

IN-THE-FLOW AGENTIC SYSTEM OPTIMIZATION FOR EFFECTIVE PLANNING AND TOOL USE

000
001
002
003
004
005 **Anonymous authors**
006 Paper under double-blind review
007
008
009

ABSTRACT

010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
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
569
570
571
572
573
574
575
576
577
578
579
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
749
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
849
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
949
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
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
1040
1041
1042
1043
1044
1045
1046
1047
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
1149
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
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
1249
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
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
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
1349
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
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
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
1449
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
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
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
1549
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
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
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
1649
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
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
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
1749
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
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
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
1849
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
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904<br

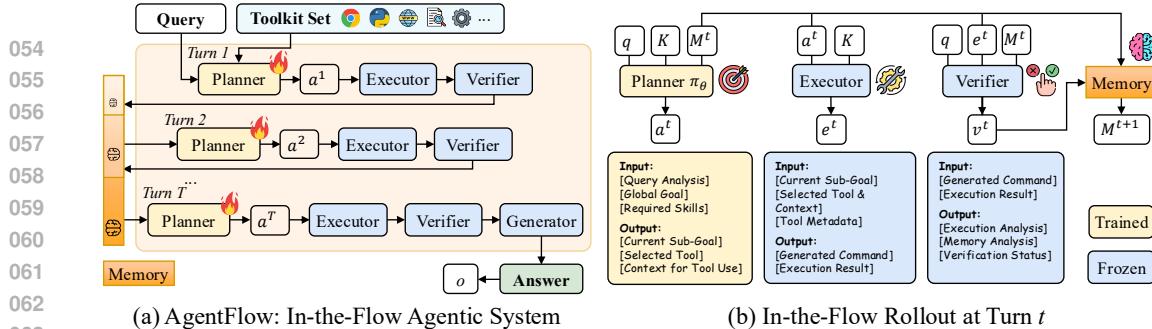


Figure 2: (a) Overview of AGENTFLOW, a trainable agentic system for in-the-flow planning and tool use. Four modules (planner, executor, verifier, generator) coordinate via a shared evolving memory M and toolset K , given a query q . The planner policy is optimized on-policy *inside* the system’s multi-turn loop to enable adaptive, long-horizon reasoning. (b) A single state transition, showing the action a^t , execution result e^t , and verifier signal v^t that update the memory from M^t to M^{t+1} .

A complementary line of work augments LLMs with external tools (e.g., web search, code execution) for knowledge retrieval and precise computation. Tool-integrated reasoning (TIR) extends reinforcement learning with verifiable rewards to learn *when* and *how* to call tools by interleaving reasoning (e.g., `<think>`) with tool invocations (e.g., `<tool_call>`) under full context (Jin et al., 2025; Song et al., 2025; Chen et al., 2025; Feng et al., 2025). Early systems supported only a single tool type, whereas recent work enables multi-tool settings by encoding tool metadata into prompts (Dong et al., 2025; Qian et al., 2025a; Zhang et al., 2025). However, these methods still train a *single*, monolithic policy under multi-turn full-context reasoning, which introduces scaling challenges: (i) *training* becomes increasingly unstable as horizons lengthen, tool diversity grows, and environments shift with tool feedback (Wang et al., 2025c; Mai et al., 2025; Moonshot AI, 2025; Xue et al., 2025); and (ii) *inference*-time generalization remains brittle to unseen tasks or tools (Dong et al., 2025; Hu et al., 2025b).

Agentic systems (Wu et al., 2024; Hong et al., 2024; Hu et al., 2025b) offer a promising alternative to monolithic tool-integrated reasoning models. They consist of multiple modules—often distinct LLMs with prescribed roles (e.g., planner, critic) or specialized components with dedicated tools and capabilities (e.g., executor, coder)—that coordinate via shared memory and inter-module communication. By decomposing problems into sub-goals and iterating over multiple turns, these systems can tackle tasks that demand diverse tools, long horizons, or multi-stage reasoning. However, achieving robust coordination in such systems ultimately requires *training*, since handcrafted logic or static prompting cannot reliably capture when and how modules should collaborate, adapt to evolving tool outputs, or recover from early mistakes. At the same time, they introduce new *training* challenges: modules coordinate sequentially, outcome feedback propagates through long reasoning chains, and state distributions shift with evolving tool outputs. As a result, most systems remain *training-free*, relying on handcrafted logic or prompting heuristics. While some employ supervised fine-tuning or preference optimization for key modules (Motwani et al., 2024; Park et al., 2025), these off-policy approaches are decoupled from live dynamics and learn poorly from downstream successes or failures. Thus, agentic systems struggle with sparse rewards, brittle adaptation, and inefficient orchestration in dynamic environments.

To address the central challenge of learning long-horizon reasoning with sparse rewards in tool-integrated agentic systems, we introduce AGENTFLOW, a *trainable* framework for effective planning and tool use (Figure 2). AGENTFLOW comprises four specialized modules—planner, executor, verifier, and generator—that interact iteratively over multiple turns via a shared evolving memory and a toolset. The system operates *in the flow*, with each turn cycling through planning, execution, and verification. Unlike prior agentic systems, AGENTFLOW directly optimizes its planner on-policy, *inside* the live multi-turn loop, allowing it to dynamically adapt to trajectories shaped by tool calls, verifier signals, and memory updates. This evolving memory serves as a deterministic, structured record of the reasoning process, enabling transparent state tracking, controllable behavior, and bounded context growth.

To train the planner on-policy within this agentic system, we need to overcome the long-horizon credit assignment problem inherent to sparse, trajectory-level rewards. We introduce *Flow-based Group Refined Policy Optimization* (Flow-GRPO, Figure 4), an on-policy algorithm designed for

108 this setting. Flow-GRPO operates on *in-the-flow* rollouts, which capture the full trajectory of states,
 109 actions, and tool events induced by the live system. Instead of attempting to assign credit with
 110 brittle, intermediate heuristics, we assign a single, verifiable final-outcome reward to the entire trajec-
 111 tory and *broadcast* it to every turn. This design effectively transforms the multi-turn reinforcement
 112 learning challenge into a series of single-turn updates: at each turn, the planner has access to the full
 113 memory context and receives a consistent reward signal aligned with global success. This approach,
 114 coupled with group-normalized advantages to stabilize training, enables robust credit assignment
 115 and allows the planner to learn effective long-horizon strategies from sparse feedback.

116 We evaluate AGENTFLOW on ten benchmarks across diverse reasoning domains, as results high-
 117 lighted in Figure 1. AGENTFLOW substantially outperforms top-performing specialized tool-
 118 integrated reasoning models and agentic systems, achieving average accuracy by 14.9% on
 119 knowledge-intensive search, 14.0% on broader agentic tasks, 14.5% on mathematical reasoning, and
 120 4.1% on scientific reasoning (§4.2). Notably, our 7B-backbone system even surpasses the ~200B-
 121 parameter GPT-4o (Hurst et al., 2024) across all domains. Further analyses confirm that our in-
 122 the-flow optimization with Flow-GRPO is crucial, far surpassing offline supervised tuning (§4.3).
 123 The trained planner learns to optimize planning, enhance tool-calling reliability, and discover ef-
 124 fective solution pathways (§4.4). Moreover, our training approach proves highly efficient, leading
 125 to increased rewards and condensed responses compared to traditional tool-integrated RL methods
 126 (§4.5). Finally, we demonstrate that these benefits generalize, with consistent gains from scaling
 127 backbone size and turn budget (§4.6).

NEW

128 Our work makes three key contributions: (1) We present AGENTFLOW, a trainable *in-the-flow* agen-
 129 tic system that directly optimizes its planner *inside* the multi-turn loop. By coordinating specialized
 130 modules through an evolving memory, it enables adaptive long-horizon planning and robust tool
 131 orchestration. (2) We introduce *Flow-GRPO*, an on-policy, outcome-driven algorithm that hat *con-*
 132 *verts* multi-turn RL into a sequence of tractable *single-turn* policy updates by *broadcasting* a sin-
 133 gle, verifiable final-outcome reward to every turn. (3) Through comprehensive experiments on ten
 134 benchmarks, we show that AGENTFLOW with a 7B backbone outperforms specialized baselines and
 135 even larger proprietary models. Further analyses reveal improved planning, enhanced tool-calling
 136 reliability, and positive scaling with model size and turn budgets.

2 PRELIMINARY

138 **Reinforcement learning for reasoning LLMs.** Recent progress in reasoning LLMs has been sig-
 139 nificantly driven by reinforcement learning from outcome feedback, using a verifiable reward sig-
 140 nal (Shao et al., 2024; Yu et al., 2025). This paradigm fine-tunes a language model to maximize
 141 an outcome-based reward while remaining close to a reference policy. Formally, the objective is to
 142 optimize a policy LLM π_θ to generate a response o for a given query q from dataset \mathcal{D} :

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, o \sim \pi_\theta(\cdot | q)} [R(q, o)] - \beta \mathbb{D}_{\text{KL}}(\pi_\theta(o | q) \parallel \pi_{\text{ref}}(o | q)), \quad (1)$$

143 where $R(q, o)$ is the outcome-based reward, π_{ref} is a reference model to prevent policy collapse, and
 144 β controls KL regularization. Algorithms like Group Relative Policy Optimization (GRPO) (Shao
 145 et al., 2024) implement this by sampling groups of responses, normalizing advantages by their re-
 146 wards, and updating the policy with a clipped objective to encourage high-reward outputs.

147 **Tool-integrated reasoning models (LLM agents).** LLMs can be augmented with external tools
 148 to access knowledge and perform precise computation under reinforcement learning with outcome-
 149 based reward. As shown in Figure 3(a), the LLM *interleaves* reasoning and tool calls, produc-
 150 ing a chain of thought within `<think></think>` tokens followed by tool invocations (e.g.,
 151 `<tool_call></tool_call>`). The resulting trajectory τ is a sequence of model generations
 152 and tool observations: $\tau = \{s^1, a^1, e^1, \dots, s^T, a^T\}$, where s^t denotes the context, a^t the generated
 153 action (thought + tool call), and e^t the tool’s execution result. The policy model π_θ is then trained to
 154 maximize a final outcome reward. Prior work has explored single- and multi-tool settings for search
 155 and code execution (Jin et al., 2025; Chen et al., 2025; Feng et al., 2025; Qian et al., 2025a).

156 **Agentic systems with tool usage.** An alternative approach is the use of agentic systems (Wu et al.,
 157 2024; Hong et al., 2024; Lu et al., 2025). As shown in Figure 3(b), these frameworks deploy mul-
 158 tiple specialized modules—often distinct LLMs with carefully designed prompts and roles—with
 159 in a collaborative workflow. By decomposing tasks and assigning subproblems to modules with dedi-
 160 cated tools and capabilities (e.g., planner, coder, critic), they can address complex problems such as

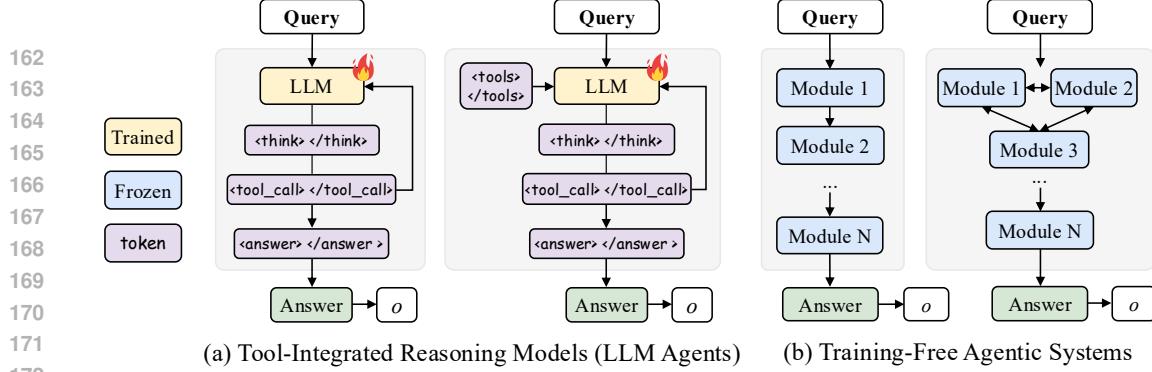


Figure 3: **Comparison of two paradigms of LLMs with tool use.** (a) Monolithic tool-integrated reasoning models train a single policy to interleave reasoning (e.g., `<think>`) and tool calls (e.g., `<tool_call>`) within a single, full-context trajectory. (b) Agentic systems decompose tasks across multiple specialized modules (e.g., planner, coder) that collaborate. These systems are typically *training-free*, orchestrated by handcrafted logic or prompting.

web browsing, document processing, and multi-stage programming that exceed the scope of a single model. A central limitation, however, is that these systems are typically *training-free*: modules remain frozen pre-trained models orchestrated by handcrafted logic or prompting heuristics.

