

## **CS 569 PROJECT PLAN REPORT**

**AGORA Revisited: A Replication Study on Innovative Test Oracle Generation for  
Advancing the Robustness of REST API**

**DONE BY**

PARTH SHAH

SANKARANARAYANAN SRINIVASAN

HARSHINI REDDY KOUKUNTLA

TARUNI MOVVA

## **Introduction:**

The rapid expansion of web services and the prevalence of microservices architecture have underscored the importance of Application Programming Interfaces (APIs) in contemporary software systems. Ensuring the reliability and quality of APIs is paramount, necessitating effective testing methodologies. AGORA, a groundbreaking approach introduced in a seminal research paper, promises to revolutionize API testing automation, offering improvements in efficiency and reliability. However, before integration into real-world development workflows, it is imperative to meticulously validate AGORA's claims and assess its performance under diverse conditions. This project proposes a comprehensive replication study aimed at investigating the scalability, efficiency, and effectiveness of AGORA, addressing the research questions in this report.

## **The Research Questions:**

This replication study aims to address the following primary research questions:

- a. How does the performance of AGORA scale with larger datasets of API requests and responses? Are there any bottlenecks in the system as the dataset size increases?
- b. How does detecting and reducing duplicate input requests affect the performance of AGORA's API testing process?
- c. How do different configurations of JSONMutator mutation operators affect the failure detection ratio achieved by AGORA?

Each research question delves into critical aspects of AGORA's functionality and its potential impact on API testing efficiency and reliability.

## **Motivation**

The motivation behind this replication study is deeply rooted in the imperative need to rigorously validate and extend the findings presented in the original research on AGORA's capabilities in API testing automation. Addressing the three primary research questions is paramount due to their direct implications on the efficiency, scalability, and reliability of AGORA:

**Scalability with Larger Datasets:** Understanding how AGORA performs as the dataset size increases is crucial for its practical deployment in real-world scenarios. Scalability issues can hinder its effectiveness, potentially leading to bottlenecks and inefficiencies in testing workflows. By investigating AGORA's performance under varying dataset sizes, we aim to uncover any scalability limitations and identify strategies to address them, ensuring its applicability to diverse and large-scale API testing scenarios.

**Impact of Duplicate Request Reduction:** The presence of duplicate input requests can significantly impact AGORA's performance, potentially skewing performance metrics and hindering test coverage. By examining the effects of detecting and reducing duplicate requests, we aim to elucidate the extent to which AGORA's efficiency and effectiveness can be enhanced. Identifying and mitigating duplicate requests can lead to streamlined testing processes, improved resource utilization, and ultimately, more reliable API testing outcomes.

**Effect of Mutation Operator Configurations:** The selection and configuration of mutation operators play a pivotal role in AGORA's ability to detect failures accurately. By exploring different configurations of JSONMutator mutation operators, we aim to determine their impact on AGORA's failure detection ratio. Understanding which mutation operator configurations yield optimal results can inform developers and testers in fine-tuning AGORA for enhanced fault detection and improving overall testing efficacy.

In essence, this replication study seeks to provide empirical evidence and insights into AGORA's performance across various dimensions crucial for its practical utility in API testing. By addressing these research questions, we aim to validate AGORA's claims, identify potential challenges, and pave the way for its adoption as a reliable and efficient tool in the arsenal of software developers and testers. Ultimately, our endeavors aim to contribute to the advancement of automated API testing methodologies, fostering the development of more robust and resilient software systems.

