterns exploitable in practice?

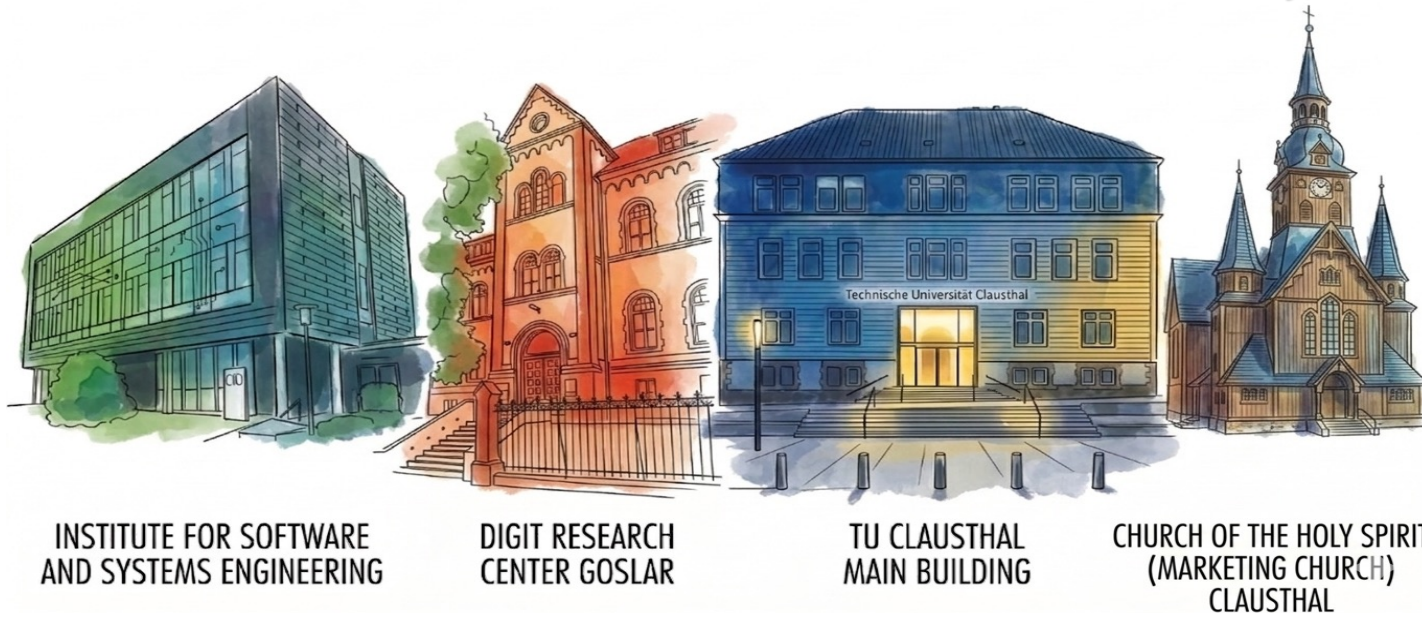
Mitigating Challenges in Experimentation with Machine Learning-Based Vulnerability Detectors (**how they count true/false positives**, **Using correct or incorrect performance metrics**, **How to use statistical significance tests**, **Bias detection and mitigation in human feedback**)

Thank You for Your Attention

Farnaz Soltaniani

PhD Viva Session @ Technische Universität Clausthal, Germany

7th of July, 2026



Chair of the Examiners: Prof. Dr. Benjamin Säfken

Thesis Supervisor: Prof. Dr. Mohammad Ghafari

2nd Reviewer: Prof. Dr. Christoph Knieke

3rd Reviewer: Prof. Dr. Mohammed Sayagh

Evaluation Metrics

Confusion Matrix • Precision • Recall • F1-score • MCC • G-measure

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	**TP** True Positive	**FN** False Negative
Actual Negative	**FP** False Positive	**TN** True Negative

Diagonal cells (green) = correct predictions. Off-diagonal cells (red) = errors.

Choosing the Right Metric

- Precision and Recall alone can mislead on imbalanced data
- F1-score balances Precision and Recall equally.
- **MCC uses all four confusion-matrix cells** — most robust to class imbalance.
- G-measure is a geometric alternative to F1.
- AUROC tells you how well the model ranks vulnerabilities generally.
- **Text datasets** for software repositories are heavily flooded with **non-bug text**. G-measure judges the model's accuracy on the bug class and non-bug class independently, then multiplies them.