FIX

3 IN-THE-FLOW AGENTIC SYSTEM OPTIMIZATION

We aim to bridge the gap between trainable but monolithic reasoning models and flexible yet static agentic systems. We present AGENTFLOW, a flexible and trainable agentic system that integrates four specialized modules with an evolving memory (§3.1). Unlike prior agentic systems, AGENTFLOW directly optimizes the planner *within* the multi-turn loop of an agentic system (§3.2).

3.1 AGENTFLOW: AN IN-THE-FLOW AGENTIC SYSTEM

We propose AGENTFLOW, a general-purpose tool-integrated agentic framework for solving complex reasoning tasks through fine-grained planning and effective tool use within a multi-turn architecture. As shown in Figure 2, the framework comprises four specialized modules—**Action Planner** \mathcal{P} , **Tool Executor** \mathcal{E} , **Execution Verifier** \mathcal{V} , and **Solution Generator** \mathcal{G} —coordinated by a shared evolving memory M and a toolset K . These modules interact sequentially and iteratively to perform *action planning*, *tool execution*, *context verification*, and *solution generation*, thereby enabling tool-integrated reasoning across multiple turns.

We formalize AGENTFLOW’s problem-solving process as a multi-turn Markov Decision Process (MDP). Given a query q and a toolset K , the system proceeds for a variable number of turns. Let M^t denote the memory state before turn t (with M^1 initialized from q). At turn t , the planner \mathcal{P} (a trainable policy π_θ) formulates a sub-goal, selects an appropriate tool $k \in K$, and retrieves relevant context from memory, producing an action: $a^t \sim \pi_\theta(a^t | q, K, M^t)$.

The executor \mathcal{E} invokes the chosen tool with context, yielding an execution observation $e^t \sim \mathcal{E}(e^t | a^t, K)$. The verifier \mathcal{V} then evaluates whether e^t is valid and whether the accumulated memory is sufficient to solve the query, producing a binary verification signal $v^t \sim \mathcal{V}(v^t | q, e^t, M^t)$. If $v^t = 0$, the memory is updated deterministically to incorporate new evidence: $M^{t+1} = f_{\text{mem}}(M^t, a^t, e^t, v^t)$, where $f_{\text{mem}}(\cdot)$ denotes the memory-update function, which records agent-process information in a concise, structured form along with contextual details such as time, turn index, and error signals.

The process repeats until $v^t = 1$ (termination) or a predefined maximum turn budget is reached. Upon termination at turn T , the solution generator \mathcal{G} produces the final solution o , conditioned on the query and the accumulated memory: $o \sim \mathcal{G}(o | q, M^T)$.

This formulation decomposes multi-turn, tool-integrated reasoning into structured, observable transitions. After T turns, the trajectory $\tau = \{(a^t, e^t, v^t)\}_{t=1}^T$ records the history of planning, execution, and verification. The joint generative process can be written as

$$p_\theta(\{a^t, e^t, v^t\}_{t=1}^T, o | q) = \left[\prod_{t=1}^T \pi_\theta(a^t | q, K, M^t) \mathcal{E}(e^t | a^t, K) \mathcal{V}(v^t | q, e^t, M^t) \right] \mathcal{G}(o | q, M^T), \quad (2)$$

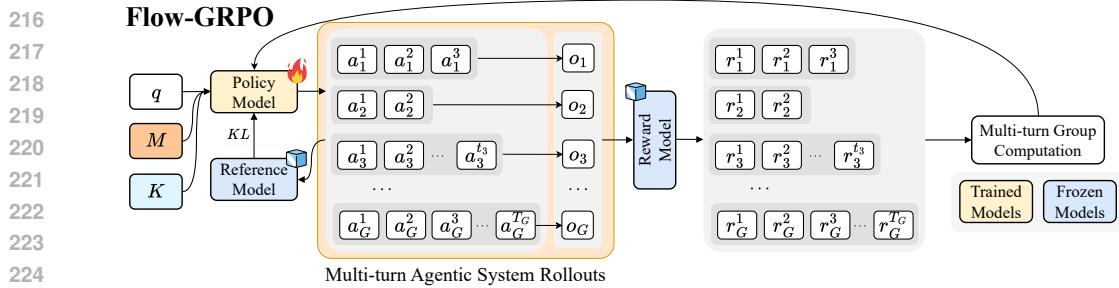


Figure 4: **Optimization for our proposed agentic system AGENTFLOW.** Given a query q , an evolving memory M , and a toolset K , the policy model generates actions that target sub-goals and select tools. It is trained via *Flow-based Group Refined Policy Optimization* (Flow-GRPO), which enables multi-turn reinforcement learning and stable optimization under collaborative dynamics.

where $\{a^t, e^t, v^t\}_{t=1}^T$ are explicit realizations of the latent reasoning chain. Importantly, unlike latent thoughts behind trajectories, our memory M is an explicit and deterministic record of the reasoning process, ensuring transparency and controllability of multi-turn decisions.

3.2 IN-THE-FLOW REINFORCEMENT LEARNING OPTIMIZATION

We target tool-integrated *agentic systems* operating under *long-horizon* tasks with *sparse* rewards. In this setting, the **Action Planner** (the trainable policy of AGENTFLOW) selects a *sequence* of interdependent actions while the state (q, K, M^t) evolves with tool results and verifier feedback. Conventional *offline* training—e.g., supervised fine-tuning or preference fine-tuning on curated traces—optimizes the planner *outside* the active loop (Motwani et al., 2024; Park et al., 2025). This decoupling prevents real-time coordination with the executor, verifier, and solution generator, induces distribution shift between training and deployment, and provides limited guidance about *which* intermediate decisions truly matter. As a result, planners often adapt poorly to multi-turn dynamics; early errors cascade, and post-hoc fixes are brittle.

In-the-flow learning. To address these issues, we optimize the planner *in the flow* of execution. We roll out the full AGENTFLOW system under the current policy, collect the actual trajectory τ of states, actions, and tool events it induces, and update the policy within the agentic system using a verifiable final-outcome signal. This exposes the multi-turn credit-assignment problem directly and trains the planner on the exact states it will face at inference. Our objective, Flow-GRPO, is designed to stabilize learning under sparse, trajectory-level rewards over multiple turns.

As established in §3.1, rollouts in AGENTFLOW define a finite-horizon MDP with a variable horizon T . At turn t , the planner observes the state (q, K, M^t) , selects an action a^t , the executor and verifier return (e^t, v^t) , and the memory updates deterministically to M^{t+1} .

Policy optimization objective. The planner policy π_θ is trained to maximize the expected return over on-policy rollouts. Let $R(\tau)$ be the reward for a complete trajectory τ . The objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \quad \theta^* = \arg \max_{\theta} \mathcal{J}(\theta), \quad (3)$$

where a rollout τ is the sequence of decisions $\{a^t\}_{t=1}^T$ generated on-policy by π_θ .

Final-outcome reward. Assigning credit to intermediate actions is challenging because each a^t influences the final solution only indirectly, and their value may only emerge after several turns (e.g., error or improvement accumulation). To avoid brittle local feedback, we adopt a *final-outcome-based reward*: every action within a rollout receives the same global reward signal, based on the correctness of the final solution o with respect to query q and ground truth y^* :

$$r = R(a^t) = \bar{R}(o, q, y^*), \quad \forall t = 1, \dots, T, \quad (4)$$

where $\bar{R}(o, q, y^*) \in \{0, 1\}$ is assigned by an LLM-as-judge rubric for semantic, numeric, and option-level equivalence (see §E.3). This propagates a trajectory-level success signal back through the reasoning chain, aligning every decision a^t with global correctness.

Objective function. We formalize **Flow-based Group Refined Policy Optimization** for the planner. The goal is to optimize the policy π_θ by maximizing the expected return over a group of parallel

270 rollouts. For each query-label pair from training corpus $(q, y^*) \sim \mathcal{D}$, we sample a group of G on-
 271 policy trajectories $\{\tau_i\}_{i=1}^G$ by running the current behavior policy $\pi_{\theta_{\text{old}}}$ inside AGENTFLOW, where
 272 $\tau_i = \{a_i^1, \dots, a_i^{T_i}, o_i\}$. Let $s_i^t = (q, K, M_i^t)$ be the state at turn t of rollout i , a_i^t the planner’s action
 273 (a token sequence of length $|a_i^t|$), and o_i the final response. This structure is key to addressing
 274 the long-horizon credit assignment challenge: by broadcasting a single trajectory-level reward to
 275 all turns, we effectively decompose the *multi-turn RL* problem into a *set of independent, single-turn*
 276 policy updates; we provide a formal proof of this equivalence and analyze its convergence properties
 277 in §B. Each update for an action a_i^t is conditioned on the full historical context encapsulated in the
 278 state s_i^t and receives the same global success signal, simplifying optimization. The objective is

$$\mathcal{J}_{\text{Flow-GRPO}}(\theta) = \mathbb{E}_{(q, y^*) \sim \mathcal{D}, \{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{T_i} \sum_{t=1}^{T_i} \frac{1}{|a_i^t|} \sum_{j=1}^{|a_i^t|} \min \left\{ \rho_{i,j}^t A_i^t, \text{clip}(\rho_{i,j}^t, 1 - \epsilon, 1 + \epsilon) A_i^t \right\} - \beta \mathbb{D}_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}}) \right], \quad (5)$$

284 where T_i is the (variable) number of turns in rollout i , and

$$\rho_{i,j}^t = \frac{\pi_{\theta}(a_{i,j}^t \mid s_i^t, a_{i,1:j-1}^t)}{\pi_{\theta_{\text{old}}}(a_{i,j}^t \mid s_i^t, a_{i,1:j-1}^t)} \quad (6)$$

288 is the token-level importance ratio for the j -th token of a_i^t , $\epsilon > 0$ is the PPO clipping parameter, and
 289 $\beta > 0$ controls the KL penalty to a fixed reference policy π_{ref} .

290 **Group-normalized advantages.** Because the reward in Eq. 4 is a single trajectory-level signal,
 291 the per-turn advantage A_i^t is constant over t within a rollout i . We reduce variance and sharpen
 292 credit assignment across the group by using a *group-normalized* advantage:

$$A_i^t = \frac{\bar{R}(o_i, q, y^*) - \text{mean}(\{\bar{R}(o_k, q, y^*)\}_{k=1}^G)}{\text{std}(\{\bar{R}(o_k, q, y^*)\}_{k=1}^G)}. \quad (7)$$

296 **Technical contribution summary.** To tackle *long-horizon, sparse-reward* training in multi-
 297 module agentic systems, we propose Flow-GRPO. This novel algorithm (i) formalizes the multi-
 298 turn RL problem in agentic systems into a series of *tractable, single-turn policy updates*, and (ii)
 299 *broadcasts* a single trajectory-level outcome to every turn to align local planner decisions with
 300 global success. Training uses an LLM-based rubric to assign verifiable final-outcome rewards,
 301 with group-normalized advantages, KL regularization, and clipping to stabilize learning.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

307 In our main experiments, all modules—Action Planner, Tool Executor, Executive Verifier, and Solu-
 308 tion Generator—are instantiated with the *Qwen2.5-7B-Instruct* model (Yang et al., 2024a). Among
 309 these, only the *Action Planner* is trainable. The system operates with five interactive tools: *Base*
 310 *Generator* is an instance of *Qwen2.5-7B-Instruct* that acts as the default reasoning engine if the
 311 planner decides not to use an external tool; *Python Coder* generates and executes Python code given
 312 a query and returns the execution result; *Google Search* searches the web and returns a summariza-
 313 tion of Top-K search results; *Wikipedia Search* searches articles matching a given query and returns
 314 a summarization; and *Web Search* returns summarized information from a given web page. Dur-
 315 ing the RL fine-tuning phase, we mix data from Search-R1 (Jin et al., 2025) and DeepMath (He
 316 et al., 2025) as training data, which provides paired question-answer examples across search and
 317 mathematical domains. We use a batch size of 32 with 8 rollouts per sample.

318 To comprehensively evaluate tool-use capabilities of AGENTFLOW, we conduct experiments on four
 319 types of reasoning tasks: (1) *Knowledge-intensive search* including Bamboogle (Press et al., 2023),
 320 2Wiki (Ho et al., 2020), HotpotQA (Yang et al., 2018), and Musique (Trivedi et al., 2022); (2) *Agen-
 321 tic reasoning* such as GAIA (Mialon et al., 2023) (where we adopt the textual split); (3) *Logic-dense
 322 mathematical reasoning* including AIME2024 (Art of Problem Solving, 2025), AMC23 (MAA,
 323 2023), and GameOf24 (Lightman et al., 2023); and (4) *Scientific reasoning* including GPQA (Rein
 324 et al., 2024) and MedQA (Yang et al., 2024c). To mitigate randomness, we report the average accu-
 325 racy across three trials for all experiments. More experimental details are in §C.

