

AFGNN: API Misuse Detection using Graph Neural Networks and Clustering

Paper #274

1

2 A API Labelling Rules

- 3 **Rule 1:** If the API signature is different (unrelated data types) they
4 should be clustered to different clusters.
- 5 **Rule 2:** If the API signature is different (related data types by inheri-
6 tance/polymorphism) they should be clustered to different
7 clusters.
- 8 **Rule 3:** API parameters sourced from local variables vs. global vari-
9 ables vs. method parameters result in different clusters.
- 10 **Rule 4:** API usage enclosed within if-else/try-catch blocks leads to
11 different clusters.
- 12 **Rule 5:** Variations in how the API call result is used should lead to
13 different clusters:
- 14 **Rule 5:1:** If the result of an API call is assigned, initialized, or ap-
15 pended to a variable (e.g., via `append()` or `add()`), it
16 should be clustered separately.
- 17 **Rule 5:2:** API call in return statement is clustered into different
18 clusters.
- 19 **Rule 6:** Multiple calls to the API within loops or conditional state-
20 ments should be clustered differently.
- 21 **Rule 7:** Multiple sequential API calls (i.e., across a series of state-
22 ments) should be placed in separate clusters.
- 23 **Rule 8:** If the same parameters, variables, or objects are used in API
24 calls with modifiers like `final`, they should be clustered
25 separately.
- 26 **Rule 9:** If the API call appears inside an assert condition predicate,
27 it should be assigned to a distinct cluster.
- 28 **Rule 10:** If the API call is used within conditional branches (`if`,
29 `switch`, `for`, `while`) either in the body or as part of the
30 predicate:
- 31 **Rule 10:1:** API calls within nested conditionals or loops should form
32 separate clusters.
- 33 **Rule 10:2:** If an API call appears in a loop or condition predicate
34 (even if nested inside another control structure), it should
35 be clustered separately from the case where it appears in
36 the body.
- 37 **Rule 11:** API calls placed inside inline functions should belong to
38 different clusters.
- 39 **Rule 12:** API calls in ternary (`? :`) expressions should be clustered
40 separately.
- 41 **Rule 13:** The result of an API call is being used in an assert condition,
42 it should be clustered separately.
- 43 **Rule 14:** The result of an API call getting typecast in the future state-
44 ments is clustered into different clusters.

45 **Rule 15: Rule*:** If any of the above rules are satisfied for a given pair
46 of examples e_1 and e_2 , then e_1 and e_2 should belong to two
47 different clusters.

B API Labelling Rules with Examples

48 **B.1 Rule 1: If the API signature is different (unrelated
49 data types) they should be clustered to different
50 clusters.**

51 **Example 1:**

```
52 public class func {  
53     public void testSubmitNullCallable() {  
54         e.submit((Callable) null);  
55     }  
56 }
```

57 **Example 2:**

```
58 public class func {  
59     public void testExecuteNullRunnable() {  
60         e.submit((Runnable) null);  
61     }  
62 }
```

63 **B.2 Rule 2: If the API signature is different (related
64 data types by inheritance/polymorphism) they
65 should be clustered to different clusters.**

66 **Example 1:**

```
67 public class func{  
68     public void shutdown_waitingOver_abruptShutdown() {  
69         final List<Integer> pool = new ArrayList<Integer>();  
70         backgroundExecutor.submit(new SleepingRunnable(1,  
71             1000, pool));  
72         backgroundExecutor.submit(new SleepingRunnable(2,  
73             1000, pool));  
74         backgroundExecutor.submit(new SleepingRunnable(3,  
75             1000, pool));  
76         backgroundExecutor.submit(new SleepingRunnable(4,  
77             1000, pool));  
78         backgroundExecutor.submit(new SleepingRunnable(5,  
79             5000, pool));  
80         ExecutorServices.shutdown(backgroundExecutor,  
81             100, TimeUnit.MILLISECONDS);  
82     }  
83 }
```

84 **Example 2:**

```
85 public class func{  
86     public void  
87         testNewCachedThreadPoolWithThreadFactory() {  
88             executorService.submit(new NoopRunnable());  
89             executorService.submit(new NoopRunnable());  
90         }  
91     }
```