Precision

$$TP / (TP + FP)$$

Of all predicted positives, how many are correct

Recall

$$TP / (TP + FN)$$

Of all actual positives, how many were found

F1-score

$$2 \times (P \times R) / (P + R)$$

Harmonic mean of Precision and Recall

G-measure

$$\sqrt{(Precision \times Recall)}$$

Geometric mean of Precision and Recall — an alternative to F1 that treats both metrics symmetrically

Matthews correlation coefficient (MCC)

$$(TP \cdot TN - FP \cdot FN) / \sqrt{[(TP+FP)(TP+FN)(TN+FP)(TN+FN)]}$$

It is the **ratio** between **the covariance of two variables** and the **product** of their **standard deviations;** the result always has a value between −1 and 1. covariance itself, the measure can only reflect **a linear correlation of variables**, and ignores many other types of relationships or correlations.

Vulnerability detection datasets are extremely imbalanced. AUROC includes True Negatives in its **denominator**. Because TNs are massive, the **False Positive Rate** stays artificially close to zero across many thresholds leading to a high AUROC.

1 Secret Detection Prevalence

- **GitGuardian published a report in March 2023 on how leaky 2022 was.**

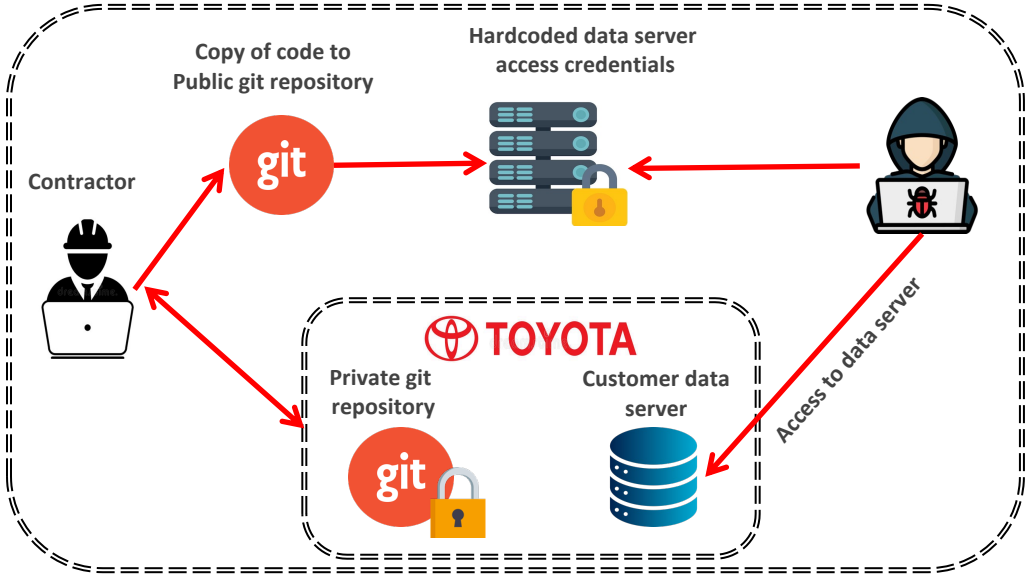
2022

10 M Secrets Occurrences detected in 2022	1 Out of 10 authors exposed a secret	1 Out of 1,000 commits exposed a secret

2024

23.8 M Secrets Occurrences detected in 2024	+25% Year-over-year increase in leaked secrets	70% Of 2022 leaked secrets still active today

- **Toyota Accidently Exposed A Secret Key Publicly On GitHub For Five Years**



- **Uber Accidently Exposed Hardcoded Credentials**

> **"There have been three reported breaches involving Uber in 2014, 2016, and now 2022. It appears that all three incidents critically involve hardcoded credentials (secrets) inside code and scripts"**
>
> Mackenzie Jackson –
> Security Advocate at GitGuardian

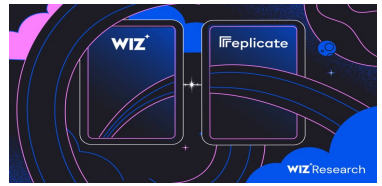


1 Secret Detection Prevalence

- **AI platforms** now face the same credential exposure risks as traditional services

11/ 2025: Wiz researchers examined 50 firms from the Forbes AI 50 list and confirmed that

65% of Leading AI Companies Found With Verified Secrets Leaks



02/ 2026: IBM X-Force 2026 Threat Intelligence Index

> **300,000** ChatGPT credentials observed for sale on the dark web



02/2026: Security breach in Moltbook AI social network

Exposed API key in client-side JavaScript enabled full read/write access



- 1.5M API tokens exposed
- 35,000 email addresses
- Full user identity data
- Write access to all posts

1 Secret Detection: Motivation & Methodology

RQ_{21}: To what extent does code duplication impact the performance and generalization of machine learning models for secret detection?