## **Solution Approach and Exploration of Findings:**

#### RQ1. How does the performance of AGORA scale with larger datasets of API requests and responses?

In our analysis of AGORA's scalability with enhanced datasets, we've developed two primary approaches. To evaluate AGORA's effectiveness with larger datasets, We expanded the OMDB dataset by adding new sources, such as the "Get Series Details by Season" option. We carefully selected these sources to ensure they work well with various data types, so as to check the efficiency of the AGORA working with data of many types. Below is an example of one of the sources we incorporated.

```
 10  paths:
 11    /movie/{id}:
 12      get:
 13        tags:
 14          - Movie
 15        summary: Get Movie by IMDB ID
 16        operationId: GetById
 17        parameters:
 18          - name: id
 19            in: path
 20            required: true
 21            type: string
 22        responses:
 23          200:
 24            description: Successful operation
 25            schema:
 26              $ref: "#/definitions/movieDetail"
 27          401:
 28            description: Not authenticated
 29            schema:
 30              $ref: "#/definitions/Error"
 31          500:
 32            description: Internal Server Error
 33            schema:
 34              $ref: "#/definitions/Error"
 35
 36 +   /series/{id}:
 37     get:
 38       tags:
 39         - Series
 40       summary: Get Series Details by Id
 41       operationId: GetSeriesById
 42       parameters:
 43         - name: id
 44           in: path
 45           required: true
 46           type: string
 47       responses:
 48         200:
 49           description: Successful operation
 50           schema:
 51             $ref: "#/definitions/seriesDetail"
 52         401:
 53           description: Not authenticated
 54           schema:
 55             $ref: "#/definitions/Error"
 56         500:
 57           description: Internal Server Error
 58           schema:
 59             $ref: "#/definitions/Error"
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1198
 1199
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1298
 1299
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1348
 1349
 1350
 1351
 1352
 1353
 1354
 1355
 1356
 1357
 1358
 1359
 1359
 1360
 1361
 1362
 1363
 1364
 1365
 1366
 1367
 1368
 1369
 1369
 1370
 1371
 1372
 1373
 1374
 1375
 1376
 1377
 1378
 1379
 1379
 1380
 1381
 1382
 1383
 1384
 1385
 1386
 1387
 1388
 1388
 1389
 1390
 1391
 1392
 1393
 1394
 1395
 1396
 1397
 1398
 1398
 1399
 1399
 1400
 1401
 1402
 1403
 1404
 1405
 1406
 1407
 1408
 1408
 1409
 1410
 1411
 1412
 1413
 1414
 1415
 1416
 1417
 1418
 1418
 1419
 1420
 1421
 1422
 1423
 1424
 1425
 1426
 1427
 1428
 1429
 1429
 1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457
 1458
 1459
 1459
 1460
 1461
 1462
 1463
 1464
 1465
 1466
 1467
 1468
 1469
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478
 1479
 1479
 1480
 1481
 1482
 1483
 1484
 1485
 1486
 1487
 1488
 1488
 1489
 1490
 1491
 1492
 1493
 1494
 1495
 1496
 1497
 1498
 1498
 1499
 1499
 1500
 1501
 1502
 1503
 1504
 1505
 1506
 1507
 1508
 1508
 1509
 1510
 1511
 1512
 1513
 1514
 1515
 1516
 1517
 1518
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1559