Model	Size	Search Intensive						Agentic	
		Bamboogle	2Wiki	HotpotQA	Musique	Avg.	Δ	GAIA	Δ
Qwen-2.5-7B-Instruct	7B-Inst	12.0	23.0	21.0	6.0	15.5	↑41.8	3.2	↑29.9
Qwen-2.5-14B-Instruct	14B-Inst	21.6	26.7	20.0	8.0	19.1	↑38.2	5.5	↑27.6
Qwen-2.5-32B-Instruct	32B-Inst	24.0	26.7	27.0	6.0	20.9	↑36.4	9.5	↑23.6
Llama-3.3-70B-Instruct	70B-Inst	18.4	22.7	52.0	16.0	27.3	↑30.0	3.2	↑29.9
GPT-4o-mini (Hurst et al., 2024)	~8B	40.8	35.6	41.0	15.0	33.1	↑24.2	7.1	↑26.0
GPT-4o (Hurst et al., 2024)	~200B	68.8	49.5	54.0	24.0	49.1	↑8.2	17.3	↑15.8
Supervised Fine-Tuning (SFT)	7B-Inst	12.0	25.9	22.0	6.6	16.6	↑40.7	3.2	↑29.9
Iter-RetGen (Shao et al., 2023)	7B-Inst	36.8	33.6	37.4	17.8	31.4	↑25.9	3.9	↑29.2
Search-R1 (Jin et al., 2025)	7B-Inst	43.2	38.2	37.0	14.6	33.3	↑24.0	19.1	↑14.0
ZeroSearch (Sun et al., 2025)	7B-Base	27.8	35.2	34.6	18.0	28.9	↑28.4	16.5	↑16.6
ReSearch (Chen et al., 2025)	7B-Base	42.4	47.6	43.5	22.3	39.0	↑18.3	17.3	↑15.8
StepSearch (Wang et al., 2025d)	7B-Base	40.0	36.6	38.6	22.6	34.5	↑22.8	—	—
VerlTool (Jiang et al., 2025)	7B-Base	46.4	45.3	44.8	19.3	39.0	↑18.3	11.2	↑21.9
AutoGen (Wu et al., 2024)	7B-Inst	59.6	44.0	50.0	15.9	42.4	↑14.9	6.3	↑26.8
AGENTFLOW	7B-Inst	58.4	60.0	51.3	19.2	47.2	↑12.1	17.2	↑15.9
AGENTFLOW (w/ Flow-GRPO)	7B-Inst	69.6	77.2	57.0	25.3	57.3	—	33.1	—

Table 1: **Accuracy comparison on search-intensive and agentic tasks.** 7B-Base refers to Qwen-2.5-7B-Base and 7B-Inst refers to Qwen-2.5-7B-Instruct. AutoGen and our AGENTFLOW method are agentic systems, which use Qwen-2.5-7B-Instruct for the LLM-powered agents and tools for fair comparison. We visualize the gains of AGENTFLOW to the each baseline in the Δ columns.

Model	Size	Math Reasoning				Scientific Reasoning				
		AIME24	AMC23	GameOf24	Avg.	Δ	GPQA	MedQA	Avg.	
Qwen-2.5-7B-Instruct	7B-Inst	6.7	47.5	33.0	29.1	↑22.5	34.0	66.0	50.0	↑13.5
Qwen-2.5-14B-Instruct	14B-Inst	6.7	60.0	25.0	30.6	↑21.0	31.0	75.0	53.0	↑10.5
Llama-3.3-70B-Instruct	70B-Inst	6.7	47.5	31.0	28.4	↑23.1	35.0	67.0	51.0	↑12.5
Llama-3.1-405B-Instruct	405B-Inst	26.7	47.5	23.0	32.4	↑19.1	30.0	62.0	46.0	↑17.5
GPT-4o-mini (Hurst et al., 2024)	~8B	13.3	57.5	16.0	28.9	↑22.6	27.0	66.0	46.5	↑17.0
GPT-4o (Hurst et al., 2024)	~200B	13.3	60.0	32.0	35.1	↑16.4	31.0	60.0	45.5	↑18.0
Supervised Fine-Tuning (SFT)	7B-Inst	6.7	47.5	33.0	29.1	↑22.5	34.0	66.0	50.0	↑13.5
SimpleRL-reason (Zeng et al., 2025b)	7B-Base	16.7	60.0	33.0	36.6	↑15.0	45.0	65.0	50.0	↑13.5
Open-Reasoner-Zero (Hu et al., 2025a)	7B-Base	16.7	54.9	32.0	34.5	↑17.0	34.0	54.0	44.0	↑19.5
General-Reasoner (Ma et al., 2025)	7B-Base	13.3	55.0	33.0	33.8	↑17.7	35.5	61.0	48.3	↑15.2
Luffy (Yan et al., 2025)	7B-Inst	30.7	44.8	33.0	36.2	↑15.3	34.0	77.0	55.5	↑8.0
TIR (Yang et al., 2024b)	7B-Inst	10.0	50.0	33.0	31.0	↑20.5	42.0	76.8	59.4	↑4.1
ToRL (Li et al., 2025b)	7B-Inst	20.0	60.0	31.0	37.0	↑14.5	35.0	76.5	55.8	↑7.7
AutoGen (Wu et al., 2024)	7B-Inst	13.3	57.5	24.0	31.6	↑19.9	42.0	72.0	57.0	↑6.5
AGENTFLOW	7B-Inst	16.7	47.4	31.0	31.7	↑19.8	37.0	76.0	56.5	↑7.0
AGENTFLOW (w/ Flow-GRPO)	7B-Inst	40.0	61.5	53.0	51.5	—	47.0	80.0	63.5	—

Table 2: **Accuracy comparison of mathematical and scientific reasoning tasks.**

4.2 MAIN RESULTS

Baselines. As presented in Tables 1 and 2, we include five categories of baselines: (1) *Open-source LLMs*: Qwen2.5 (Yang et al., 2024a), Llama-3.1, and Llama-3.3 (Dubey et al., 2024); (2) *Proprietary LLMs*: GPT-4o-mini and GPT-4o; (3) *Reasoning LLMs*: supervised fine-tuning (Yang et al., 2024b), SimpleRL-reason, Open-Reasoner-Zero, General-Reasoner, and LUFFY; (4) *Tool-integrated reasoning LLMs*: both search-enhanced, including Iter-RetGen, Search-R1, ZeroSearch, ReSearch, StepSearch, and VerlTool, and code-enhanced, including TIR and ToRL; (5) *Training-free agentic system*: AutoGen. More details on baseline implementations are in §C.3.

Key insights. AGENTFLOW consistently outperforms all baseline models by large margins. Compared to the best-performing 7B models without tool integration, AGENTFLOW achieves absolute gains of 40.7% on search (SFT), 29.9% on agentic reasoning (SFT), 15.0% on math (SimpleRL-reason), and 8.0% on scientific tasks (Luffy). Against specialized tool-integrated systems, AGENTFLOW surpasses the top models by 14.9% in search (AutoGen), 14.0% in agentic reasoning (Search-R1), 14.5% in math (ToRL), and 4.1% in science (TIR). Notably, our 7B-backbone AGENTFLOW even outperforms the ~200B-parameter GPT-4o across all domains, with gains ranging from 8.2% to 18.0%. A detailed analysis is provided in §D.1.

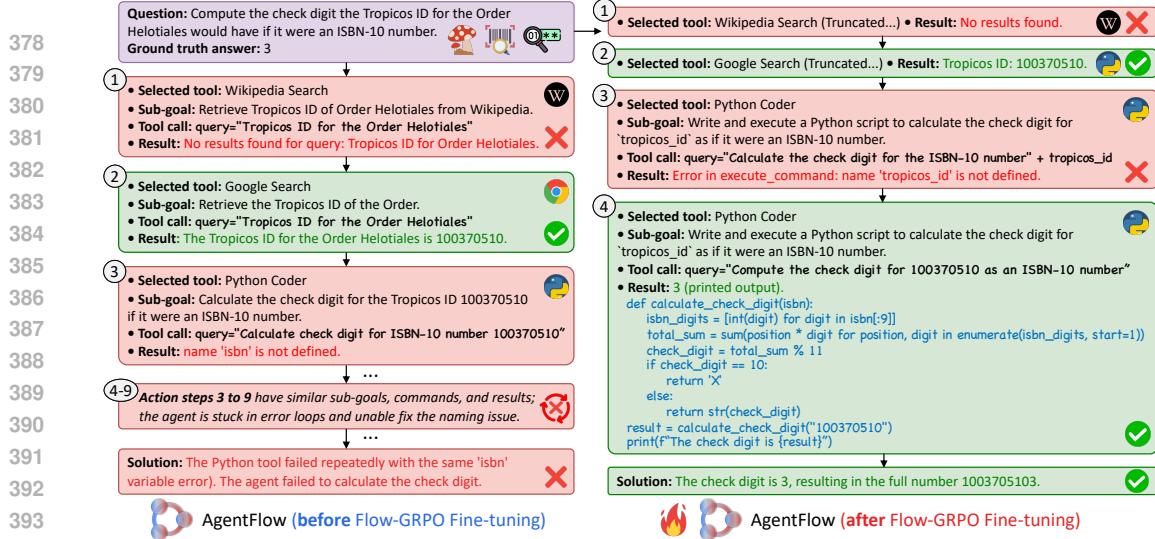


Figure 5: **One case study example.** Initially failed with repetitive errors (left), AGENTFLOW, trained with Flow-GRPO, explores a new solution pathway at turn 4 after two failed attempts (right).

4.3 TRAINING STRATEGIES ON THE PLANNER

We conduct an ablation study to analyze the impact of different training strategies for the *Action Planner* module in AGENTFLOW, with results reported in Table 3. The executor, verifier, and generator modules remain fixed as Qwen2.5-7B-Instruct, consistent with our main setup (§4.1).

Planner Model	Training	Bamboogle	2Wiki	GAIA	AIME24	AMC23	GameOf24	Avg.
Qwen-2.5-7B	Frozen	58.4	60.0	17.2	16.7	47.4	31.0	38.5
GPT-4o	Frozen	65.0 \uparrow 6.6	70.0 \uparrow 10.0	23.6 \uparrow 6.4	16.7 \uparrow 0.0	48.7 \uparrow 1.3	42.0 \uparrow 11.0	44.3 \uparrow 5.8
Qwen-2.5-7B	SFT	30.4 \downarrow 28.0	32.7 \downarrow 27.3	6.3 \downarrow 10.9	3.3 \downarrow 13.4	37.5 \downarrow 9.9	7.0 \downarrow 24.0	19.5 \downarrow 19.0
Qwen-2.5-7B	Flow-GRPO	69.6 \uparrow 11.2	77.2 \uparrow 17.2	33.1 \uparrow 15.9	40.0 \uparrow 23.3	61.5 \uparrow 14.1	53.0 \uparrow 22.0	55.7 \uparrow 17.2

Table 3: Performance comparison of AGENTFLOW across different training methods.

A more capable planner is beneficial, but has limits. Replacing the frozen *Qwen2.5-7B-Instruct* baseline with a stronger proprietary model, GPT-4o, yields only a modest 5.8% average gain. This indicates a key bottleneck that, while a more powerful model improves planning, its static nature prevents co-adaptation with the live dynamics of AGENTFLOW.

Offline SFT leads to performance collapse, while in-the-flow RL is crucial. The limitations of a static planner are further exposed when distilling GPT-4o’s behavior via offline supervised fine-tuning (SFT) on its trajectories as *Action Planner* in AGENTFLOW. This results in a catastrophic performance collapse, with an average accuracy drop of 19.0% compared to the frozen baseline. This failure arises from the token-level imitation objective of SFT, which misaligns with trajectory-level task success and prevents the planner from adapting to dynamic tool feedback or recovering from compounding errors. In contrast, training the planner with our on-policy Flow-GRPO method proves highly effective: by optimizing for the final outcome, the planner learns to handle long-horizon workflows, achieving a 17.2% average gain over the frozen baseline.

4.4 IN-DEPTH ANALYSIS OF OPTIMIZED PLANNING

Flow-GRPO optimizes tool usage. We compare tool usage distributions before and after in-the-flow RL training. Figure 6 shows results on two knowledge-intensive tasks, 2Wiki and MedQA, which exhibit distinct optimization patterns alongside improved task accuracy. For 2Wiki, which requires broad factual knowledge, Flow-GRPO optimizes the planner to increase Google Search usage by 42.0%. In contrast, for the specialized MedQA benchmark, which requires deep, domain-specific information retrieval, fine-tuning shifts the planner away from general tools, reducing Google Search calls (66.2 \rightarrow 10.9%) in favor of in-document Web Search (0 \rightarrow 19.5%) and specialized Wikipedia Search (0 \rightarrow 59.8%). This demonstrates that the planner learns to select task-appropriate tools.