Let C = {c1, c2, . . . , cn} be the multiset of all n contexts in the dataset.

1. Exact Duplicates (C_{exact}): Contexts that are identical to one or more otl contexts in the dataset.
   - We also define a set of representatives (R_{exact}) from the multiset containing one copy of every duplicated context and $R_{exact} \subseteq C_{exa}$

2. Near Duplicates (C_{near}): Near duplicates are semantically similar conte that are not identical and differ only in a few aspects.

```
TestFramework):
    def __init__(self):
        super().__init__(4, 0)
        self.sporkprivkey = "XXX"

    def run_test(self):
        self.sporkAddress =
            self.nodes[0].getaccountaddress("")
```

```
        super().__init__(6, 5,
            extra_args=[['-debug=instantsend']]
            * 6 )
        self.sporkprivkey = "XXX"

    def run_test(self):
        self.sporkAddress =
            self.nodes[0].getaccountaddress("s")
```

1. **The values provided during initialization**
2. **Arguments passed to methods.**

Secret Detection: Motivation & Methodology

SAHA Features:

- Some of these features describe the **secret candidate itself** like entropy, length,... , while others describe its **syntactics and contextual** features like file path.
- Entropy measures the randomness of a string. For example ‘aaaaaaaaaa’ has very low entropy, while ‘AKJd92j!LsP0’ has a high entropy.
- Many non-secret strings are also highly random, SHA-1 hashes, Git commit IDs, UUIDs, Checksums. Therefore entropy produces many false positives.

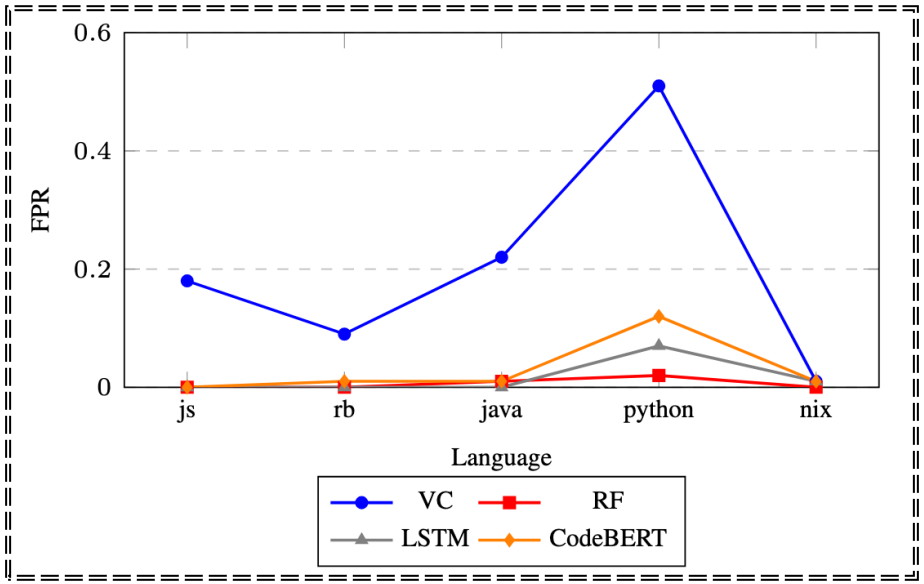
	Description
1	Length of the potential secret.
2	Is a potential secret present in multiple lines?
3	Shannon entropy value of the secret.
4	Is potential secret a placeholder (such as "MY PASSWORD")?
5	Is a potential secret part of a URL (e.g., ‘http://user:pwd@host.com)’?
6	Does potential secret have parentheses? (possible function call)
7	Does potential secret have brackets? (possible variable declaration)
8	Does potential secret have period? (possible function call)
9	Does the secret have the word "Password"? (possible placeholder or variable)
10	Does potential secret have white spaces? (possible sentence/loop)
11	Does potential secret have ‘-----BEGIN PRIVATE KEY-----’?
12	Does secret begin with a ‘$’ sign? (possible variable)
13	Does secret contain null/nill/undefined/None/true/false?
14	Does potential secret have an arrow (->, =>, ::)? (possible pointer variable)
15	Is a potential secret a numerical value?
16	Is ‘config’ in the secret's file name (including path)?
17	Is ‘test’ or ‘example’ in the file name (including path)?