 1560
 1561
 1562
 1563
 1564
 1565
 1566
 1567
 1568
 1569
 1569
 1570
 1571
 1572
 1573
 1574
 1575
 1576
 1577
 1578
 1579
 1579
 1580
 1581
 1582
 1583
 1584
 1585
 1586
 1587
 1588
 1588
 1589
 1590
 1591
 1592
 1593
 1594
 1595
 1596
 1597
 1598
 1598
 1599
 1599
 1600
 1601
 1602
 1603
 1604
 1605
 1606
 1607
 1608
 1608
 1609
 1610
 1611
 1612
 1613
 1614
 1615
 1616
 1617
 1618
 1618
 1619
 1620
 1621
 1622
 1623
 1624
 1625
 1626
 1627
 1628
 1629
 1629
 1630
 1631
 1632
 1633
 1634
 1635
 1636
 1637
 1638
 1639
 1639
 1640
 1641
 1642
 1643
 1644
 1645
 1646
 1647
 1648
 1648
 1649
 1650
 1651
 1652
 1653
 1654
 1655
 1656
 1657
 1658
 1659
 1659
 1660
 1661
 1662
 1663
 1664
 1665
 1666
 1667
 1668
 1669
 1669
 1670
 1671
 1672
 1673
 1674
 1675
 1676
 1677
 1678
 1679
 1679
 1680
 1681
 1682
 1683
 1684
 1685
 1686
 1687
 1688
 1688
 1689
 1690
 1691
 1692
 1693
 1694
 1695
 1696
 1697
 1698
 1698
 1699
 1699
 1700
 1701
 1702
 1703
 1704
 1705
 1706
 1707
 1708
 1708
 1709
 1710
 1711
 1712
 1713
 1714
 1715
 1716
 1717
 1718
 1718
 1719
 1720
 1721
 1722
 1723
 1724
 1725
 1726
 1727
 1728
 1729
 1729
 1730
 1731
 1732
 1733
 1734
 1735
 1736
 1737
 1738
 1739
 1739
 1740
 1741
 1742
 1743
 1744
 1745
 1746
 1747
 1748
 1748
 1749
 1750
 1751
 1752
 1753
 1754
 1755
 1756
 1757
 1758
 1759
 1759
 1760
 1761
 1762
 1763
 1764
 1765
 1766
 1767
 1768
 1769
 1769
 1770
 1771
 1772
 1773
 1774
 1775
 1776
 1777
 1778
 1779
 1779
 1780
 1781
 1782
 1783
 1784
 1785
 1786
 1787
 1788
 1788
 1789
 1790
 1791
 1792
 1793
 1794
 1795
 1796
 1797
 1798
 1798
 1799
 1799
 1800
 1801
 1802
 1803
 1804
 1805
 1806
 1807
 1808
 1808
 1809
 1810
 1811
 1812
 1813
 1814
 1815
 1816
 1817
 1818
 1818
 1819
 1820
 1821
 1822
 1823
 1824
 1825
 1826
 1827
 1828
 1829
 1829
 1830
 1831
 1832
 1833
 1834
 1835
 1836
 1837
 1838
 1839
 1839
 1840
 1841
 1842
 1843
 1844
 1845
 1846
 1847
 1848
 1848
 1849
 1850
 1851
 1852
 1853
 1854
 1855
 1856
 1857
 1858
 1859
 1859
 1860
 1861
 1862
 1863
 1864
 1865
 1866
 1867
 1868
 1869
 1869
 1870
 1871
 1872
 1873
 1874
 1875
 1876
 1877
 1878
 1879
 1879
 1880
 1881
 1882
 1883
 1884
 1885
 1886
 1887
 1888
 1888
 1889
 1890
 1891
 1892
 1893
 1894
 1895
 1896
 1897
 1898
 1898
 1899
 1899
 1900
 1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1908
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1918
 1919
 1920
 1921
 1922
 1923
 1924
 1925
 1926
 1927
 1928
 1929
 1929
 1930
 1931
 1932
 1933
 1934
 1935
 1936
 1937
 1938
 1939
 1939
 1940
 1941
 1942
 1943
 1944
 1945
 1946
 1947
 1948
 1948
 1949
 1950
 1951
 1952
 1953
 1954
 1955
 1956
 1957
 1958
 1959
 1959
 1960
 1961
 1962
 1963
 1964
 1965
 1966
 1967
 1968
 1969
 1969
 1970
 1971
 1972
 1973
 1974
 1975
 1976
 1977
 1978
 1979
 1979
 1980
 1981
 1982
 1983
 1984
 1985
 1986
 1987
 1988
 1988
 1989
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1998
 1999
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2009
 2010
 2011
 2012
 2013
 2014
 2015
 2016
 2017
 2018
 2019
 2019
 2020
 2021
 2022
 2023
 2024
 2025
 2026
 2027
 2028
 2029
 2029
 2030
 2031
 2032
 2033
 2034
 2035
 2036
 2037
 2038
 2039
 2039
 2040
 2041
 2042
 2043
 2044
 2045
 2046
 2047
 2048
 2048
 2049
 2050
 2051
 2052
 2053
 2054
 2055
 2056
 2057
 2058
 2059
 205
```