NEW

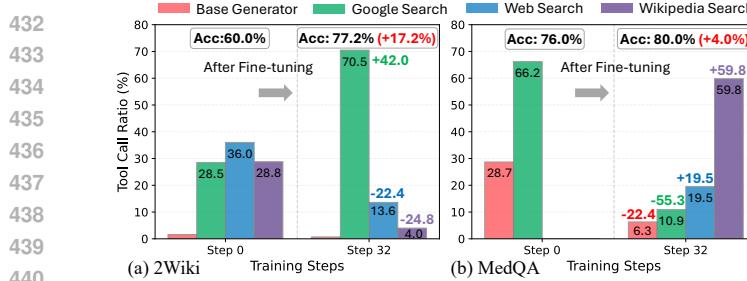


Figure 6: Tool call ratio change by Flow-GRPO fine-tuning.

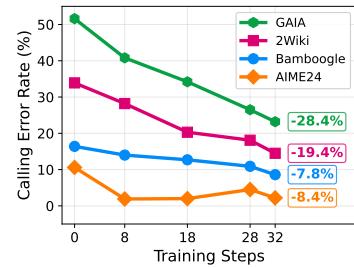


Figure 7: Calling error rate.

Flow-GRPO enhances tool-calling efficacy. A key aspect of the model’s improvement is its increased reliability in tool usage. As shown in Figure 7, the tool-calling error rate consistently decreases across tasks during training, with a reduction of up to 28.4% on GAIA. This trend indicates that the training process not only teaches the model *which* tool to use but also *how* to invoke it correctly with proper arguments and format, leading to more robust and effective tool integration.

Flow-GRPO incentivizes autonomous discovery of new solutions. We further examine qualitative examples in Figure 5 and additional cases in §F. These cases show that AGENTFLOW, trained with Flow-GRPO, develops enhanced capabilities for task planning and tool use. The planner exhibits adaptive efficiency, stronger self-correction, and spontaneous new integration of tools throughout step-by-step problem-solving, autonomously discovering effective solution pathways.

NEW

4.5 TRAINING EFFICIENCY ANALYSIS

455 Optimized planning with increased 456 rewards and condensed responses.

457 We analyze the training dynamics of
458 the AGENTFLOW planner by tracking
459 its average reward and response
460 length on the train set (Figure 8a).
461 Training rewards steadily increase,
462 indicating effective policy improve-
463 ment via Flow-GRPO. Meanwhile,
464 response length, after an initial ex-
465 ploratory rise, progressively shortens and
466 stabilizes. This shows the planner learns to
467 balance conciseness and informativeness,
468 avoiding unnecessarily long outputs.

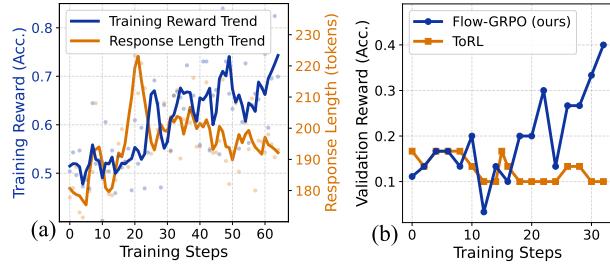


Figure 8: Training dynamics and efficiency of Flow-GRPO.

Flow-GRPO efficiency over tool-integrated reasoning RL. We compare AGENTFLOW (trained with Flow-GRPO) against a monolithic tool-integrated reasoning baseline (ToRL) on AIME24. As shown in Figure 8b, AGENTFLOW achieves sustained performance gains, with validation accuracy growing steadily. In contrast, ToRL’s performance quickly stagnates and trends downwards, highlighting the superior efficiency of our agentic training approach, which uses decomposition and stable credit assignment to avoid the instability.

4.6 SCALING TRENDS IN AGENTFLOW

475 Training scaling in backbone size.

476 We study how backbone LLM scale
477 affects AGENTFLOW’s performance
478 and the efficacy of Flow-GRPO. We
479 build two versions of the system: one
480 using *Qwen2.5-3B-Instruct* and an-
481 other using *Qwen2.5-7B-Instruct* for
482 all four modules (planner, executor,
483 verifier, and generator) and tools. In
484 both, only the planner is fine-tuned
485 with Flow-GRPO. As shown in Figure 9,
486 Flow-GRPO fine-tuning consistently
487 improves performance across tasks for
488 both backbones. This demonstrates
489 that our in-the-flow optimization is
490 effective across model capacities,
491 enhancing AGENTFLOW regardless of
492 LLM size.

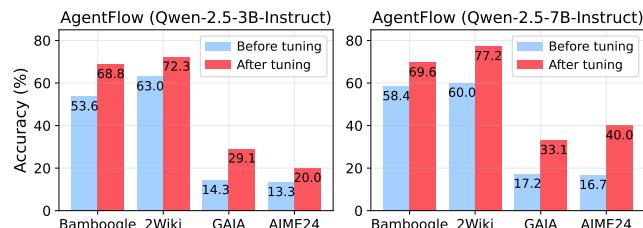


Figure 9: Flow-GRPO fine-tuning offers consistent gains on AGENTFLOW as the backbone model size scales from 3B to 7B.

Figure 9: Flow-GRPO fine-tuning consistently improves performance across tasks for both backbones. This demonstrates that our in-the-flow optimization is effective across model capacities, enhancing AGENTFLOW regardless of LLM size.

Inference scaling in turn budgets. We investigate how the maximum allowed turns (T_{\max}) affect reasoning depth and final performance of AGENTFLOW during test-time inference with the Qwen2.5-7B-Instruct backbone. As shown in Figure 10, increasing T_{\max} from 3 to 10 consistently improves outcomes across all tasks, accompanied by a rise in average turns consumed. On knowledge-intensive benchmarks such as 2Wiki and GAIA, a larger turn budget enables AGENTFLOW for deeper information retrieval. On mathematical benchmarks like GameOf24 and AIME24, it supports decomposed sub-goals, alternative strategies, and refinement of errors. Final performance peaks at $T_{\max} = 10$ for all tasks, confirming that a longer reasoning horizon benefits the system without causing degenerate loops. This validates that AGENTFLOW adapts its turn allocation to problem complexity to achieve better solutions through iterative refinement.

5 RELATED WORK

Reinforcement learning (RL) from outcome-based rewards has become a dominant paradigm for training LLMs to use external tools. Much of this work trains a single, monolithic policy to interleave reasoning with tool calls. This strategy has proven effective in specialized, single-tool settings, such as code execution for mathematical problems (Mai et al., 2025; Xue et al., 2025; Feng et al., 2025; Li et al., 2025b) and web search for knowledge-intensive questions (Chen et al., 2025; Jin et al., 2025; Song et al., 2025; Li et al., 2025a; Sun et al., 2025). Recent efforts have extended this monolithic framework to multi-tool environments by focusing on data synthesis (Dong et al., 2025), unified training infrastructure (Jiang et al., 2025), and principled reward design (Qian et al., 2025a; Zhang et al., 2025). However, this monolithic approach scales poorly as task complexity and planning horizons grow. The central challenge is long-horizon credit assignment; attributing a final outcome to specific intermediate tool calls remains difficult, even with fine-grained, turn-level rewards (Zeng et al., 2025a; Wang et al., 2025d). This difficulty leads to training instability and brittle inference-time generalization, manifesting as strategic deficiencies like tool overuse or “cognitive offloading” (Wang et al., 2025b; Qian et al., 2025b), suboptimal personalization (Cheng et al., 2025), and poor alignment with user preferences for tool invocation (Huang et al., 2025).

Agentic systems with tool use. Agentic systems offer an alternative to monolithic models by decomposing tasks across specialized modules. Many such systems are training-free, orchestrating pre-trained LLMs with handcrafted logic and prompting, as seen in frameworks like AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024), and OctoTools (Lu et al., 2025). This static approach, however, limits their ability to learn and adapt collaborative strategies from experience. Recognizing this, recent work explores training these systems to improve coordination (Deng et al., 2025; Liao et al., 2025). However, most training paradigms are *offline*, relying on supervised fine-tuning or preference optimization on static datasets (Motwani et al., 2024; Park et al., 2025). These methods are decoupled from the live, multi-turn dynamics of the system, preventing modules from learning to adapt to evolving tool outputs or recover from early mistakes. Training directly *in the flow* with on-policy RL is difficult due to sparse rewards and long-horizon credit assignment, where feedback is delayed across long reasoning chains and shifting state distributions (Wang et al., 2025c). Consequently, these systems often suffer from brittle adaptation and require complex reward shaping to learn effectively (Wang et al., 2025a).

6 CONCLUSION

We presented AGENTFLOW, a trainable, *in-the-flow* agentic system that coordinates four specialized modules via an evolving memory and optimizes its planner directly *inside* the multi-turn loop. To enable stable on-policy learning under long-horizon, sparse-reward settings, we introduced Flow-GRPO, which converts multi-turn RL into a sequence of tractable *single-turn* policy updates by broadcasting a single, verifiable trajectory-level outcome to every turn and stabilizing credit assignment with group-normalized advantages. Comprehensive experiments show that AGENTFLOW achieves strong cross-domain performance, surpassing specialized baselines and even larger proprietary models. In-depth analyses confirm improved planning and tool-calling reliability, along with positive scaling trends in model size and allowed turn budgets.

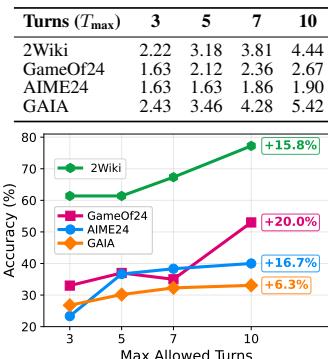


Figure 10: Average turns and accuracy with increased T_{\max} .

540
541
ETHICS STATEMENT

542 We affirm compliance with the ICLR Code of Ethics. Our research exclusively utilizes publicly
 543 available benchmarks, and our methodology does not involve human subjects, personally identi-
 544 fiable information, or proprietary user data. We adhere to the licensing and usage terms of all
 545 datasets employed in this study. The agentic system interacts with external tools, for which we have
 546 implemented safeguards to ensure responsible use. Web-based tools, such as Google Search and
 547 Wikipedia Search, are used solely to access public information while respecting platform terms of
 548 service and API rate limits. All code execution is performed within a sandboxed local environment
 549 with restricted network access to mitigate the security risks of executing model-generated code.

550 We acknowledge two primary ethical considerations. First, the use of an LLM-as-judge for reward
 551 signaling could introduce or amplify biases. To mitigate this, we employ a structured, rubric-based
 552 evaluation protocol, report results averaged over multiple random seeds to ensure robustness, and
 553 conduct detailed analyses of failure modes. Second, advanced agentic systems pose a risk of misuse
 554 in harmful automation. To address this, our work and the released codebase are intentionally focused
 555 on benign research domains (e.g., mathematics, scientific reasoning). We document the intended
 556 scope and limitations to discourage misuse.

557 In the interest of transparency and research integrity, we will release our codebase, model prompts,
 558 and experimental configurations to support reproducibility. The authors declare no conflicts of in-
 559 terest. All funding sources and affiliations will be fully disclosed in the camera-ready version.

560
561
REPRODUCIBILITY STATEMENT

562 To ensure the reproducibility of our work, we provide comprehensive documentation and re-
 563 sources. Our full codebase, including end-to-end scripts for training and evaluation, is available at
 564 <https://anonymous.4open.science/r/agentflow>. This repository contains all con-
 565 figuration files (hyperparameters, model IDs, rollout settings), prompt templates for the planner,
 566 executor, verifier, generator, and memory modules (§E.1), toolset metadata (§E.2), and the LLM-as-
 567 judge evaluation rubric (§E.3). Our experimental setup, including baselines, datasets, and evaluation
 568 protocols, is detailed in §C, with training details provided in §C.1 and evaluation details in §C.2. For
 569 our theoretical contributions, a mathematical analysis of Flow-GRPO, including proofs and conver-
 570 gence guarantees, is presented in §B.