Statistics of Lines of Code and Secret Length

Statistic	Lines of Code	Secret Length
Minimum	1	2
Maximum	602,221	9,475
Mean (Average)	7,264.19	88.3
Standard Dev.	23,741.28	342.84
25% (1st Quartile)	583	25
50% (Median)	2,333.5	40
75% (3rd Quartile)	4,281.00	64

- Overall, The large standard deviations indicate considerable variability and some secrets are unusually long.

1 Secret Detection: Models for Secret Detection




```
"private_key_id": "test-private-key",
"project_id": "test-project",
"token_uri": "https://oauth2.googleapis.com/token",
"type": "service_account",
}

config = BigQueryV2Config.parse_obj(
```

- All models show high FPR, with the VC model having the most severe level of misclassify non-secrets as secrets. In Python, despite having more true secrets all models show high false positive rates, with the VC model having the most severe level of misclassification.

- Language structure influences performance more than secret prevalence

1 Secret Detection: Generalisation to Unseen Language (C#)

How non-generative models transfer knowledge to identify secrets in an unseen programming language that the models have not previously encountered

CodeBERT falls 21% (0.86 → 0.65)
RF drops 8% (0.75 → 0.67)

C# vs. Java:

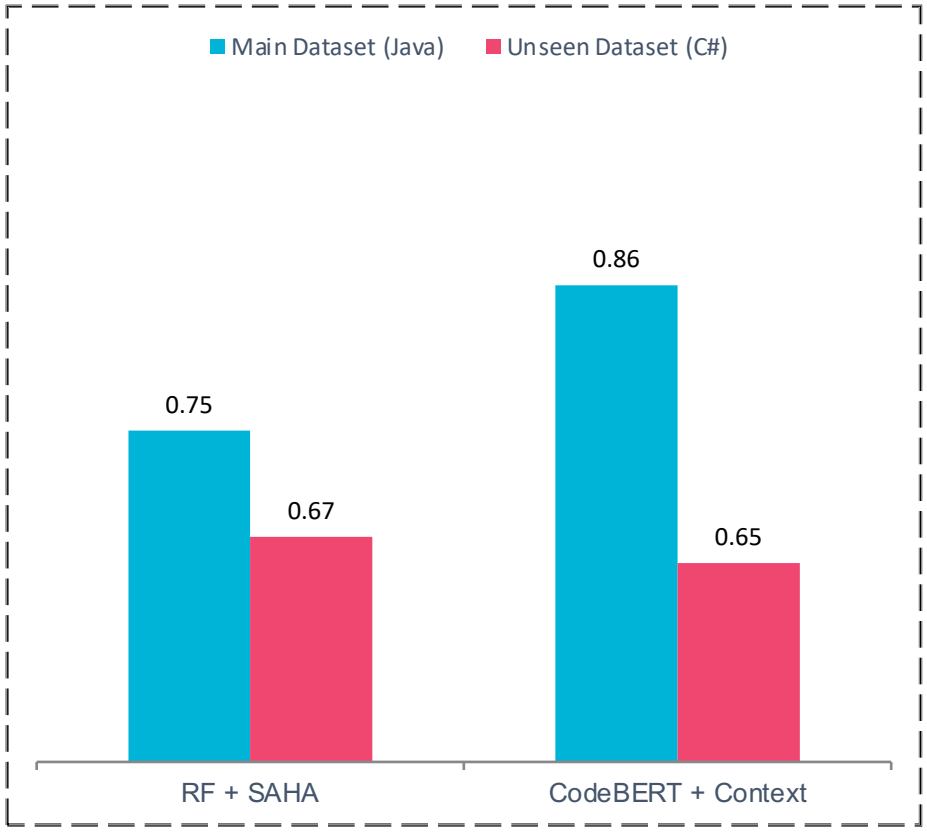
- C# is statically typed and object-oriented, whereas Python, Ruby, and JavaScript are dynamically typed and scripting-oriented.

RF is Robust:

- Entropy & length-like features are language-agnostic; less reliant on syntax patterns.

