Answers to Reviewer Questions

A. Answer to R1Q2: Limited Causal Analysis: The causal analysis is restricted to the top 100 tokens, possibly overlooking important relationships in larger codebases or complex projects.

Motivation and Approach. Our selection of the top 100 tokens is based on the characteristics of causality analysis, which are time-consuming and require high hardware requirements. Besides, we consider top-100 tokens the primary representative information across releases. If we extend the token number, the potential overfitting would increase. To eliminate concerns about the impact of limited causal analysis on experimental results, we selected the top 200 tokens in the Causal Analysis to conduct a controlled experiment.

Results. Table I shows the comparison results between the top 100 and top 200 settings. As can be seen in Table I, in two cases under IFA and two cases under Effort@20%, performance with the top 100 is better than that of the top 200; on the contrary, in five cases under IFA, performance with the top 200 is better than top 100. In the remaining 11 cases, their performances are the same. We conclude that the performance of the top 100 and top 200 are almost the same.

TABLE I: Comparison between our method SOUND under 100 and 200 tokens. Gray background indicates better.

		Top 100		Top 200	
		Mean	Median	Mean	Median
IFA	Barinel	13	0	14	0
	Dstar	60	8	52	8
	Ochiai	45	2	44	2
	Op2	69	11	70	8
	Tarantula	21	3	18	1
Effort@20%	Barinel	0.039	0.007	0.039	0.007
	Dstar	0.072	0.044	0.072	0.044
	Ochiai	0.054	0.028	0.054	0.028
	Op2	0.077	0.047	0.077	0.048
	Tarantula	0.052	0.021	0.052	0.022

Answer to **R1Q2:** Extending the number of tokens in the causal analysis does not obviously affect the performance of our methods.

B. Answer to R2Q3. A real Example with the ranking list of the code lines in a file

Motivation and Approach. To demonstrate the rationale of our method in a real scenario, we present a detailed example along with the ranking list of the code lines in a file that is easy to check.

We select the file NetworkBridgeFactory.java in the project amq-5.3.0, which contains fewer than 100 lines, making it easy for manual checks. We construct the prediction model using the historical data from the previous release amq-5.2.0.

Results. We present the results of ranking all lines in NetworkBridgeFactory.java in Figure 1.

For our method with Barinel, based on the historical data information in amq-5.2.0, the suspiciousness score of `remoteTransport` is approximately 0.03. As a result, lines 59 and 61 that contain `remoteTransport` are ranked with higher priority and are buggy lines.

For GLANCE-LR, the score is evaluated based on NT (the number of tokens) and NFC (the number of function calls). In this example, line 41, which ranked first, is with $NT = 6$ and $NFC = 1$. So the line score is $NT * (NFC + 1) = 12$. Moreover, the lines containing Control elements are given higher priority, like `return`, `if`, and `else`.

From the IFA view, our method achieves 1, and GLANCE achieves 2. For the Recall@20% view, since the total number of lines is 20 here, we check the performance within top 4 ($20 \times 20\%$) lines. Our method identified two buggy lines, while GLANCE identified one.

By comparing the methods' results, we can see that our method based on historical data performs better.

Answer to **R2Q3:** This real example shows that our method based on the historical data performs better than GLANCE.