571
572
REFERENCES

573
574 Art of Problem Solving. Aime problems and solutions, 2025. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. 6, 22

575
576
577 Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan,
 578 Wen Zhang, Huajun Chen, Fan Yang, et al. ReSearch: Learning to reason with search for llms via
 579 reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025. 2, 3, 7, 10, 21

580
581 Zihao Cheng, Hongru Wang, Zeming Liu, Yuhang Guo, Yuanfang Guo, Yunhong Wang, and
 582 Haifeng Wang. ToolSpectrum: Towards personalized tool utilization for large language mod-
 583 els. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20679–20699,
 584 2025. 10

585
586 Yingfan Deng, Anhao Zhou, Yuan Yuan, Xian Zhang, Yifei Zou, and Dongxiao Yu. Pe-ma:
 587 Parameter-efficient co-evolution of multi-agent systems. *arXiv preprint arXiv:2506.11803*, 2025.
 10

588
589 Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui
 590 Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner
 591 via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025. 7, 10

592
593 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha
 594 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models.
 595 *arXiv preprint arXiv:2407.21783*, 2024. 7, 20

594 Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang,
 595 Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms.
 596 *arXiv preprint arXiv:2504.11536*, 2025. 2, 3, 10
 597

598 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
 599 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
 600 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 1

601 Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian
 602 Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, de-
 603 contaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint*
 604 *arXiv:2504.11456*, 2025. 6

605 Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop
 606 qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th Interna-*
 607 *tional Conference on Computational Linguistics (COLING)*, pp. 6609–6625, 2020. 6, 22
 608

609 Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin
 610 Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. MetaGPT: Meta programming for a
 611 multi-agent collaborative framework. In *International Conference on Learning Representations*
 612 (*ICLR*), 2024. 2, 3, 10

613 Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum.
 614 Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base
 615 model. *arXiv preprint arXiv:2503.24290*, 2025a. 7, 21
 616

617 Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan
 618 Jin, Yingru Li, Qiguang Chen, et al. Owl: Optimized workforce learning for general multi-agent
 619 assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025b. 2

620 Chengrui Huang, Shen Gao, Zhengliang Shi, Dongsheng Wang, and Shuo Shang. TTPA: Token-
 621 level tool-use preference alignment training framework with fine-grained evaluation. *arXiv*
 622 *preprint arXiv:2505.20016*, 2025. 10

623 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
 624 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint*
 625 *arXiv:2410.21276*, 2024. 3, 7, 20
 626

627 Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai
 628 Zou, Chao Du, et al. VerlTool: Towards holistic agentic reinforcement learning with tool use.
 629 *arXiv preprint arXiv:2509.01055*, 2025. 7, 10, 21

630 Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and
 631 Jiawei Han. Search-R1: Training llms to reason and leverage search engines with reinforcement
 632 learning. *arXiv preprint arXiv:2503.09516*, 2025. 2, 3, 6, 7, 10, 20, 21
 633

634 Di Jin, Eileen Pan, Nassim Oufattolle, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What dis-
 635 ease does this patient have? a large-scale open domain question answering dataset from medical
 636 exams. *Applied Sciences*, 11(14):6421, 2021. 22

637 Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and
 638 Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint*
 639 *arXiv:2501.05366*, 2025a. 10
 640

641 Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling tool-integrated rl. *arXiv preprint*
 642 *arXiv:2503.23383*, 2025b. 7, 10, 21

643 Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-
 644 tuning. *arXiv preprint arXiv:2504.16129*, 2025. 10
 645

646 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan
 647 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth*
 648 *International Conference on Learning Representations (ICLR)*, 2023. 6

648 Nathan Lile. Math twenty four (24s game) dataset. <https://huggingface.co/datasets/nlile/24-game>, 2024. 22

649

650

651 Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. OctoTools: An agentic
652 framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025.
653 3, 10

654

655 Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhui Chen. General-reasoner:
656 Advancing llm reasoning across all domains. *arXiv preprint arXiv:2505.14652*, 2025. 7, 21

657

658 MAA. American mathematics competitions. In *American Mathematics Competitions*, 2023. 6, 22

659

660 Xinji Mai, Haotian Xu, Xing W, Weinong Wang, Yingying Zhang, and Wenqiang Zhang. Agent RL
661 Scaling Law: Agent RL with Spontaneous Code Execution for Mathematical Problem Solving.
662 *arXiv preprint arXiv:2505.07773*, 2025. 2, 10

663

664 Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia:
665 a benchmark for general ai assistants. In *The Twelfth International Conference on Learning
666 Representations (ICLR)*, 2023. 6, 22

667

668 Moonshot AI. Kimi-Researcher: End-to-End RL Training for Emerging Agentic Capabilities.
669 <https://moonshotai.github.io/Kimi-Researcher/>, June 2025. 2

670

671 Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev,
672 Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improv-
673 ing reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024. 2, 5, 10

674

675 Chanwoo Park, Seungju Han, Xingzhi Guo, A. Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. MA-
676 PoRL: Multi-agent post-co-training for collaborative large language models with reinforcement
677 learning. In *Annual Meeting of the Association for Computational Linguistics (ACL*, 2025. URL
678 <https://api.semanticscholar.org/CorpusId:276580906>. 2, 5, 10

679

680 Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring
681 and narrowing the compositionality gap in language models. In *Findings of the Association for
682 Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023. 6, 22

683

684 Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan
685 Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*,
686 2025a. 2, 3, 10

687

688 Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür,
689 Gokhan Tur, and Heng Ji. SMART: Self-aware agent for tool overuse mitigation. In *Findings
690 of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025b. 10

691

692 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Di-
693 rani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a bench-
694 mark. In *First Conference on Language Modeling*, 2024. 6, 22

695

696 John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region
697 policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897.
698 PMLR, 2015. 19

699

700 Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing
701 retrieval-augmented large language models with iterative retrieval-generation synergy. In *Find-
702 ings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9248–9274, 2023. 7,
703 21

704

705 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
706 Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathemati-
707 cal reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 3

708

709 Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang,
710 and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement
711 learning. *arXiv preprint arXiv:2503.05592*, 2025. 2, 10

702 Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang,
 703 Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without
 704 searching. *arXiv preprint arXiv:2505.04588*, 2025. 7, 10, 21

705 Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop
 706 questions via single-hop question composition. *Transactions of the Association for Computational
 707 Linguistics (TACL)*, 10:539–554, 2022. 6, 22

708 Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. SPA-RL: Reinforcing
 709 llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*, 2025a. 10

710 Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin,
 711 Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to
 712 act efficiently. *arXiv preprint arXiv:2504.14870*, 2025b. URL <https://arxiv.org/pdf/2504.14870.pdf>. 10

713 Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin,
 714 Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding self-evolution in llm
 715 agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025c. 2, 10

716 Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu.
 717 Stepsearch: Igniting llms search ability via step-wise proximal policy optimization. *arXiv preprint
 718 arXiv:2505.15107*, 2025d. 7, 10, 21

719 Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun
 720 Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-
 721 agent conversations. In *First Conference on Language Modeling (COLM)*, 2024. 2, 3, 7, 10,
 722 21

723 Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Sim-
 724 pletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint
 725 arXiv:2509.02479*, 2025. 2, 10

726 Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang.
 727 Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*, 2025. 7, 21

728 An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li,
 729 Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin
 730 Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang,
 731 Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang,
 732 Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan,
 733 Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint
 734 arXiv:2412.15115*, 2024a. 6, 7, 20

735 An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jian-
 736 hong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2.5-math technical report: Toward mathematical
 737 expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024b. 7, 21

738 Hang Yang, Hao Chen, Hui Guo, Yineng Chen, Ching-Sheng Lin, Shu Hu, Jinrong Hu, Xi Wu,
 739 and Xin Wang. Llm-medqa: Enhancing medical question answering through case studies in large
 740 language models. *arXiv preprint arXiv:2501.05464*, 2024c. 6

741 Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov,
 742 and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question
 743 answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language
 744 Processing (EMNLP)*, pp. 2369–2380, 2018. 6, 22

745 Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian
 746 Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system
 747 at scale. *arXiv preprint arXiv:2503.14476*, 2025. 3

748 Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. Re-
 749 inforging multi-turn reasoning in llm agents via turn-level credit assignment. *arXiv preprint
 750 arXiv:2505.11821*, 2025a. 10

756 Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-
757 zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv*
758 *preprint arXiv:2503.18892*, 2025b. 1, 7, 20, 21

759

760 Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu,
761 Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Tool-using language models with rein-
762 forced reasoning. *arXiv preprint arXiv:2505.00024*, 2025. 2, 10

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810 TABLE OF CONTENTS
811

813	A Training Algorithm of AGENTFLOW	17
814		
815	B Theoretical Analysis of Flow-GRPO	18
816	B.1 Preliminaries and Notation	18
817	B.2 Equivalence Proof for Optimization Objectives	18
818	B.3 Convergence Analysis	19
819		
820	C Experimental Details	20
821		
822	C.1 Training Details	20
823	C.2 Evaluation Details	20
824	C.3 Compared Baselines	20
825	C.4 Evaluation Datasets	21
826		
827	D More Discussion about Experiment Results	23
828		
829	D.1 Main Result Analysis	23
830	D.2 In-depth Analysis of Optimized Planning	24
831		
832	E Instruction Templates in AGENTFLOW	25
833		
834	E.1 Modules and Memory	25
835	E.2 Toolset Metadata	30
836	E.3 LLM-based Judging	35
837		
838	F Case Studies	36
839		
840	F.1 Example 1: Efficient Search for Simple Tasks	36
841	F.2 Example 2: Spontaneous Brute-force	37
842	F.3 Example 3: A Good Initial Plan is Essential	39
843	F.4 Example 4: Robust Self-Correction and Adaptation	41
844	F.5 Example 5: New Combo: Retrieve with Specific URL	43
845	F.6 Example 6: Rapid and Correct Physics Calculation	45
846	F.7 Example 7: Multi-Source Cross-Verification	47
847		
848	G LLM Usage Statement	49
849		
850		
851		
852		
853		
854		
855		
856		
857		
858		
859		
860		
861		
862		
863		

864 A TRAINING ALGORITHM OF AGENTFLOW
865866 We provide a flowchart of the overall training algorithm of AGENTFLOW (§3) in Algorithm 1.
867868 **Algorithm 1** In-the-Flow Optimization for AGENTFLOW
869

Require: Dataset \mathcal{D} , Action Planner policy π_θ , Tool Executor \mathcal{E} , Executive Verifier \mathcal{V} , Solution Generator \mathcal{G} , Toolset K , and Shared Evolving Memory M
Ensure: Optimized Action Planner parameters θ^*

```

1: for each training iteration do
2:   for each query-label pair  $(q, y^*) \sim \mathcal{D}$  do
3:     1. IN-THE-FLOW ROLLOUT GENERATION
4:     Initialize:  $t \leftarrow 1, M^t \leftarrow q$ 
5:     repeat
6:        $a^t \sim \pi_\theta(a^t | q, K, M^t)$  {Plan Action}
7:        $e^t \sim \mathcal{E}(e^t | a^t, K)$  {Execute Action}
8:        $v^t \sim \mathcal{V}(v^t | q, e^t, M^t)$  {Verify Result}
9:        $M^{t+1} = f_{\text{mem}}(M^t, a^t, e^t, v^t)$  {Update Memory}
10:       $t \leftarrow t + 1$ 
11:    until termination condition met
12:     $o \sim \mathcal{G}(o | q, M^T)$  {Generate Final Solution}
13:    2. REWARD COMPUTATION
14:     $R(a^t) = \bar{R}(o, q, y^*), \quad \forall t = 1, \dots, T$ 
15:    3. POLICY UPDATE
16:    Update the Action Planner policy  $\pi_\theta$  by maximizing the Flow-GRPO objective (Eq. 5)
17:  end for
18: end for
19: return optimized parameters  $\theta^*$ 

```

890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

918 B THEORETICAL ANALYSIS OF FLOW-GRPO

919 B.1 PRELIMINARIES AND NOTATION

920 We adopt the notation from the paper to formalize our analysis.

921 **Definition B.1** (Core Components). Here we list core definition of variables.

922 Symbol and Description

π_θ	The trainable planner policy, parameterized by θ .
$\pi_{\theta_{\text{old}}}$	The behavior policy used to sample trajectories.
s^t	The state at turn t , defined as $s^t = (q, K, M_t)$.
a^t	The action (a sequence of tokens) generated at state s^t , where $a^t \sim \pi_\theta(\cdot s^t)$.
τ	A trajectory of states and actions over T time steps, defined as $\tau = \{(s^t, a^t)\}_{t=1}^T$.
$R(\tau)$	The outcome-based reward for trajectory τ , where $R(\tau) \in \{0, 1\}$.
A_τ	The group-normalized advantage for trajectory τ . A crucial property is that the advantage is constant for all timesteps within a trajectory defined in Eq. 7: $a^t = A_\tau, \forall (s^t, a^t) \in \tau$.
$\rho_{i,j}^t$	The token-level importance sampling ratio, defined as:

$$923 \quad \rho_{i,j}^t = \frac{\pi_\theta(a_{i,j}^t | s_i^t, a_{i,1:j-1}^t)}{\pi_{\theta_{\text{old}}}(a_{i,j}^t | s_i^t, a_{i,1:j-1}^t)}.$$

924 $L_{\text{clip}}(\rho, A)$ The PPO clipped objective term, defined as $L_{\text{clip}}(\rho, A) = \min(\rho A, \text{clip}(\rho, 1 - \epsilon, 1 + \epsilon) A)$.