The API response for these endpoints are generated. Some of the response are manually generated to make sure the API response are inline with the Open API format. The main point of this analysis is to ensure whether modified version of the daikon detects more invariant when compared with the invariant generated with existing dataset.

For our next step, we introduced a new endpoint by integrating the weather API, which we obtained from APIs.guru. We specifically selected this API because its endpoints provide numerical data. This choice was intentional to evaluate AGORA's capability in handling arithmetic comparisons. Additionally, we ensured to use the latest version of the API to guarantee that we are working with the most up-to-date responses.

Here is the OpenAPI spec for the Weather API

Here is the snapshot of the generated endpoint

We utilized a tool called POSTMAN for generating API responses. After generation, we systematically organized these responses into folders categorized by the size of the dataset, ensuring adherence to the existing organizational structure. To create the necessary .decl and .dtrace files for analysis, we employed BEET within the IntelliJ Integrated Development Environment (IDE). For the execution of Daikon, which is instrumental in identifying program invariants, we relied on Docker. This setup facilitated a streamlined process, enabling us to efficiently run Daikon with the generated files and ultimately produce valuable invariants.

Here is the example of the generated invariant for the weather API Response

1. /weather-api&getCurrentWeather&200&data():::EXIT;data.attributes.temp is Numeric;daikon.inv.unary.string.IsNumeric;(data.attributes.temp)
  2. /weather-api&getCurrentWeather&200&data():::EXIT;data.attributes.weather one of {"Mild", "Sweltering", "Scorching", "Frost", "Cool", "Humid", "Rain", "Partly Cloudy", "Freezing", "Extreme Heat", "Muggy", "Chilly", "Breezy", "Clear", "Cloudy", "Wet"};daikon.inv.unary.scalar.OneOfScalar;(data.attributes.weather)
  3. /weather-api&getCurrentWeather&200&data():::EXIT;data.type == "weather";daikon.inv.binary.twoString.StringEqual;(data.type)

Our primary objective was to assess the performance of the AGORA system using enhanced datasets. AGORA employs Dynamic Program Analysis, a method whose effectiveness is directly influenced by both the quality and quantity of the data it processes. Therefore, we are confident that as the size of the dataset expands, AGORA's efficiency—specifically its ability to identify more invariants—will also improve.

To measure the efficiency of AGORA for a larger dataset compared to the authors' existing dataset,

Efficiency = Number of relevant invariants generated from the extended dataset / Total number of invariants generated from the authors' dataset.

## RQ2. Impact of Duplicate Request Reduction on AGORA's Performance

## Observations:

During our preliminary analysis of the AGORA dataset, it became evident that duplicate input requests were prevalent. Specifically, we observed redundancy in the dataset, with certain API endpoints being called repeatedly with identical parameters. For instance, the "createPlaylist" endpoint exhibited duplicate requests with the same 'userid' parameter, while the "getAlbumTrack" endpoint consistently included the 'market' parameter across requests. Such redundancy in input requests can potentially skew performance metrics, leading to inefficiencies in testing efforts and resource utilization.

createPlaylist		getAlbumTracks			
operationId	path	httpMethod	F	G	H
createPlaylist	/users/{user_id}/playlists	POST	market=AD;offset=7;limit=4;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=ID;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IE;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=I;offset=12;limit=38;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IL;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IN;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=I;limit=12;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IN;limit=38;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=I;offset=10;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IT;limit=32;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=IO;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KE;offset=19;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KE;offset=19;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KE;offset=8;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KG;offset=8;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KH;offset=3;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KL;offset=12;limit=47;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KM;	getAlbumTracks	GET
createPlaylist	/users/{user_id}/playlists	POST	market=KN;offset=19;limit=46;	getAlbumTracks	GET

## Plan:

### Experiment 1:

- Set a threshold value to control duplicate REST API requests in industrial REST API datasets.
- Limit duplicate records up to the threshold value.
- Provide the modified dataset CSV and OpenAPI Specification (OAS) to Beet as input.
- Use modified Daikon to produce invariants based on the Beet results.
- Apply JSONMutator tool to generate a test suite for the new dataset with limited duplicates.
- Detect any new invariants generated and evaluate the impact on the test suite.

### Experiment 2:

- Repeat the steps from Experiment 1 with a different OAS specification, addressing another research question.
- Utilize a duplication removal utility to remove duplicates based on the threshold value.
- Record observations and compare results with Experiment 1.

### Evaluation Metrics:

#### - Number of Unique Invariants Identified:

- This metric will assess the impact of removing duplicates on the diversity and richness of the invariants generated.
- A higher number of unique invariants could indicate more thorough test coverage.

#### - Mutation Score:

- Apply mutation testing to the application and use the test suite to detect mutants.
- The mutation score, i.e., the percentage of mutants detected, can serve as a robust indicator of the quality of the test suite.

Investigating the impact of duplicate request reduction on AGORA's performance is crucial for understanding its efficiency and effectiveness in API testing. By conducting experiments and evaluating key metrics, we aim to provide insights into the significance of removing duplicates and its implications for test coverage and quality. These findings will contribute to enhancing the reliability and efficiency of AGORA as a tool for automated API testing.