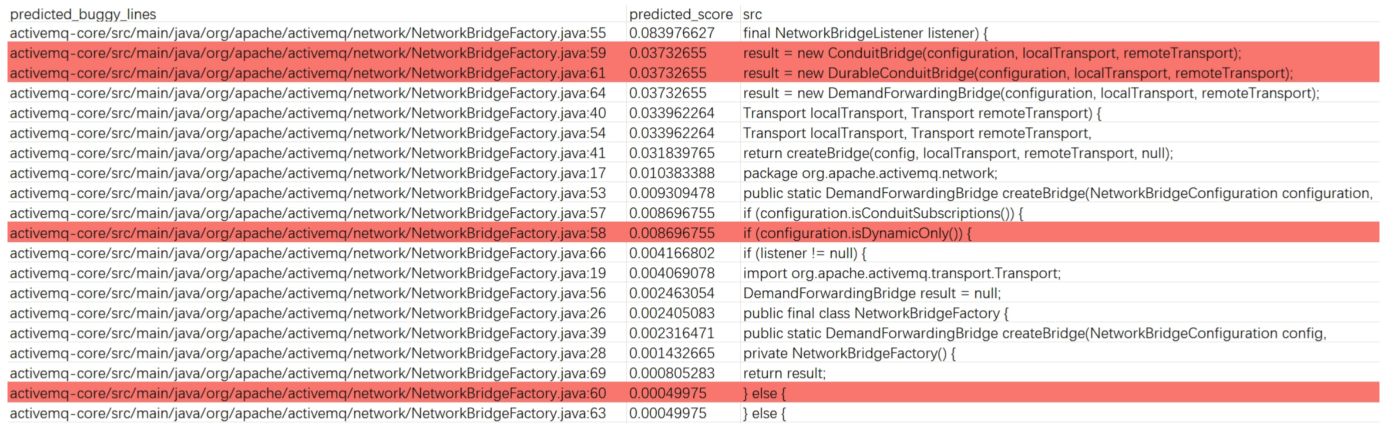
C. Answer to R2Q4: IFA improvement

Motivation and Approach. We extend the results listed in Table VI is RQ4 with calculating the increasing percent after adding Causality Analysis on Non-Effective projects (i.e., the bottom 25% projects with worse IFAs).

Results. Table II shows the results of the increasing percent after adding Causality Analysis on Non-Effective projects. As can be seen in Table II, IFA performance essentially increases by adding Causal Analysis on Non-Effective projects, e.g., median values of Tarantula is from 178 to 17, i.e., increase 90.4% ((178-17)/178).

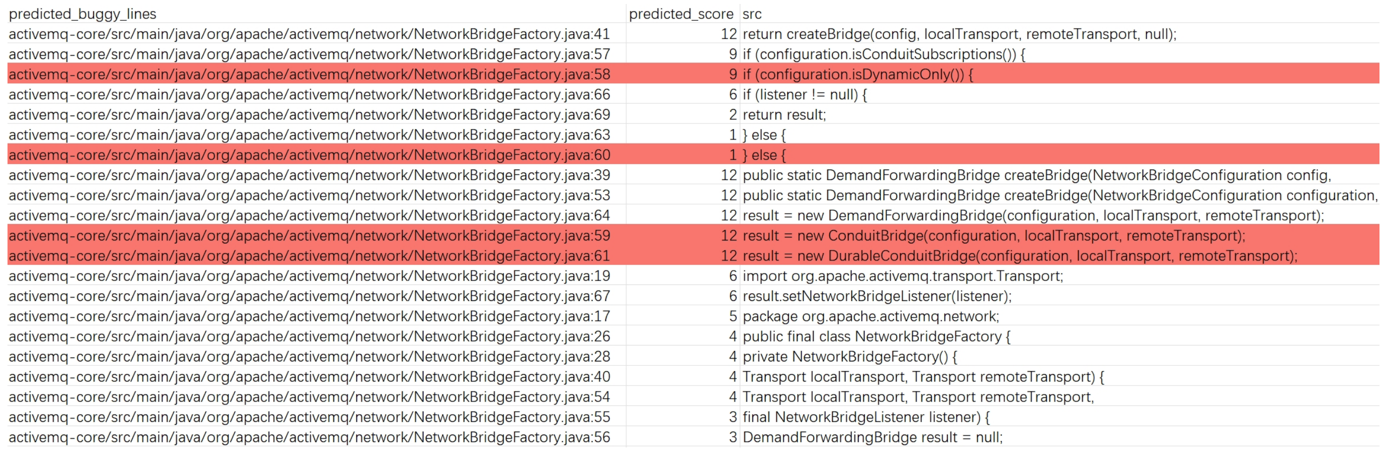
TABLE II: the increasing percent of IFA On Non-effective set after adding Causality Analysis (CA).