925 **Definition B.2** (Objective Functions). The *global policy objective* is the expected trajectory-level
926 reward:

$$927 \quad \mathcal{J}(\theta) := \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]. \quad (8)$$

928 The *single-turn optimization objective* for a given state s^t is defined as:

$$929 \quad \mathcal{J}_{\text{local}}(\theta; s^t) := \mathbb{E}_{a^t \sim \pi_{\theta_{\text{old}}}(\cdot | s^t)} \left[\frac{1}{|a^t|} \sum_{j=1}^{|a^t|} L_{\text{clip}}(\rho_{i,j}^t, A_i^t) \right]. \quad (9)$$

930 The full Flow-GRPO objective function in the multi-turn setting is given by:

$$931 \quad \mathcal{J}_{\text{Flow-GRPO}}(\theta) := \mathbb{E}_{\substack{(q, y^*) \sim \mathcal{D} \\ \{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{T_i} \sum_{t=1}^{T_i} \frac{1}{|a_i^t|} \sum_{j=1}^{|a_i^t|} L_{\text{clip}}(\rho_{i,j}^t, A_i^t) \right] - \beta \mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}). \quad (10)$$

932 B.2 EQUIVALENCE PROOF FOR OPTIMIZATION OBJECTIVES

933 **Theorem B.1.** *In Flow-GRPO, maximizing the global multi-turn objective is mathematically equivalent to maximizing the expected token-level local objective at each time step under the on-policy induced state distribution, given standard sampling assumptions (trajectories sampled i.i.d. from the policy with fixed finite turn T).*

934 *Proof.* Let's denote the clipping part of the Flow-GRPO objective as $\mathcal{J}_{\text{clip}}(\theta)$.

935 First, by the linearity of expectation, we can simplify the expectation over a group of G trajectories.
936 Since the trajectories $\{\tau_i\}$ are sampled independently and identically (i.i.d.) from the behavior policy
937 $\pi_{\theta_{\text{old}}}$, the expectation of their average is equal to the expectation over a single trajectory.

$$938 \quad \mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q, y^*) \sim \mathcal{D}} \left[\mathbb{E}_{\{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{T_i} \sum_{t=1}^{T_i} \left(\frac{1}{|a_i^t|} \sum_{j=1}^{|a_i^t|} L_{\text{clip}}(\rho_{i,j}^t, A_i^t) \right) \right] \right] \quad (11)$$

$$939 \quad = \mathbb{E}_{(q, y^*) \sim \mathcal{D}} \left[\mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}(\cdot | q)} \left[\frac{1}{T} \sum_{t=1}^T \left(\frac{1}{|a^t|} \sum_{j=1}^{|a^t|} L_{\text{clip}}(\rho_j^t, A_\tau) \right) \right] \right]. \quad (12)$$

972 Here, $\tau = \{(s^t, a^t)\}_{t=1}^T$ represents a single, arbitrarily sampled trajectory with advantage A_τ .
 973

974 Next, we can re-interpret the expectation over trajectories as an expectation over the state-visitation
 975 distribution induced by the policy $\pi_{\theta_{\text{old}}}$. Let $d^{\pi_{\theta_{\text{old}}}}$ be the on-policy distribution of states visited,
 976 where each state s^t in a trajectory of length T is weighted by $1/T$. The expectation can be rewritten
 977 as:

$$978 \quad \mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q, y^*) \sim \mathcal{D}} \left[\mathbb{E}_{s^t \sim d^{\pi_{\theta_{\text{old}}}}} \left[\mathbb{E}_{a^t \sim \pi_{\theta_{\text{old}}}(\cdot | s^t)} \left[\frac{1}{|a^t|} \sum_{j=1}^{|a^t|} L_{\text{clip}}(\rho_j^t, A^t) \right] \right] \right]. \quad (13)$$

981 Note that A^t is the advantage corresponding to the trajectory from which s^t was sampled.
 982

983 We now recognize that the inner expectation is precisely the definition of the local, per-state objec-
 984 tive, $\mathcal{J}_{\text{local}}(\theta; s^t)$.

$$985 \quad \mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q, y^*) \sim \mathcal{D}, s^t \sim d^{\pi_{\theta_{\text{old}}}}} [\mathcal{J}_{\text{local}}(\theta; s^t)]. \quad (14)$$

987 Adding the KL-divergence term back, we arrive at the final equivalence:

$$988 \quad \mathcal{J}_{\text{Flow-GRPO}}(\theta) = \mathbb{E}_{(q, y^*) \sim \mathcal{D}, s^t \sim d^{\pi_{\theta_{\text{old}}}}} [\mathcal{J}_{\text{local}}(\theta; s^t)] - \beta \mathbb{D}_{KL}(\pi_\theta \| \pi_{\text{ref}}). \quad (15)$$

990 This proves that maximizing the global multi-turn Flow-GRPO objective is equivalent to maximiz-
 991 ing the expected token-level local objective at each time step under the on-policy induced state
 992 distribution. \square

994 B.3 CONVERGENCE ANALYSIS

996 Having established the structural validity of the objective, we now analyze its convergence proper-
 997 ties. The analysis builds on the monotonic improvement guarantee provided by trust-region meth-
 998 ods (Schulman et al., 2015).

999 **Lemma B.2** (Policy Performance Difference). *For two policies π_θ and $\pi_{\theta_{\text{old}}}$, the difference in ex-
 1000 pected return can be expressed as:*

$$1001 \quad \mathcal{J}(\theta) - \mathcal{J}(\theta_{\text{old}}) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^T A_{\theta_{\text{old}}}(s^t, a^t) \right], \quad (16)$$

1004 where $A_{\theta_{\text{old}}}$ is the advantage function under the old policy.
 1005

1006 This lemma enables the construction of a lower bound on policy improvement.

1007 **Theorem B.3** (Monotonic Improvement Guarantee). *Define the surrogate objective*

$$1009 \quad \mathcal{L}_{\theta_{\text{old}}}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}} \left[\sum_{t=1}^T \frac{\pi_\theta(a^t | s^t)}{\pi_{\theta_{\text{old}}}(a^t | s^t)} A_{\theta_{\text{old}}}(s^t, a^t) \right]. \quad (17)$$

1012 Then the performance improvement satisfies the lower bound

$$1013 \quad \mathcal{J}(\theta) - \mathcal{J}(\theta_{\text{old}}) \geq \mathcal{L}_{\theta_{\text{old}}}(\theta) - C \cdot \bar{\mathbb{D}}_{KL}(\pi_{\theta_{\text{old}}}, \pi_\theta), \quad (18)$$

1015 where $C > 0$ is a constant depending on the horizon and reward scale, and $\bar{\mathbb{D}}_{KL}$ denotes the average
 1016 KL-divergence between the two policies.

1017 By optimizing the right-hand side of the above inequality, we are guaranteed to improve the perfor-
 1018 mance of π_θ . Therefore, for policies π_θ^t and π_θ^{t+1} obtained from iterations t and $t+1$, we have:
 1019

$$1020 \quad \mathcal{J}(\theta^{t+1}) \geq \mathcal{J}(\theta^t). \quad (19)$$

1022 **Conclusion.** This analysis establishes that Flow-GRPO optimizes a valid surrogate objective and
 1023 guarantees monotonic policy improvement, thereby converging reliably to a locally optimal policy.

1026 **C EXPERIMENTAL DETAILS**
10271028 **C.1 TRAINING DETAILS**
10291030 We provide further details on the training setup for AGENTFLOW. Our Flow-GRPO implementation
1031 uses a learning rate of 1×10^{-6} . The Action Planner generates actions with a sampling temperature
1032 of 0.5 to balance exploration and exploitation. To prevent policy collapse and stabilize training, we
1033 incorporate a KL-divergence penalty against a reference policy with a coefficient $\beta = 0.001$. The
1034 maximum output length for the planner is set to 2048 tokens to ensure complete exploration during
1035 rollouts.1036 To accelerate the training speed, we limit the maximum number of turns per rollout to 3. The final-
1037 outcome reward signal (Eq. 4) is provided by an LLM-as-judge, for which we use *GPT-4o*. All
1038 tool calls are executed synchronously with a 500-second timeout to handle external service latency
1039 robustly. The LLM engines within the tools are set to a temperature of 0.0 to ensure deterministic
1040 and stable outputs. The full training process was conducted on 8 NVIDIA A100 GPUs. Further
1041 details on agent prompts and the memory update mechanism are provided in §E.1.
10421043 **C.2 EVALUATION DETAILS**
10441045 Here, we outline the specifics of our evaluation protocol. For evaluation, we increase the maximum
1046 number of turns per rollout to $T = 10$ to allow for more extensive and deeper reasoning. The
1047 planner’s sampling temperature is set to 0.7 to encourage diverse solution paths. Unless otherwise
1048 specified, all tool LLM engines are initialized with Qwen2.5-7B-Instruct.
10491050 For fair and consistent evaluation, we adopt the previous work’s methodology while standardizing
1051 tools: we replace search tools in search-enhanced models with our Google Search tool and code
1052 tools in code-enhanced models with our Python Coder tool. We use GPT-4o as an LLM-based judge
1053 to determine the correctness of final answers. This approach provides a robust measure of semantic
1054 and numerical equivalence, which is critical for complex reasoning tasks. The specific judging
1055 prompt is detailed in §E.3, and additional information on evaluation datasets can be found in §C.4.
1056 To mitigate randomness, we report the average accuracy with standard deviation across three trials
1057 for all experiments.
10581059 **C.3 COMPARED BASELINES**
10601061 **Proprietary LLMs:**
10621063

- **Qwen2.5 Series** (Yang et al., 2024a), created by Alibaba, comes in multiple configurations. These
1064 models undergo training on multilingual corpora covering 29 different languages, demonstrating
1065 superior performance in cross-lingual applications. Furthermore, Qwen2.5 showcases robust pro-
1066 ficiency in programming and mathematical domains.
- **Llama-3 Series** (Dubey et al., 2024), created by Meta AI, encompasses various iterations.
1067 Each model configuration within the Llama family provides dual versions: foundational and
1068 instruction-following variants. Training incorporates diverse dataset combinations spanning mul-
1069 tiple domains and linguistic varieties. The Llama model family demonstrates excellent results in
1070 logical reasoning, software development, and cross-lingual comprehension evaluations. Through
1071 progressive enhancements in fine-tuning methodologies and expanded sequence lengths, these
1072 models become more applicable to practical deployment scenarios.
- **GPT-4o Series** (Hurst et al., 2024), produced by OpenAI, includes several model variants such
1073 as GPT-4o and GPT-4o-mini, with training leveraging extensive multimodal datasets encompass-
1074 ing text, vision, and audio modalities. The series achieves outstanding performance in complex
1075 reasoning tasks, creative generation, and multimodal understanding benchmarks with continuous
1076 refinements in alignment techniques and enhanced processing capabilities.

1077 **Reasoning LLMs:**
10781079

- **SFT** (Zeng et al., 2025b) serves as our basic baseline following Search-R1 (Jin et al., 2025). We
1080 fine-tune models using supervised fine-tuning on GPT-4o-generated reasoning chains.

- 1080 • **SimpleRL-Zoo** (Zeng et al., 2025b) investigates zero reinforcement learning training across 10
1081 diverse base models spanning different families and sizes using GRPO algorithm with simple
1082 rule-based rewards, achieving substantial improvements in reasoning accuracy.
- 1083 • **Open-Reasoner-Zero** (Hu et al., 2025a) presents the first open-source implementation of large-
1084 scale reasoning-oriented RL training using PPO with GAE and straightforward rule-based re-
1085 wards, without KL regularization. The framework demonstrates that minimalist design can suc-
1086 cessfully scale both response length and benchmark performance.
- 1087 • **General-Reasoner** (Ma et al., 2025) extends LLM reasoning capabilities beyond mathematics
1088 to diverse domains using RLVR through a 230K verifiable reasoning questions dataset spanning
1089 physics, chemistry, and finance.
- 1090 • **LUFFY** (Yan et al., 2025) addresses limitations in on-policy RLVR by introducing an off-policy
1091 framework that augments training with external reasoning demonstrations using Mixed Policy
1092 GRPO and regularized importance sampling.