<p>B.3 Rule 3: API parameters sourced from local variables vs. global variables vs. method parameters result in different clusters.</p> <p>Example 1:</p> <pre>public class func{ public void doRepeater(Socket sock, String host, int port) { DataInputStream is = new DataInputStream(sock.getInputStream()); String line = is.readLine(); } }</pre> <p>Example 2:</p> <pre>public class func{ public void AckProcessor(Map<String, PeerHandler> quorumMap, ClusterConfiguration cnf) { Executors.newSingleThreadExecutor(DaemonThreadFactory.FACTORY); ft = es.submit(this); es.shutdown(); } }</pre> <p>Example 3:</p> <pre>public class func{ public void submit(ExecutorService execService, Callable<T> proc, int numTasks, String label) { execService.submit(proc); } }</pre> <p>B.4 Rule 4: API usage enclosed within if-else / try-catch blocks leads to different clusters.</p> <p>Example 1:</p> <pre>public class func{ public void add(final PatchSetApproval ca) { approvals.add(ca); final Timestamp g = ca.getGranted(); if (g != null && g.compareTo(sortOrder) < 0) { sortOrder = g; } if (ca.getValue() != 0) { hasNonZero = 1; } } }</pre> <p>B.5 Rule 5: Variations in how the API call result is used should lead to different clusters:</p> <p>B.5.1 Rule 5.1: If the result of an API call is assigned, initialized, or appended to a variable (e.g., via append() or add()), it should be clustered separately.</p> <p>Example 1:</p> <pre>public class func{ public void readHeader(DataInputStream in) { String buf = in.readLine(); if (buf == null) { throw new IOException("Unexpected_EOF_reading_magic_ token"); } if (buf.charAt(0) == '#' && buf.charAt(1) == '?') { valid = VALIDPROGRAMTYPE; programType = buf.substring(2); buf = in.readLine(); if (buf == null) throw new IOException("Unexpected EOF reading line after magic token"); } } }</pre> <p>B.5.2 Rule 5.2: API call in return statement is clustered into different clusters.</p> <p>Example 1:</p> <pre>public class func{ public void readLine() { DataInputStream d = new DataInputStream(this); return d.readLine(); } }</pre> <p>B.6 Rule 6: Multiple calls to the API within loops or conditional statements should be clustered differently.</p> <p>Example 1:</p> <pre>public class func{ public void createKafkaMessageConsumer(ConsumerConnector consumerConn, String topic, String topicInHeader, CountDownLatch messagesLatch, Map<String, Integer> topicCountMap) { for (final KafkaStream<byte[], byte[]> stream : consumerMap.get(topic)) { executor.submit(new KafkaTopicConsumer(stream, messagesLatch)); } for (final KafkaStream<byte[], byte[]> stream : consumerMap.get(topicInHeader)) { executor.submit(new KafkaTopicConsumer(stream, messagesLatch)); } } }</pre> <p>Example 2:</p> <pre>public class func{ public void timeSheetChecker(HttpServletRequest request, HttpServletResponse response) { for (int i = 0; i < 5; i++) { Timestamp realTimeDate = UtilDateTime.addDaysToTimestamp(timesheetDate, i); Timestamp nowStartDate = UtilDateTime.getDayStart(now); if ((timesheetDate.compareTo(weekStart) <= 0) && (realTimeDate.compareTo(nowStartDate) < 0)) { List<GenericValue> timeEntryList = timesheetMap.getRelated("TimeEntry", UtilMisc.toMap("partyId", partyId, "timesheetId", timesheetId, "fromDate", realTimeDate), null, false); List<GenericValue> empLeaveList = EntityQuery.use(delegator) .from("EmplLeave").where("partyId", partyId, "fromDate", realTimeDate).cache(true).queryList(); if (UtilValidate.isEmpty(timeEntryList) && UtilValidate.isEmpty(empLeaveList)) { Map<String, Object> noEntryMap = new HashMap<String, Object>(); noEntryMap.put("timesheetId", timesheetId); noTimeEntryList.add(noEntryMap); break; } } } } }</pre> <p>B.7 Rule 7: Multiple sequential API calls (i.e., across a series of statements) should be placed in separate clusters.</p> <p>Example 1:</p> <pre>public class func{ public void if_two_threads_wait_concurrently_then</pre>
--

```

227     _both_of_them_will_read_the_same_output() {
228         Future<Boolean> t1 = executor.submit(new
229             WaitForOutput("Runtime",
230                 process.subscribeToOutput()));
231         Future<Boolean> t2 = executor.submit(new
232             WaitForOutput("Runtime",
233                 process.subscribeToOutput()));
234         process.start();
235     }
236 }
```

B.8 Rule 8: If the same parameters, variables, or objects are used in API calls with modifiers like `final`, they should be clustered separately.