	Formula	Mean	Median
Non-Effective	Barinel	1.7%	30.8%
	Dstar	10.3%	27.1%
	Ochiai	17.0%	67.2%
	Op2	18.8%	49.7%
	Tarantula	-5.1%	90.4%



predicted_buggy_lines	predicted_score	src
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:55	0.083976627	final NetworkBridgeListener listener) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:59	0.03732655	result = new ConduitBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:61	0.03732655	result = new DurableConduitBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:64	0.03732655	result = new DemandForwardingBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:40	0.033962264	Transport localTransport, Transport remoteTransport) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:54	0.033962264	Transport localTransport, Transport remoteTransport,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:41	0.031839765	return createBridge(config, localTransport, remoteTransport, null);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:17	0.010383388	package org.apache.activemq.network;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:53	0.009309478	public static DemandForwardingBridge createBridge(NetworkBridgeConfiguration configuration,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:57	0.008696755	if (configuration.isConduitSubscriptions()) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:58	0.008696755	if (configuration.isDynamicOnly()) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:66	0.004166802	if (listener != null) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:19	0.004069078	import org.apache.activemq.transport.Transport;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:56	0.002463054	DemandForwardingBridge result = null;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:26	0.002405083	public final class NetworkBridgeFactory {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:39	0.002316471	public static DemandForwardingBridge createBridge(NetworkBridgeConfiguration config,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:28	0.001432665	private NetworkBridgeFactory() {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:69	0.000805283	return result;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:60	0.00049975	} else {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:63	0.00049975	} else {

(a) The ranking list sorted by our method with Barinel



predicted_buggy_lines	predicted_score	src
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:41	12	return createBridge(config, localTransport, remoteTransport, null);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:57	9	if (configuration.isConduitSubscriptions()) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:58	9	if (configuration.isDynamicOnly()) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:66	6	if (listener != null) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:69	2	return result;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:63	1	} else {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:60	1	} else {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:39	12	public static DemandForwardingBridge createBridge(NetworkBridgeConfiguration config,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:53	12	public static DemandForwardingBridge createBridge(NetworkBridgeConfiguration configuration,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:64	12	result = new DemandForwardingBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:59	12	result = new ConduitBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:61	12	result = new DurableConduitBridge(configuration, localTransport, remoteTransport);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:19	6	import org.apache.activemq.transport.Transport;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:67	6	result.setNetworkBridgeListener(listener);
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:17	5	package org.apache.activemq.network;
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:26	4	public final class NetworkBridgeFactory {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:28	4	private NetworkBridgeFactory() {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:40	4	Transport localTransport, Transport remoteTransport) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:54	4	Transport localTransport, Transport remoteTransport,
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:55	3	final NetworkBridgeListener listener) {
activemq-core/src/main/java/org/apache/activemq/network/NetworkBridgeFactory.java:56	3	DemandForwardingBridge result = null;

(b) The ranking list sorted by GLANCE-LR

Fig. 1: An real example to show the results of ranking all lines in NetworkBridgeFactory.java

Answer to **R2Q4:** IFA performance essentially increases by adding Causal Analysis on Non-Effective projects.

D. Answer to R2Q5: Performance under FAR and D2H.

Motivation and Approach. We conduct experiments under FAR and D2H to check the performance of our method and baselines. As FAR and D2H are classification indicators, we need to set the threshold. Since 20%LOC is used in our paper as a cutoff point for ranking indicators, we continue to employ 20%LOC as the threshold for classification. Our experiment first ranks all lines in a release from the highest to lowest predicted values. Then, we classify the top 20% lines in the ranking list as **defective** and the remaining modules as **clean**.
Results.

FAR and D2H see 2

E. Answer to R3Q1: Why do you think the actual validation approach is realistic?

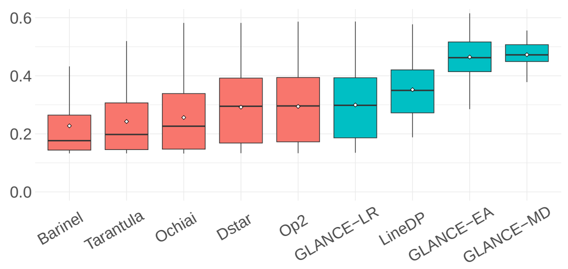
Motivation and Approach.

We use the single release before the predicted as the training set since there may be concept drift between too-old previous releases and the predicted release, i.e., the distribution of buggy lines in the old releases deviates from the prediction release. Therefore, introducing too old releases may not necessarily enhance prediction performance. In other words, using the single release before the predicted would reduce the risk of concept drift.