## Initial Findings - Experiment 1:

Authors' Output - 50 duplicate `createPlaylist` REST API - 43 invariants

The screenshot shows an Excel spreadsheet titled "Authors' Output - 50 duplicate createPlaylist REST API - 43 invariants". The spreadsheet contains two tabs: "Invariants\_50\_modified" and "Invariants\_50". The "Invariants\_50" tab displays 50 rows of API requests, each starting with "main.user\_id...". The "Invariants\_50\_modified" tab displays 43 rows of invariants, each starting with "return.owner.id...". The invariants include various assertions about user IDs, names, descriptions, collaborative tracks, follower counts, and track URLs.

Experiment 1 – (threshold value) duplicate REST apis - 50 invariants  
Findings - 7 new invariants found.

This screenshot shows an Excel spreadsheet with two tabs: "Invariants" and "Invariants\_50". The "Invariants" tab contains 50 rows of invariants, mostly identical to those in the previous screenshot. The "Invariants\_50" tab contains 7 rows of new invariants, which are highlighted in yellow. These new invariants include assertions about specific URLs and user types.

## RQ3. How do different configurations of JSONMutator mutation operators affect the failure detection ratio achieved by AGORA?

JSONMutator is employed to introduce specific faults into the existing dataset deliberately. This is done to test whether the invariants identified by Daikon can successfully detect these introduced anomalies. With respect to the research question , We have decided to perform 3 experiments so as to explore the impact of the different factors on the invariant generation process and consequently on the failure detection capabilities of AGORA.

### Updating OpenAPI Format with Existing Dataset

The OpenAPI Specification serves as a crucial guide for understanding an API's intended behavior. Changes in its structure or content are likely to affect the types of invariants that can be derived. In our investigation, we focused on the Amadeus Hotel API, comparing the most recent Swagger file obtained from APIs.guru with an earlier version. This comprehensive analysis revealed several key updates in the newer Swagger files. Here are the highlights:

1. Version Update: The API version was updated from 3.0.5 to 3.0.8, indicating potential bug fixes or updates in API functionality.

2. Consistent Endpoints: Both versions feature the same endpoints, operating similarly, which shows a stable API functional range.
3. Parameter Adjustments: Minor modifications in parameters and their descriptions were noted, potentially impacting the generation of invariants.
4. Enhanced Enumerations: The latest version introduces expanded enum values, providing finer categorization of hotel offers.
5. Addition of New Properties and Descriptions: More detailed properties and descriptions have been incorporated into various definitions.

The API responses were intentionally left unchanged to evaluate how AGORA performs when API responses do not align with the OpenAPI specification.

The updated Swagger file, along with the existing test cases (100 Amadeus API), was processed through BEET, successfully producing trace and declaration files. These files were then analyzed by Daikon using a Docker command, resulting in the generation of invariants. The outcome revealed that the modified version produced fewer invariants compared to the original version, with Daikon identifying some unique invariants for the updated version. This discrepancy is primarily linked to changes in the parameters within the modified Swagger file.