Example 1:

```

241     public class func{ public void compare(final Object
242         obj1,final Object obj2){
243             return ((Timestamp)obj1).compareTo((Timestamp)obj2);
244         }
245     }
```

Example 2:

```

247     public class func {
248         public void runConcurrently(final Callable<Void>...
249             tasks) {
250             for (final Callable<Void> task : tasks) {
251                 futures.add(service.submit(task));
252             }
253             for (final Future<?> future : futures) {
254                 future.get();
255             }
256         }
257     }
```

B.9 Rule 9: If the API call appears inside an assert condition predicate, it should be assigned to a distinct cluster.

Example 1:

```

262     public class func{
263         public void
264             test_readLine_interaction_with_array_read_1(){
265                 BufferedReader r = new BufferedReader(new
266                     StringReader("1\r\n2"));
267                 assertEquals(2, r.read(new char[2], 0, 2));
268                 assertEquals("", r.readLine());
269                 assertEquals("2", r.readLine());
270                 assertNull(r.readLine());
271             }
272     }
```

B.10 Rule 10: If the API call is used within conditional branches (`if`, `switch`, `for`, `while`) either in the body or as part of the predicate:

B.10.1 Rule 10.1: API calls within nested conditionals or loops should form separate clusters.

Example 1:

```

279     public class func{
280         public void getResourceURL(String resource,Class<?>
281             c){
282             if (c != null) {
283                 ClassLoader classLoader = c.getClassLoader();
284                 if (classLoader != null) {
285                     return classLoader.getResource(resource);
286                 }
287             }
288             ClassLoader classLoader =
289                 Thread.currentThread().getContextClassLoader();
290             if (classLoader != null) {
291                 return classLoader.getResource(resource);
292             }
293             return ClassLoader.getSystemResource(resource);
294         }
295     }
```

Example 2:

```

296     public class func{
297         public void getStreamLines(final InputStream is){
298             DataInputStream dis = new DataInputStream(is);
299             if (dis.available() > 0) {
300                 buffer = new StringBuffer(dis.readLine());
301                 while (dis.available() > 0) {
302                     buffer.append('\n').append(dis.readLine());
303                 }
304             }
305             dis.close();
306         }
307     }
```

Example 3:

```

310     public class func{
311         public void HTTPConnectSocket(String host,int
312             port,String proxyHost,int proxyPort){
313             DataInputStream is = new
314                 DataInputStream(getInputStream());
315             String str = is.readLine();
316             if (!str.startsWith("HTTP/1.0\u200e")) {
317                 if (str.startsWith("HTTP/1.0\u200e"))
318                     str = str.substring(9);
319                 throw new IOException("Proxy_reports_\\" + str +
320                     "\\"");
321             }
322             do {
323                 str = is.readLine();
324             }
325             while (str.length() != 0);
326         }
327     }
```

B.10.2 Rule 10.2: If an API call appears in a loop or condition predicate (even if nested inside another control structure), it should be clustered separately from the case where it appears in the body.

Example 1:

```

328     public class func{
329         public void buyPack(int packId,String payForIt){
330             input = new DataInputStream (urlConn.getInputStream
331                 ());
332             while (null != (str = input.readLine())) {
333                 returnStr.append(str);
334             }
335             input.close ();
336         }
337     }
```

Example 2:

```

338     public class func {
339         public void timestampInRange(Timestamp lowerBound,
340             Timestamp upperBound, Timestamp timestamp) {
341             if (timestamp == null) {
342                 if (lowerBound == null) {
343                     return true;
344                 }
345                 else {
346                     return false;
347                 }
348             }
349             if ((lowerBound == null)
350                 (timestamp.compareTo(lowerBound) > 0)) {
351                 if ((upperBound == null)
352                     (timestamp.compareTo(upperBound) < 0)) {
353                     int a = 0;
354                 }
355             }
356         }
357     }
```

B.11 Rule 11: API calls placed inside inline functions should belong to different clusters.