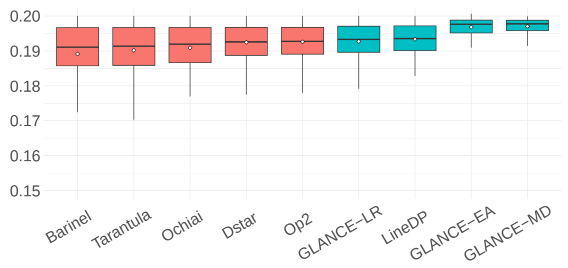
We conduct the experiments using all previous releases, i.e., for release i in the studied projects, we employ all previous releases, i.e., releases $1, \ldots, i-1$, as the training set to construct the prediction models. We conduct two comparisons: (a) we compare our method with the best baseline GLANCE[1] under the "using all previous releases" setting, (b) we compare our method with default setting (i.e., "the single release before") and the "using all previous releases" setting.

Results. Figure 4 shows the comparison result between our method and GLANCE, where "X_newset" indicates Method X under the new setting, i.e., "using all previous releases".

[1]GLANCE-LR is with the best IFA values among all baselines.



(a) D2H (↓) boxplots in the increasing order



(b) FAR (↓) boxplots in the increasing order

Fig. 2: The Boxplots of our methods marked red and baseline methods marked green, where ↑/↓ indicates that a higher/lower value of the metric is better.

In Figure 4, the white diamond points represent the mean value. As can be seen in Figure 4, our methods still perform better than GLANCE: for IFA, top-3 are our methods; for Recall@20%, top-3 are our methods; and for Effort@20%, top-5 are all our methods.

Figure 5 shows the comparison result between our method with default setting (i.e., "using the single release before") and the "using all previous releases" setting. In Figure 5, the white diamond points represent the mean value. As can be seen in Figure 5, the performance under the default setting is slightly better than new settings: for IFA, the default setting is with the better mean values and the same median values; for Recall@20%, the default setting is with the better mean and median values; for Effort@20%, the default setting is with the worse mean values and the better median values.

Answer to **R3Q1:** . The performance under the default setting (i.e., "using the single release before") is slightly better than new settings ("using all previous releases").

F. Answer to R3Q2: Why have you not performed any statistical tests?

Motivation and Approach. We follow the statistical test applied in GLANCE [11], i.e., employ the Wilcoxon-signed Test and Cliff's Delta to compare our method with the baseline methods under all studied projects (similar to Table 7 in [11]). We also calculate the W/T/L numbers for comparison. If BH-corrected p values of the Wilcoxon-signed Test are smaller than 0.05 and the Cliff's Delta values are not negligibly better ($\delta > 0.147$), we mark our methods as Win (W). Note that for IFA and Effort@20%, **smaller** values mean better, i.e., more **negative** Cliff's Delta means our method performs much better than baselines. The definitions of Tie (T) and Loss (L) are similar.

Results. Table IV presents the results of the Wilcoxon-signed Test and Cliff's Delta. The gray background indicates that (a) BH-corrected p values of the Wilcoxon-signed Test are smaller than 0.05, and (b) the Cliff's Delta values are not negligible. Table III summarizes the W/T/L results of our methods with five formulas compared with all baselines.

As can be seen from Tables IV and Table III, our methods with five formulas significantly overcome baselines in most cases, e.g., our methods with Barinel, Ochiai, and Tarantula perform significantly better than all baselines under three indicators.

Answer to **R3Q2:** . According to the result of statistical tests, our methods with five formulas significantly overcome baselines in most cases.

G. Answer to R3Q3: Why have you not included classification and other effort-aware metrics?

Motivation and Approach. As 20% is a widely used inspection threshold, we choose Recall@20% as one of our indicators.

We conducted additional experiments under AUC, Recall@10%, and Recall@30%.

Results. see 3

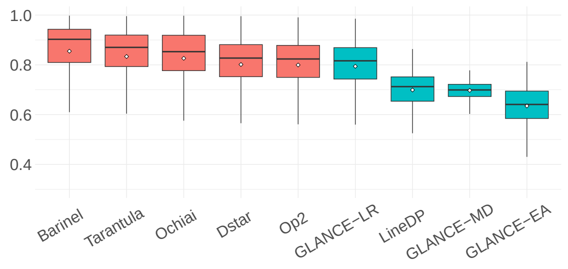
Answer to **R3Q3:** .