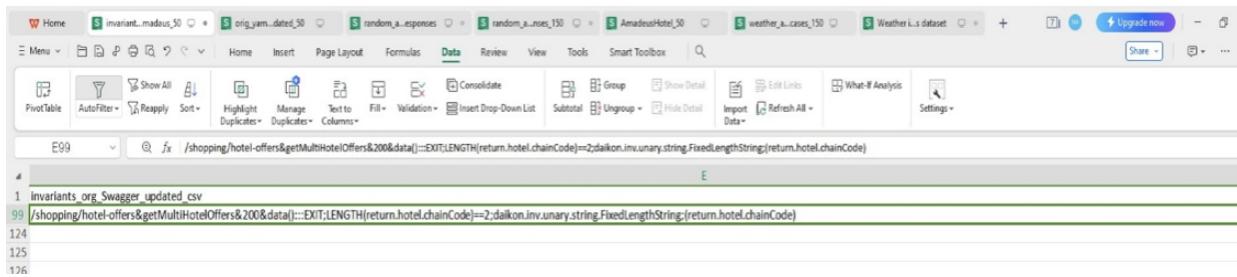
Here is the snapshot of the findings

A	B
1	invariants generated with modified swagger
9	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers&price&variations&changes()::EXIT{return.base == return.total;daikon.inv.binary.twoString.StringEqual();return.base}
20	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{return.category == return.rateCode;daikon.inv.binary.twoString.StringEqual();return.category}
21	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{return.category == return.rateFamilyEstimated.code;daikon.inv.binary.twoString.StringEqual();return.category}
22	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{return.commission.description == return.description;daikon.inv.binary.twoScalar.IntPtrEqual();return.commission.description}
92	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{size(return.policies.guarantee.acceptedPayments.creditCards) >= size(return.price.variations.changes);daikon.inv.binary.twoScalar.IntPtrGreaterEqual();size(return.policies.guarantee.acceptedPayments.creditCards)}
93	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{size(return.policies.guarantee.acceptedPayments.methods) <= size(return.price.taxes);daikon.inv.binary.twoScalar.IntPtrLessEqual();size(return.policies.guarantee.acceptedPayments.methods)}
94	/shopping/hotel-offers&getMultiHotelOffers&200&data&offers()::EXIT{size(return.policies.guarantee.acceptedPayments.methods) <= size(return.price.variations.changes);daikon.inv.binary.twoScalar.IntPtrLessEqual();size(return.policies.guarantee.acceptedPayments.methods)}
95	/shopping/hotel-offers&getMultiHotelOffers&200()::EXIT{return.data.hotel == null;daikon.inv.unary.scalar.OneOfScalar{return.data.hotel}}
96	/shopping/hotel-offers&getMultiHotelOffers&200()::EXIT{return.data.offers == null;daikon.inv.unary.scalar.OneOfScalar{return.data.offers}}
99	
100	
101	

Such a variation in the number of invariants directly impacts the Fault Detection Ratio, as a reduced count of invariants could lead to some faults going undetected.

## 2<sup>nd</sup> Experiment Updating API Responses with Previous Version of OpenAPI Format

In this part of the study, we examined the impact of changes in API responses, while keeping the OpenAPI specification version constant, on the detection of invariants. We manually altered the parameter values and removed some parameters in the response field of the Amadeus Hotel API. Subsequently, invariants were generated using Daikon. A comparison between the invariants from the original and modified versions revealed minimal differences; specifically, there was only one invariant that differed in the modified version. From this analysis, we deduce that minor modifications in the API responses do not significantly alter the invariants generated. Consequently, such changes are unlikely to substantially affect the fault detection ratio.



### 3<sup>rd</sup> Experiment Time Based Mutation Analysis

The approach of Time-based Mutation Analysis focuses on mutations affecting the time-related data within API responses, an aspect crucial for APIs in various industries that often contain at least one time-based variable. This analysis becomes particularly important for events with time constraints.

Taking an HRMS API (Human Resource Management System) as an example, consider if the API responses include time-sensitive data covering scenarios like:

- Adjusting Timestamps: Simulating cases where time data is set too far into the future or past compared to the expected timeline.
- Modifying Durations and Intervals: Altering time spans, for instance, extending or shortening a reservation duration.
- Changing Expiration Dates: Assessing the system's response to data that is either already expired or will expire earlier or later than initially planned.

To implement time-based mutations, the following steps are necessary:

- Identifying Time-Sensitive Data: Adapt the mutation logic to pinpoint fields in JSON objects representing time data, either through naming conventions or by directly indicating target fields.
- Developing Custom Mutation Operators: Tailor or set up mutation operators within JSONMutator to aptly manipulate time-based data. This includes:
  - Adjusting dates and times within certain limits through JSON Mutation Properties.
  - Injecting specific time-related errors, such as scheduling an event to start after its end time.
- Mutation Testing with AGORA: Utilize the AGORA testing framework on mutated test cases featuring time-based discrepancies.

If executed correctly, AGORA should identify faults (time-based mutants) assuming the time-based invariants are accurately generated by Daikon

## Conclusion

In conclusion, this replication study endeavors to provide comprehensive insights into the scalability, efficiency, and effectiveness of AGORA for API testing automation. By systematically addressing the research questions outlined in the original paper, this study aims to validate AGORA's claims and assess its suitability for real-world deployment. The findings of this study will serve as valuable guidance for software developers and testers, facilitating informed decision-making regarding the adoption of AGORA in their testing workflows. Furthermore, this replication study contributes to the advancement of automated API testing methodologies, fostering the development of more reliable and efficient software systems.