Example 1:

```

367 public class func{
368     public void createAsciidoctor(){
369         asciidoctor = es.submit(new Callable<Asciidoctor>()
370             {
371                 @Override public Asciidoctor call()
372                     throws Exception {
373                         return Asciidoctor.Factory.create();
374                     }});
375             es.shutdown();
376         }
377     }

```

378 **B.12 Rule 12: API calls in ternary (?:) expressions
379 should be clustered separately.**

380 **Example 1:**

```

381 public class func{
382     public void getSourceLocation(Class<?> clazz){
383         final ClassLoader loader = clazz.getClassLoader();
384         final URL resource = loader != null ?
385             loader.getResource(name) :
386             ClassLoader.getSystemResource(name);
387         return resource != null ? resource.toExternalForm()
388             : "<unknown>";
389     }
390 }

```

391 **B.13 Rule 13: The result of an API call is being used
392 in an assert condition, it should be clustered
393 separately.**

394 **Example 1:**

```

395 public class func{
396     public void testMetaInfAccessible(){
397         ClassLoader cl =
398             Thread.currentThread().getContextClassLoader();
399         URL manifestResource =
400             cl.getResource("META-INF/example.txt");
401         assertNotNull(manifestResource);
402     }
403 }

```

404 **B.14 Rule 14: The result of an API call getting
405 typecast in the future statements is clustered into
406 different clusters.**

407 **Example 1:**

```

408 public class func{
409     public void determineFileName(){
410         ClassLoader classLoader = (classloader != null) ?
411             classloader : findClassLoader();
412         URL url =
413             classLoader.getResource(DEFAULT_CATALOG_WEB);
414         if (url != null) {
415             return url.toString();
416         }
417         url = classLoader.getResource(DEFAULT_CATALOG_EJB);
418         return url == null? null: url.toString();
419     }
420 }

```

421 **Example 2:**

```

422 public class func{
423     public void getXmlUrl(String fileName){
424         ClassLoader loader =
425             Thread.currentThread().getContextClassLoader();
426         return loader.getResource(fileName).toExternalForm();
427     }
428 }

```

C Rand Index (RI) and Mutual Information (MI)

The **Rand Index (RI)** measures the proportion of sample pairs that are either assigned to the same cluster in both the predicted and ground truth labels, or assigned to different clusters in both. Mathematically, the Rand Index is:

$$RI = \frac{\text{Number of agreements}}{\text{Total number of pairs}} \quad (1)$$

The **Adjusted Rand Index (ARI)** corrects for chance by considering the expected similarity of all pairwise assignments. It is defined as:

$$ARI(U, V) = \frac{RI - \mathbb{E}[RI]}{1 - \mathbb{E}[RI]} \quad (2)$$

where $\mathbb{E}[RI]$ is the expected Rand Index under random clustering. ARI ranges from -1 (poor agreement) to 1 (perfect match), with 0 indicating random labeling.

The **Mutual Information (MI)** measures the amount of shared information between two clusterings. It is given by:

$$MI(U, V) = \sum_{u \in U} \sum_{v \in V} P(u, v) \log \left(\frac{P(u, v)}{P(u)P(v)} \right) \quad (3)$$

where: $P(u)$ = Probability of cluster u in U (the true labels), $P(v)$ = Probability of cluster v in V (the predicted labels), and $P(u, v)$ = Joint probability of a sample being in cluster u in U and cluster v in V .

The **Adjusted Mutual Information (AMI)** further normalizes this value against chance, and it is defined as:

$$AMI(U, V) = \frac{MI(U, V) - E(MI(U, V))}{\frac{1}{2} [H(U) + H(V)] - E(MI(U, V))} \quad (4)$$

AMI ranges from 0 (no mutual information beyond chance) to 1 (perfect correlation). The above clustering evaluation metrics are adapted from the definitions and formulations provided by the scikit-learn documentation.

D External index results for API usage clustering. The upward arrow indicates that higher RI and MI scores are better

Table 1 summarizes the clustering results for the Java APIs for four different models (AFGCN, AFRGCN and the two baseline models). For each model, the table shows the Rand Index (RI) and Mutual Information (MI) scores, computed using the predicted clusters and the ground-truth labels. In each row, the highest RI and MI scores are indicated in boldface. The suffix “Birch1.5” in first row refers to the threshold parameter (set to 1.5) used in the BIRCH clustering algorithm, which controls the maximum diameter of subclusters during clustering.

E Performance of AFGNN with and without sequence edges

Table 2 summarizes the AFGNN results when SE edges are considered versus when they are not. In the case of AFRGCN, adding SE edges improves the effectiveness of AFRGCN by 48% and 40% for RI and MI scores respectively. For AFGCN also there is meaningful performance improvement.