TABLE III: The test results from the statistical analysis between SOUND and the baselines under the cross-release scenario

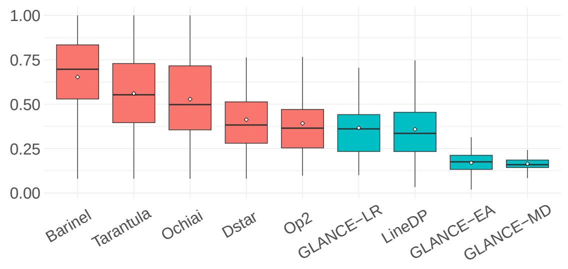
			GLANCE-LR	GLANCE-EA	GLANCE-MD	LineDP	DeepLineDP	N-gram	ErrorProne
IFA (↓)	p-value	Barinel	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Dstar	0.314	0.009	0.057	0.000	0.000	0.279	0.000
		Ochiai	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Op2	0.019	0.301	0.401	0.026	0.000	0.766	0.000
		Tarantula	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Cliff-delta	Barinel	-0.588(L)	-0.691(L)	-0.679(L)	-0.709(L)	-0.853(L)	-0.759(L)	-0.846(L)
		Dstar	0.045(N)	-0.189(S)	-0.196(S)	-0.213(S)	-0.559(L)	-0.181(S)	-0.575(L)
		Ochiai	-0.354(M)	-0.513(L)	-0.502(L)	-0.515(L)	-0.753(L)	-0.551(L)	-0.752(L)
		Op2	0.114(N)	-0.113(N)	-0.131(N)	-0.152(S)	-0.474(L)	-0.108(N)	-0.502(L)
		Tarantula	-0.446(M)	-0.596(L)	-0.585(L)	-0.599(L)	-0.830(L)	-0.650(L)	-0.827(L)
Recall@20% (↑)	p-value	Barinel	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Dstar	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Ochiai	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Op2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Tarantula	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Cliff-delta	Barinel	0.408(M)	0.799(L)	0.815(L)	0.647(L)	0.942(L)	0.932(L)	0.944(L)
		Dstar	0.081(N)	0.697(L)	0.748(L)	0.314(S)	0.938(L)	0.927(L)	0.936(L)
		Ochiai	0.263(S)	0.772(L)	0.798(L)	0.510(L)	0.940(L)	0.929(L)	0.941(L)
		Op2	0.072(N)	0.696(L)	0.752(L)	0.297(S)	0.941(L)	0.930(L)	0.939(L)
		Tarantula	0.326(S)	0.792(L)	0.813(L)	0.571(L)	0.942(L)	0.932(L)	0.943(L)
Effort@20% (↓)	p-value	Barinel	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Dstar	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Ochiai	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Op2	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		Tarantula	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Cliff-delta	Barinel	-0.809(L)	-0.907(L)	-0.921(L)	-0.797(L)	-0.949(L)	-0.948(L)	-0.947(L)
		Dstar	-0.307(S)	-0.847(L)	-0.873(L)	-0.255(S)	-0.946(L)	-0.941(L)	-0.942(L)
		Ochiai	-0.638(L)	-0.896(L)	-0.897(L)	-0.588(L)	-0.947(L)	-0.945(L)	-0.945(L)
		Op2	-0.221(S)	-0.802(L)	-0.849(L)	-0.158(S)	-0.946(L)	-0.941(L)	-0.941(L)
		Tarantula	-0.707(L)	-0.908(L)	-0.912(L)	-0.662(L)	-0.949(L)	-0.948(L)	-0.947(L)

TABLE IV: W/T/L

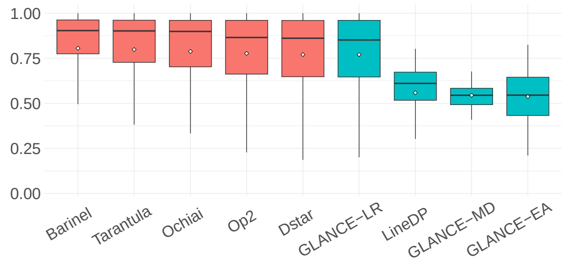
	IFA	Recall@20%LOC	Effort@20%Recall
Barinel	7/0/0	7/0/0	7/0/0
Dstar	4/3/0	6/1/0	7/0/0
Ochiai	7/0/0	7/0/0	7/0/0
Op2	3/4/0	6/1/0	7/0/0
Tarantula	7/0/0	7/0/0	7/0/0