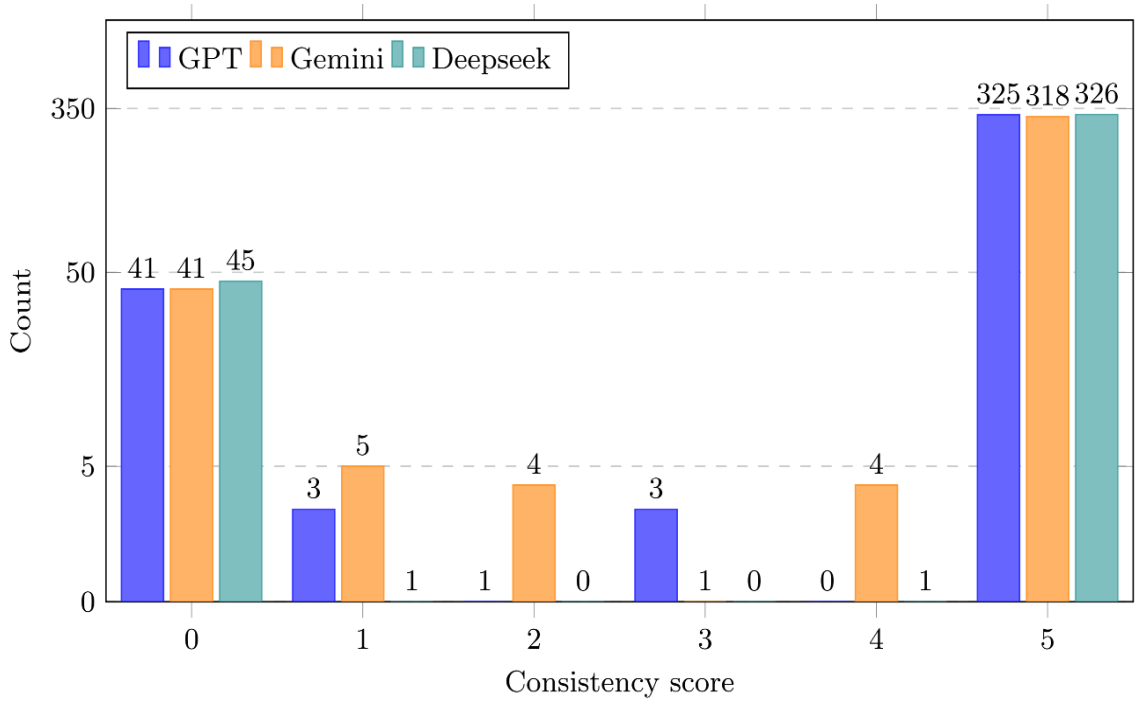
Best Model MCC by Secret Type:

- CodeBERT — Best for Token/Key Types (i.e., API key & secret)
- RF — Best for Structured Secrets (i.e., Database and server URL)



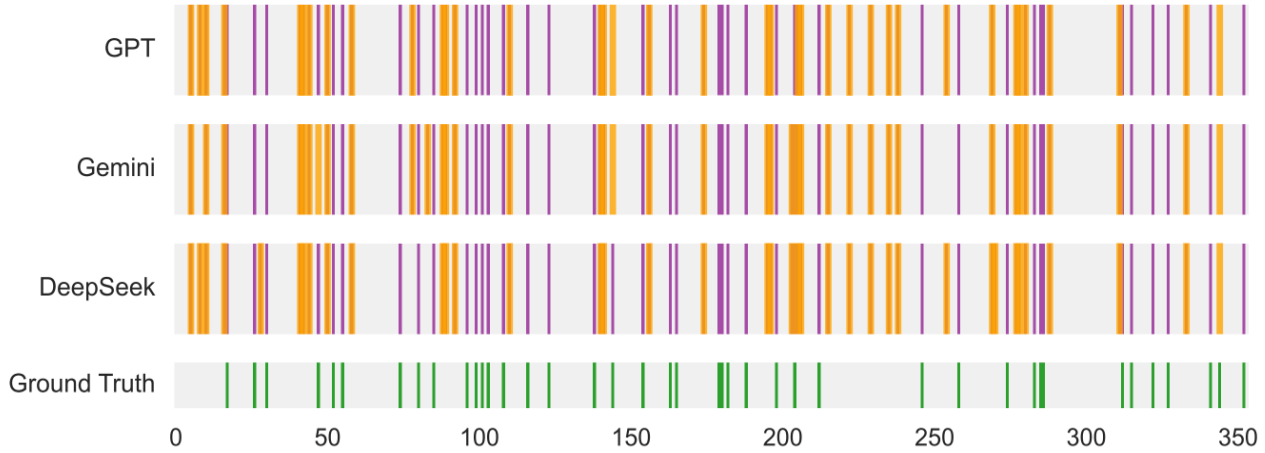
1 Secret Detection: Determinism Analysis of Generative Models

Consistency Score Distribution (0–5) per Model
0 (no correct predictions) to 5 (no incorrect predictions)



Evaluation of each model **five times** on a statistically significant sample size from the main dataset. The statistical tests showed no significant difference across the runs.