API Name	UnixCoder-Birch(1.5)		GraphCodeBERT-Birch(1.5)		AFGCN-Birch(1.5)		AFRGCN-Birch(2.1)	
	126M		124M		331K		1.3M	
	RI score ↑	MI score ↑	RI score ↑	MI score ↑	RI score ↑	MI score ↑	RI score ↑	MI score ↑
ClassLoader.getResource()	-0.004	-0.005	0.401	0.454	0.254	0.23	0.322	0.396
Thread.start()	-0.003	-0.003	0.158	0.176	0.114	0.231	0.439	0.472
Statement.execute()	0.219	0.235	0.237	0.33	0.34	0.422	0.174	0.184
BufferedReader.read()	0.027	0.055	0.299	0.366	0.144	0.238	0.412	0.492
Timestamp.compareTo()	0.497	0.497	0.239	0.239	0.39	0.489	0.325	0.325
DataInputStream.readLine()	0.163	0.182	0.4	0.416	0.383	0.423	0.41	0.465
ServerSocket.bind()	0.114	0.141	0.288	0.34	0.098	0.198	0.231	0.298
ExecutorService.submit()	0.665	0.665	0.192	0.205	0.021	0.056	0.278	0.306
URI.getFragment()	0.302	0.302	0.449	0.538	0.433	0.449	0.683	0.719
Calendar.getTime()	0.219	0.235	0.304	0.423	0.064	0.207	0.313	0.425
Socket.connect()	0.302	0.335	0.608	0.671	0.198	0.412	0.58	0.621
JPanel.add()	0.114	0.157	0.273	0.352	0.055	0.122	0.384	0.49
FileChannel.read()	-0.004	-0.004	0.171	0.182	-0.007	-0.007	0.142	0.157
DateFormat.format()	-0.004	-0.004	0.141	0.189	0.042	0.096	0.364	0.339
ClassLoader.loadClass()	-0.004	-0.004	0.087	0.11	-0.041	-0.072	0.208	0.232
Runtime.freeMemory()	-0.004	-0.008	0.143	0.246	0.007	0.059	0.151	0.175
Graphics2D.fill()	0.139	0.171	0.004	0.02	0.013	0.071	0.004	0.02
DriverManager.getConnection()	0.178	0.189	0.288	0.352	0.284	0.328	0.301	0.335
URL.openConnection()	-0.004	-0.004	0.408	0.476	0.357	0.379	0.506	0.578
File.toURI()	-0.004	-0.004	-0.01	-0.011	-0.02	-0.038	0.171	0.171
BufferedReader.readLine()	0.087	0.116	0.18	0.247	0.216	0.284	0.183	0.304
Average	0.143	0.154	0.25	0.301	0.159	0.218	0.313	0.357

Table 1: External index results for API usage clustering. The upward arrow indicates that higher RI and MI scores are better.

Algorithm 1: Algorithm to find the best clustering for an API

Input: E: Embeddings as $[e_1, e_2, \dots, e_n]$, where e_i is the AFGNN embedding for i th example; M: Birch clustering algorithm

Output: C: Cluster labels as $[c_1, c_2, \dots, c_n]$, where c_j is the predicted cluster label for j th example

Procedure clusterTheEmbeddings (E, M)

```

1   min_db_score ← ∞;
2    $C \leftarrow [0, 0, \dots, 0]$ ;           // Initially, all
3   examples are in the 0th cluster
4   for cluster_cnt ← 2 to n do
5     pred_labels ← M( $E, cluster\_cnt$ );
6     db_score ← calcDBscore( $E, pred\_labels$ );
7     if db_score < min_db_score then
8       min_db_score ← db_score;
9       C ← pred_labels;           // Clustering
                                improved

```

```

1 public class Example{
2   public void initDocumentCache(Book book){
3     Thread documentIndexerThread = new Thread(new
4       DocumentIndexer(book),
5         "DocumentIndexer");
6     documentIndexerThread.
7       setPriority(Thread.MIN_PRIORITY);
8     documentIndexerThread.start();
9   }

```

(a) Thread.start () Example 1

```

1 public class Example{
2   public void BackgroundStreamSaver(InputStream
3     in, OutputStream out){
4     Thread myThread = new Thread(this,
5       getClass().getName());
6     myThread.setPriority(Thread.MIN_PRIORITY);
7   }

```

(b) Thread.start () Example 2

Figure 1: Two semantically similar usages of the Thread.start () API. When SE edges (e.g., between setPriority() and start()) are included, APIFlowGNN clusters these examples together, highlighting the impact of SE edges on capturing meaningful API usage patterns.