(a) AUC (↑) boxplots in the decreasing order

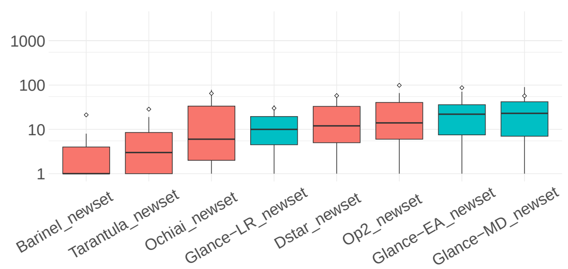


(b) Recall@10% (↑) boxplots in the increasing order

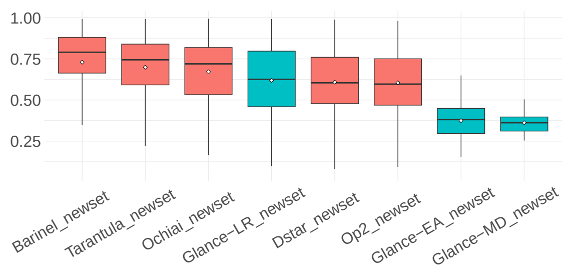


(c) Recall@30% (↑) boxplots in the increasing order

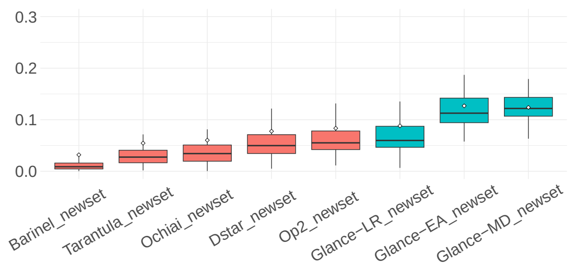
Fig. 3: The Boxplots of our methods marked red and baseline methods marked green, where ↑/↓ indicates that a higher/lower value of the metric is better.



(a) IFA (↓) boxplots in the increasing order

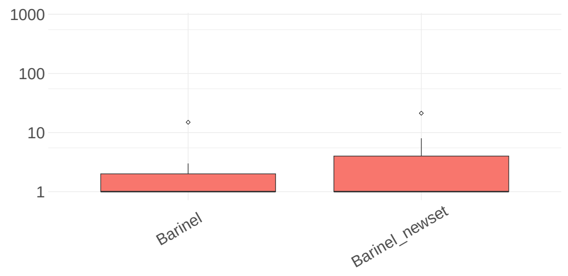


(b) Recall@20%LOC (↑) boxplots in the decreasing order

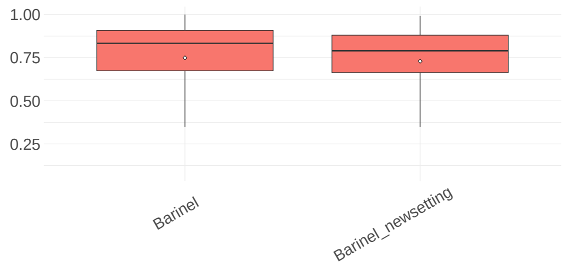


(c) Effort@20%Recall(↓) boxplots in the increasing order

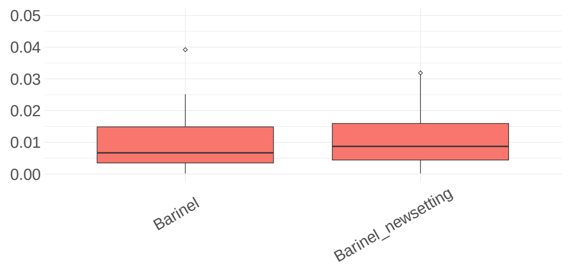
Fig. 4: The Boxplots of our methods marked red and baseline methods marked green, where ↑/↓ indicates that a higher/lower value of the metric is better.



(a) IFA (↓) boxplots in the increasing order



(b) Recall@20%LOC (↑) boxplots in the decreasing order



(c) Effort@20%Recall(↓) boxplots in the increasing order

Fig. 5: The Boxplots of our methods marked red and baseline methods marked green, where ↑/↓ indicates that a higher/lower value of the metric is better.