1094 **Search-Integrated Reasoning LLMs:**

- 1095 • **Iter-RetGen** (Shao et al., 2023) addresses limitations in retrieval-augmented language models by
1096 introducing iterative retrieval-generation synergy, where a model’s previous response serves as
1097 context for retrieving more relevant knowledge in subsequent iterations.
- 1098 • **Search-R1** (Jin et al., 2025) represents a reinforcement learning approach that develops a model
1099 from the ground up to invoke search functionality throughout the reasoning process.
- 1100 • **ZeroSearch** (Sun et al., 2025) addresses high API costs in RL-based search training by using an
1101 LLM to simulate search engines, employing lightweight supervised fine-tuning to transform an
1102 LLM into a retrieval module that generates both useful and noisy documents. The framework
1103 combines this with a curriculum-based rollout strategy that progressively degrades document
1104 quality, achieving better performance than real search engine-based methods while incurring zero
1105 API costs.
- 1106 • **ReSearch** (Chen et al., 2025) proposes a reinforcement learning framework that trains LLMs
1107 to integrate search operations as components of the reasoning chain without supervised data on
1108 reasoning steps, treating search decisions as guided by text-based thinking.
- 1109 • **StepSearch** (Wang et al., 2025d) addresses the sparse reward problem in multi-hop reasoning
1110 by training search LLMs using step-wise proximal policy optimization with intermediate rewards
1111 and token-level process supervision based on information gain and redundancy penalties.
- 1112 • **VerlTool** (Jiang et al., 2025) addresses fragmentation and synchronization bottlenecks in Agentic
1113 Reinforcement Learning with Tool use by introducing a unified modular framework that extends
1114 beyond single-turn RLVR paradigms, providing upstream VeRL alignment and unified tool man-
1115 agement with asynchronous rollout execution achieving near 2 \times speedup.

1117 **Code-Integrated Reasoning LLMs:**

- 1118 • **TIR** (Yang et al., 2024b) is a basic baseline that demonstrates the model’s ability to generate code
1119 for tool utilization. In our implementation, we directly prompt the model to write code that calls
1120 the programming interpreter and processes the returned results to generate the final answer.
- 1121 • **ToRL** (Li et al., 2025b) is a code-enhanced architecture developed via reinforcement learning
1122 that empowers models to independently activate code execution environments for mathematical
1123 reasoning tasks.

1125 **Training-free Agentic System**

- 1126 • **AutoGen** (Wu et al., 2024) introduces an agentic conversation framework that enables developers
1127 to build LLM applications through conversable agents that can operate using combinations of
1128 LLMs, human inputs, and tools.

1131 **C.4 EVALUATION DATASETS**

1132 We provide a detailed introduction to the *search-intensive* and *agentic* benchmarks in our experi-
1133 ments as follows:

- **Bamboogle** (Press et al., 2023) presents a demanding multi-step reasoning dataset containing manually constructed questions requiring up to four inferential steps. The dataset evaluates models’ capacity for intricate compositional reasoning across interconnected facts.
- **2Wiki (2WikiMultihopQA)** (Ho et al., 2020) constitutes a comprehensive multi-step QA corpus combining structured Wikidata knowledge with unstructured Wikipedia text. The dataset encompasses varied question formats and annotated reasoning chains to facilitate interpretable sequential inference. We randomly sample 100 examples as a test set for efficiency.
- **HotpotQA** (Yang et al., 2018) represents a widely-adopted question answering corpus featuring multi-step queries constructed from Wikipedia entries. We randomly sample 100 examples as a test set for efficiency.
- **Musique** (Trivedi et al., 2022) comprises a multi-step reasoning corpus requiring sequential inference where each reasoning stage depends on information derived from preceding steps. We conduct evaluations using the development partition of this particularly challenging dataset. We randomly sample 100 examples as a test set for efficiency.
- **GAIA** (Mialon et al., 2023) constitutes a benchmark engineered to assess general AI systems and agents, demanding capabilities including sequential reasoning, web navigation, and comprehensive tool utilization skills. We utilize the text-exclusive portion of this dataset, designed to challenge base language models in our experimental setup.

Furthermore, we also conduct a series of experiments on *math* and *scientific reasoning* benchmarks:

- **AIME24** (Art of Problem Solving, 2025) A collection of 30 demanding mathematical problems sourced from the 2024 American Invitational Mathematics Examination (AIME), encompassing algebra, geometry, number theory, and combinatorics. Each JSONL-formatted record contains the problem identifier, question text, comprehensive solution methodology, and the final numerical result. Created to assess large language models’ sophisticated mathematical reasoning abilities, the dataset presents substantial difficulty, systematic multi-phase solutions, and distinctive answers—establishing it as a robust benchmark for evaluating advanced analytical capabilities.
- **AMC23** (MAA, 2023) contains mathematical problems derived from the 2023 American Mathematics Competition, emphasizing areas such as functional equations and complex analysis.
- **GameOf24** (Lile, 2024) derives from the traditional numerical puzzle known as 24 (alternatively called the 24 numbers game). The challenge requires utilizing four given numbers with fundamental arithmetic operations (addition, subtraction, multiplication, division) to create an expression yielding 24. For instance, with numbers 4, 9, 10, and 13, a correct solution would be “ $(10 - 4) \times (13 - 9) = 24$ ”. Successfully solving requires computational proficiency along with iterative attempts to validate potential solutions. Each challenge is formatted as open-ended inquiries.
- **GPQA** or Graduate Level Google-Proof Q&A Benchmark (Rein et al., 2024) comprises a collection of demanding text-based multiple choice problems authored by subject specialists in biology, physics, and chemistry, intentionally crafted to be “exceptionally challenging”. We randomly sample 100 examples as a test set for efficiency.
- **MedQA** (Jin et al., 2021) features text-based multiple choice problems assembled from professional medical licensing examinations. Problems encompass comprehensive medical knowledge and clinical reasoning skills.

1188 D MORE DISCUSSION ABOUT EXPERIMENT RESULTS

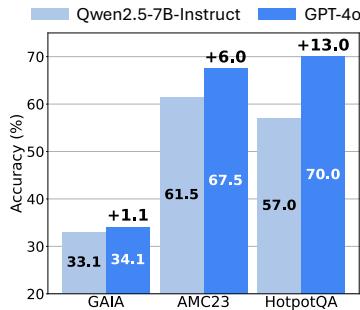
1190 D.1 MAIN RESULT ANALYSIS

1192 Our main results are presented in Tables 1 and 2. Overall, AGENTFLOW consistently outperforms all
 1193 baseline models across diverse domains, including search-intensive tasks, agentic tasks, and mathematical
 1194 and scientific reasoning tasks. These comprehensive results yield several key insights:

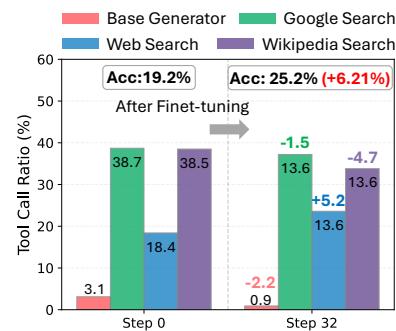
1195 **Monolithic LLMs are insufficient for complex reasoning.** While scaling up model size (from 7B
 1196 model to GPT-4o) improves average performance, their monolithic nature presents limitations when
 1197 facing complex tasks that require multi-turn reasoning and sub-goal decomposition. In contrast, our
 1198 proposed AGENTFLOW consistently outperforms these larger models. Specifically, it achieves an
 1199 average improvement of 8.2% over GPT-4o on search-intensive tasks (57.3% vs. 49.1% in Table 1),
 1200 and a remarkable 15.8% gain over GPT-4o on agentic tasks (33.1% vs. 17.3% in Table 1). For
 1201 mathematical reasoning benchmarks, AGENTFLOW obtains a substantial improvement of 16.4%
 1202 over GPT-4o (51.5% vs. 35.1% in Table 2). Furthermore, it surpasses the strong Llama-3.3-70B
 1203 by 12.5% on scientific reasoning tasks (63.5% vs. 51.0% in Table 2). These results demonstrate
 1204 that the carefully designed agentic system of AGENTFLOW, despite being built on a 7B-parameter
 1205 backbone, can deliver superior and more efficient performance compared to substantially larger
 1206 monolithic LLMs.

1207 **Specialized reasoning models exhibit strong in-domain focus but limited generalizability.** While domain-specific fine-tuning and tailored tool integration provide clear benefits over base
 1208 LLMs, they fail to deliver robust cross-domain performance due to fundamental scaling limitations.
 1209 Our evaluation across three reasoning domains substantiates these limitations. On search-intensive
 1210 tasks, specialized models such as Search-R1 (33.3%) and VerlTool (39.0%) perform well within
 1211 their narrow scope yet fall substantially short of AGENTFLOW (57.3%) as shown in Table 1. Simi-
 1212 larly, in mathematical reasoning, methods like SimpleRL-reason (36.6%) and ToRL (37.0%) trail
 1213 significantly behind AGENTFLOW (51.5%) in Table 2. Even in scientific reasoning, where models
 1214 such as Luffy (55.5%) offer competitive results, they are consistently surpassed by AGENTFLOW
 1215 (63.5%) in Table 2. These findings demonstrate that while specialized reasoning models excel within
 1216 narrow domains, their reliance on a single monolithic policy introduces poor generalization, making
 1217 them brittle when confronted with diverse, cross-domain challenges.

1218 **AGENTFLOW demonstrates superior, versatile reasoning through its adaptive agentic system.**
 1219 AGENTFLOW establishes a new state-of-the-art agentic system by achieving an average accuracy
 1220 of 57.3% on search-intensive tasks, 33.1% on agentic tasks, 51.5% on mathematical reasoning, and
 1221 63.5% on scientific reasoning. Our method’s advantage stems from combining an agentic system
 1222 with targeted planning policy refinement via on-policy reinforcement learning in an online fash-
 1223 ion. When compared to AutoGen—a general agent framework with the same backbone model—
 1224 AGENTFLOW demonstrates a massive improvement of 14.9% on search tasks and 19.9% on math
 1225 tasks. This underscores that the core advantage comes from our dedicated trainable agentic system
 1226 that integrates our novel Flow-GRPO for in-system on-policy optimization, enabling effective agent
 1227 planning and tool utilization to solve complex, long-horizon problems across diverse domains.



1239 Figure 11: **Tool scaling study.** AGENTFLOW’s
 1240 performance improves when its tools are up-
 1241 graded from Qwen-2.5-7B-Instruct to GPT-4o.



1239 Figure 12: **Tool call optimization on Musique.**
 1240 AGENTFLOW’s planner increases Web Search
 1241 usage after Flow-GRPO training.

1242
1243

D.2 IN-DEPTH ANALYSIS OF OPTIMIZED PLANNING

1244
1245
1246
1247
1248
1249
1250

AGENTFLOW adapts to inference-time tool scaling. We scale the tools—the Base Generator and Python Coder—to GPT-4o-powered versions. Empirical results on search and math datasets (Figure 11) show that AGENTFLOW, when using these GPT-4o-powered tools, substantially outperforms its performance with Qwen2.5-7B-Instruct-powered tools, achieving improvements of 1.0% on GAIA, 6.0% on AMC23, and a notable 13.0% on HotpotQA. This finding further supports a consistent trend: after in-the-flow RL training, the planner can adaptively leverage improvements in the underlying tools to enhance the agentic system’s overall performance.

FIX

1251
1252
1253
1254
1255

Flow-GRPO spontaneous tool usage preference change. We further compare tool usage distributions before and after in-the-flow RL training on Musique. Figure 12 shows that due to Musique’s need for a diverse source of information, Flow-GRPO optimizes the planner to increase Web Search to delve deeper into the URL provided by other search tools. This maneuver presents a steady performance improvement of 6.1%.

FIX

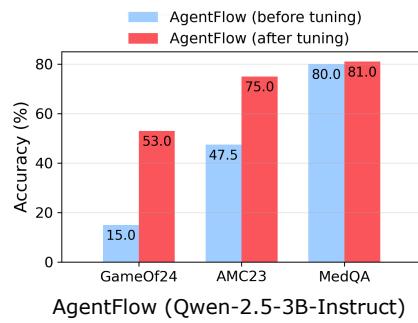
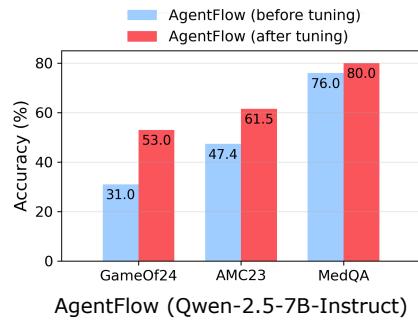
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266

Figure 13: Flow-GRPO fine-tuning offers consistent gains on AGENTFLOW as the backbone model size scales from 3B to 7B.