G Two semantically similar usages of the Thread.start () API

470 F Algorithm for determining the best clusters

471 The algorithm shown in Algorithm 1 is used to determine the best
472 clustering for a particular API. Initially, all the examples are con-
473 sidered to be in one cluster, and then we iteratively cluster the ex-
474 amples for all possible cluster counts (from 2 to the total number of
475 examples) and calculate the DB score in each iteration. The algo-
476 rithm chooses the final output clustering that has the minimum DB
477 score.

480 Figure 1a and 1b are two similar examples of using
481 Thread.start () API where the thread is created, priority is set
482 to minimum, and finally, the thread is started. AFGNN keeps these
483 two examples in the same cluster when the SE edge between Lines
484 4 and 5 (Thread.setPriority() to Thread.start())
485 is there otherwise, it considers the two usage examples different,
486 illustrating the effect of SE edges.

API Name	Without Sequence Edges				With Sequence Edges			
	AFGCN-Birch(1.5)		AFRGCN-Birch(2.1)		AFGCN-Birch(1.5)		AFRGCN-Birch(2.1)	
	RI score ↑	MI score ↑	RI score ↑	MI score ↑	RI score ↑	MI score ↑	RI score ↑	MI score ↑
ClassLoader.getResource()	0.219	0.206	0.173	0.226	0.254	0.23	0.322	0.396
Thread.start()	0.114	0.231	0.124	0.143	0.114	0.231	0.439	0.472
Statement.execute()	0.065	0.165	0.211	0.268	0.34	0.422	0.174	0.184
BufferedReader.read()	0.056	0.117	0.248	0.326	0.144	0.238	0.412	0.492
Timestamp.compareTo()	0.39	0.489	0.492	0.492	0.39	0.489	0.325	0.325
DataInputStream.readLine()	0.445	0.5	0.289	0.315	0.383	0.423	0.41	0.465
ServerSocket.bind()	0.093	0.214	0.317	0.379	0.098	0.198	0.231	0.298
ExecutorService.submit()	-0.016	-0.032	-0.014	-0.016	0.021	0.056	0.278	0.306
URI.getFragment()	0.192	0.376	0.547	0.613	0.433	0.449	0.683	0.719
Calendar.getTime()	0.023	0.081	0.318	0.356	0.064	0.207	0.313	0.425
Socket.connect()	0.034	0.143	0.465	0.533	0.198	0.412	0.58	0.621
JPanel.add()	0.048	0.14	0.107	0.227	0.055	0.122	0.384	0.49
FileChannel.read()	-0.015	-0.019	-0.014	-0.014	-0.007	-0.007	0.142	0.157
DateFormat.format()	0.031	0.074	0.102	0.119	0.042	0.096	0.364	0.339
ClassLoader.loadClass()	-0.041	-0.072	0.057	0.068	-0.041	-0.072	0.208	0.232
Runtime.freeMemory()	0.009	0.026	0.213	0.282	0.007	0.059	0.151	0.175
Graphics2D.fill()	0.006	0.026	0.188	0.229	0.013	0.071	0.004	0.02
DriverManager.getConnection()	0.433	0.466	0.366	0.389	0.284	0.328	0.301	0.335
URL.openConnection()	0.34	0.404	0.133	0.226	0.357	0.379	0.506	0.578
File.toURI()	-0.022	-0.056	-0.014	-0.015	-0.02	-0.038	0.171	0.171
BufferedReader.readLine()	0.095	0.139	0.118	0.224	0.216	0.284	0.183	0.304
Average	0.119	0.172	0.211	0.256	0.159	0.218	0.313	0.357

Table 2: Performance of AFGNN with and without sequence edges

H P-value Comparison of UnixCoder, GraphCodeBERT, and AFGNN

P-Value evaluation: AFGNN performs well in most API clusterings, as indicated by the RI and MI scores in Table 1. However, in some cases, its performance is close to GraphCodeBERT and UnixCoder, prompting a statistical significance analysis. We conducted a p-value analysis with a null hypothesis of no performance variation among models, using a significance threshold of 0.05. The results are summarized in Table 3.

Model Pairwise Comparisons	P-Value	
	For RI score ↓	For MI score ↓
UnixCoder and AFRGCN	0.0022	0.0008
UnixCoder and AFGCN	0.8938	0.1944
GraphCodeBERT and AFRGCN	0.0104	0.0227
GraphCodeBERT and AFGCN	0.0053	0.0128

Table 3: Comparison of UnixCoder, GraphCodeBERT, and AFGNN