Each cell corresponds to one record. If it is colored, it means the model predicted a hardcoded secret; otherwise, it is gray.



Generative AI models are often consistent in secret prediction. They show comparable performance and make similar mistakes.

1 Secret Detection: Models Comparison

	Non-Generative Models	Generative Models
Best MCC	CodeBERT: 0.88 (context)	GPT: 0.63 (base) → 0.71 (CoT)
Generalization	RF: 0.67 / CodeBERT: 0.65	Unseen
Training cost	~5h/fold (CodeBERT, M3 Mac)	No training needed
Inference cost	Free after training	GPT ~$60 · DeepSeek ~$30
Privacy	Local	API call — code sent to vendor
Data needed	Requires labelled dataset	Zero-shot — works out of the box

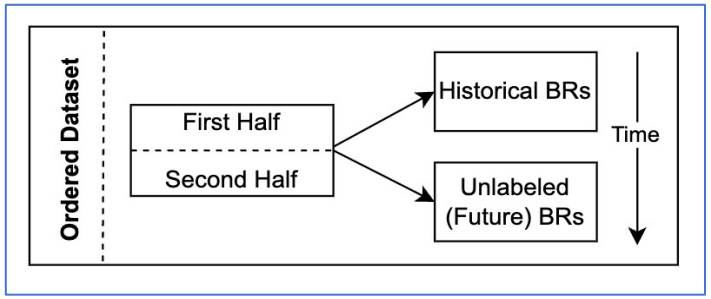
Conclusion & Future Direction

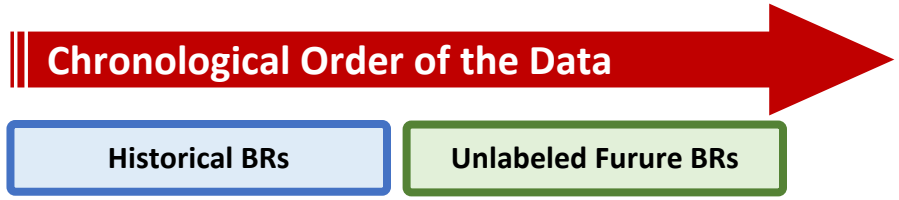
- When sufficient labelled data is available, fine-tuned non-generative models outperform LLMs
- When data is scarce, GPT with prompt engineering provides a competitive alternative.
- A hybrid pipeline — CodeBERT for initial detection + LLM for reasoning on uncertain cases — is the most promising future direction.
- Human oversight remains necessary

Security Bug Report Prediction: Datasets

Project	Description	Total BRs	SBRs	SBR %
Chromium	Web browser (Chrome)	**41,940**	808	1.92%
Derby	Relational DBMS	1,000	179	**17.9%**
Camel	Rule-based routing engine	1,000	74	7.4%
Ambari	Hadoop management UI	1,000	56	5.6%
Wicket	Java web app framework	1,000	47	4.7%

Class imbalance is a fundamental challenge across all five datasets





Assumption: The training (past) data and present (test) data have the same features and distribution.
In real-world: This assumption rarely holds. Accordingly, we design our experimental setup to capture the temporal mismatch.

Security Bug Report Prediction: Datasets

> *FTP endpoints should support charset property like the File endpoint does. Currently you have to remember to set the CamelCharsetName Exchange property if you want to have the message body transformed to another charset. I'll work on a patch and attach it later.*

Apache Camel JIRA — Ground-truth label: Non-Security (Feature Enhancement)

Why It's NOT Security

- No mention of Unauthorized access
- No mention of Authentication / authorization
- No mention of Encryption
- No mention of Confidentiality
- No mention of Integrity violations
- No mention of Vulnerabilities or exploits

This is a feature enhancement for API consistency & encoding — not a security fix.

Misleading Terms (Why the LLM Flagged It)

Term	Why It May Mislead the Model
FTP	Network protocol often discussed alongside plaintext-credential vulnerabilities
charset	Character encoding sometimes tied to injection / encoding-based attacks
"transform … charset"	"Transform" / "encoding" evoke encoding-attack patterns (e.g., XSS bypass)
Exchange property	Technical config terms can resemble security-configuration reports

Token-level correlation, not reasoning about intent