1267

More evidence of training scaling in backbone size. We further investigate how the backbone LLM scale affects AGENTFLOW’s performance and the efficacy of Flow-GRPO on GameOf24, AMC23, and MedQA. We construct two versions of the system: one using *Qwen2.5-3B-Instruct* and another using *Qwen2.5-7B-Instruct* for all four modules (planner, executor, verifier, and generator) as well as the associated tools. In both versions, only the planner is fine-tuned with Flow-GRPO. As shown in Figure 13, Flow-GRPO fine-tuning consistently improves performance across tasks for both backbones. These results demonstrate that our in-the-flow optimization is effective across model capacities, enhancing AGENTFLOW regardless of LLM size.

1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

1296 E INSTRUCTION TEMPLATES IN AGENTFLOW
12971298 E.1 MODULES AND MEMORY
12991300 E.1.1 ACTION PLANNER
13011302 Tool Metadata can be found in §[E.2](#).1303 **Instruction for Action Planner**
13041305 **Task:** Determine the optimal next step to address the query using available tools and previous context.
13061307 **Context:**
13081309 Query: {Question}
1310 Available Tools: [Base Generator, Python Coder, Google Search, Wikipedia Search, Web Search]
1311 Toolbox Metadata: [Tool Metadata1, Tool Metadata2, ...]
1312 Previous Steps: {Actions from Memory}1313 **Instructions:**
13141315 1. Analyze the current objective, the history of executed steps, and the capabilities of the available tools.
1316 2. Select the single most appropriate tool for the next action.
1317 3. Consider the specificity of the task (e.g., calculation vs. information retrieval).
1318 4. Consider the source of required information (e.g., general knowledge, mathematical computation, a
1319 specific URL).
1320 5. Consider the limitations of each tool as defined in the metadata.
1321 6. Formulate a clear, concise, and achievable sub-goal that precisely defines what the selected tool should
1322 accomplish.
1323 7. Provide all necessary context (e.g., relevant data, variable names, file paths, or URLs) so the tool can
1324 execute its task without ambiguity.
13251326 **Response Format:**
13271328 1. Justification: Explain why the chosen tool is optimal for the sub-goal, referencing its capabilities and
1329 the task requirements.
1330 2. Context: Provide all prerequisite information for the tool.
1331 3. Sub-Goal: State the exact objective for the tool.
1332 4. Tool Name: State the exact name of the selected tool (e.g., Wikipedia Search).
13331334 **Rules:**
13351336 Select only one tool per step.
1337 The Sub-Goal must be directly and solely achievable by the selected tool.
1338 The Context section must contain all information the tool needs; do not assume implicit knowledge.
1339 The final response must end with the Context, Sub-Goal, and Tool Name sections in that order. No
1340 additional text should follow.
13411342 **FIX**
1343

1350 E.1.2 TOOL EXECUTOR

1351

1352 **Instruction for Tool Executor**

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

Task: Generate a precise command to execute the selected tool.**Context:**

Query: {Question}

Sub-Goal: {Sub Goal from Next Step Plan}

Tool Name: {Selected Tool from Next Step Plan}

Toolbox Metadata: {Selected Tool Metadata from Next Step Plan}

Relevant Data: {Context from Next Step Plan}

Instructions:

1. Analyze the tool's required parameters from its metadata.
2. Construct valid Python code that addresses the sub-goal using the provided context and data.
3. The command must include at least one call to `tool.execute()`.
4. Each `tool.execute()` call must be assigned to a variable named `execution`.
5. Use exact numbers, strings, and parameters in the `tool.execute()` call based on the context.

Output Format: Present your response in the following structured format. Do not include any extra text or explanations.**Example 1:**

Generated Command:

```
execution = tool.execute(query="Summarize the following porblom: "Isaac has
100 toys, masa gets ...., how much are their together?")
```

Example 2:

Generated Command:

```
execution = tool.execute(query=["Methanol", "function of hyperbola",
"Fermat's Last Theorem"])
```

FIX

1404 E.1.3 EXECUTION VERIFIER
14051406 **Instruction for Execution Verifier**
14071408 **Task:** Evaluate if the current memory is complete and accurate enough to answer the query, or if more
1409 tools are needed.
14101411 **Context:**
14121413 Query: {Question}
1414 Available Tools: [Base Generator, Python Coder, Google Search, Wikipedia Search, Web Search]
1415 Toolbox Metadata: [Tool Metadata1, Tool Metadata2, ...]
1416 Memory (Tools Used & Results): {Actions from Memory}1417 **Instructions:**
14181419 1. Review the original query, the initial analysis, and the complete history of actions and results in the
1420 memory.
1421 2. Does the accumulated information fully address all aspects of the query?
1422 3. Are there any unanswered sub-questions or missing pieces of information?
1423 4. Are there any inconsistencies or contradictions between different steps?
1424 5. Is any information ambiguous, potentially hallucinated, or in need of verification?
1425 6. Determine if any unused tools could provide critical missing information based on their metadata.
14261427 **Final Determination:**
14281429 If the memory is sufficient to form a complete and accurate answer, explain why and conclude with
1430 “Conclusion: STOP”.
1431 If more information is needed, clearly state what is missing, suggest which tool(s) could help, and con-
1432 clude with “Conclusion: CONTINUE”.
14331434 **Rules:**
14351436 The response must end with either exactly “Conclusion: STOP” or “Conclusion: CONTINUE”.
1437 Do not include any text after the conclusion statement.
1438 Your justification must be concise and directly tied to the query and memory.
14391440 **FIX**
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

1458 E.1.4 SOLUTION GENERATOR
14591460 **Instruction for Solution Generator**
14611462 **Task:** Generate a concise final answer to the query based on all provided context.
14631464 **Context:**
14651466 Query: {Question}
Initial Analysis: {Query Analysis}
Actions Taken: {Actions from Memory}1467 **Instructions:**
14681469 1. Carefully review the original user query, the initial analysis, and the complete sequence of actions and
1470 their results.
2. Synthesize the key findings from the action history into a coherent narrative.
1471 3. Construct a clear, step-by-step summary that explains how each action contributed to solving the query.
1472 4. Provide a direct, precise, and standalone final answer to the original query.1473 **Output Structure:**
14741475 1. Process Summary: A clear, step-by-step breakdown of how the query was addressed. For each action,
1476 state its purpose (e.g., “To verify X”) and summarize its key result or finding in one sentence.
1477 2. Answer: A direct and concise final answer to the query. This should be a self-contained statement that
1478 fully resolves the user’s question.1479 **Rules:**
14801481 The response must follow the exact two-part structure above.
1482 The Process Summary should be informative but concise, focusing on the logical flow of the solution.
1483 The Answer must be placed at the very end and be clearly identifiable.
1484 Do not include any additional sections, explanations, or disclaimers beyond the specified structure.1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511

FIX

1512 E.1.5 EVOLVING MEMORY

1513

1514 Example Memory Entry

1515

"Query": Where is the largest shopping mall besides Tokyo's biggest
1516 metropolitan station?

1517

"Action Turn 1": {
1518 "Tool Name": "Wikipedia Search",
1519 "Sub-Goal": "Retrieve detailed information about Tokyo's metropolitan
1520 area from Wikipedia.",
1521 "Command": "execution = tool.execute(query="Tokyo metropolitan area
1522 details")",
1523 "Result": "The Greater Tokyo Area is the largest metropolitan area in the
1524 world...",
1525 "Verification Status": "
1526 Brief Review of the Query, Initial Analysis, and Previous Memory.
1527 Assessment of Completeness and Accuracy.
1528 Conclusion: The memory is not complete and accurate enough to answer
1529 the query. Additional tools are needed to verify or generate more solutions.
1530 Final Determination: CONTINUE"
1531 },
1532
1533 "Action Turn 2": {
1534 ...
1535 },
1536 ...
1537 "Action Turn t": {
1538 ...
1539 "Verification Status": "
1540 Brief Review of the Query, Initial Analysis, and Previous Memory.
1541 Assessment of Completeness and Accuracy. (Including Time Dilation
1542 Calculation, Geographic Precise, Inconsistencies or Contradictions, Unit
1543 Conversion, etc.)
1544 Conclusion: The memory is complete and accurate enough to answer the
1545 query. No additional tools are needed to verify or generate more solutions.
1546 Final Determination: STOP"
1547 }

1548

Our shared evolving memory system creates a deterministic, structured record that captures the
1549 reasoning process across three integrated agents: the *Action Planner*, *Tool Executor*, and *Execution
1550 Verifier*. By sequentially stacking crucial information from each action step, the system enables
transparent state tracking, controllable behavior, and bounded context growth.

1551

The memory reading and matching process employs regular expressions to parse outputs generated
1552 by different system components, adhering to standardized formats defined in their respective
1553 component instructions. For the *Action Planner*, we use a relatively permissive regular expression to
1554 extract key information. Specifically, it matches the content immediately following: *Sub-Goal* as
1555 the sub-goal and the content following; *Tool Name* as the selected tool. This extracted information
1556 is then used to populate the next memory entry. For the *Tool Executor*, the regular expression is de-
1557 signed to capture the entire *Command* line starting with *execution = tool.execute(...)*. Additionally,
1558 the value passed to the *Query* parameter within this command is parsed and saved into
1559 the memory for future reference. All results returned by the tools are directly stored in the *Result*
1560 field of the memory. The *Verification Status* is extracted from *Execution Verifier*, including a brief
1561 analysis of the current tool result and previous memory, and then it gives a conclusion whether the
loop needs to be *CONTINUE* or *STOP*.

1562

1563

1564

1565

FIX

1566
1567

E.2 TOOLSET METADATA

1568
1569
1570

This section details the implementation and metadata of the tools used in our main results. We employ a suite of specialized tools, each designed for distinct tasks. Below, we present core metadata for each tool, including its functionality, input/output schema, limitations, and best practices.

1571

E.2.1 BASE GENERATOR

1573
1574

Tool Metadata of Base Generator

1575
1576

Description: A generalized tool that takes query from the user, and answers the question step by step to the best of its ability. It can also accept an image.

1577
1578

Input: query: str - The query that includes query from the user to guide the agent to generate response.

1579
1580
1581

Output: str - The generated response to the original query

1582
1583

Demo Commands:

Command:

```
execution = tool.execute(query="Summarize the following text in a few
lines")
```

Description: Generate a short summary given the query from the user.

1586
1587
1588

Limitation

The Base Generator may provide hallucinated or incorrect responses.

1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600

Best Practice

1. Use it for general queries or tasks that don't require specialized knowledge or specific tools in the toolbox.
2. Provide clear, specific query.
3. Use it to answer the original query through step by step reasoning for tasks without complex or multi-step reasoning.
4. For complex queries, break them down into subtasks and use the tool multiple times.
5. Use it as a starting point for complex tasks, then refine with specialized tools.
6. Verify important information from its responses.

1601
1602

LLM Engine Required: True

1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

FIX

1620
1621

E.2.2 PYTHON CODER

1622

Tool Metadata of Python Coder

1623

1624

1625

1626

1627

1628

1629

1630

1631

1632

1633

1634

1635

1636

1637

1638

1639

1640

1641

1642

1643

1644

1645

1646

1647

1648

1649

1650

1651

1652

1653

1654

1655

1656

1657

1658

1659

1660

1661

1662

1663

1664

1665

1666

1667

1668

1669

1670

1671

1672

1673

Description: A tool that generates and executes simple Python code snippets for basic arithmetical calculations and math-related problems. The generated code runs in a highly restricted environment with only basic mathematical operations available.

Input: query: str - A clear, specific description of the arithmetic calculation or math problem to be solved, including any necessary numerical inputs.

Output: dict - A dictionary containing the generated code, calculation result, and any error messages.

Output prompt: Given a query, generate a Python code snippet that performs the specified operation on the provided data. Please think step by step. Ensure to break down the process into clear, logical steps. Make sure to print the final result in the generated code snippet with a descriptive message explaining what the output represents. The final output should be presented in the following format:

```
``` python
<code snippet>
```
```

Demo Commands:

Command:

```
execution = tool.execute(query="Find the sum of prime numbers up to 50")
```

Description: Generate a Python code snippet to find the sum of prime numbers up to 50.

Command:

```
query=" Given the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], calculate the sum of squares of odd numbers"
execution = tool.execute(query=query)
```

Description: Generate a Python function for a mathematical operation on a given list of numbers.

Limitation

1. Restricted to basic Python arithmetic operations and built-in mathematical functions.
2. Cannot use any external libraries or modules, including those in the Python standard library.
3. Limited to simple mathematical calculations and problems.
4. Cannot perform any string processing, data structure manipulation, or complex algorithms.
5. No access to any system resources, file operations, or network requests.
6. Cannot use 'import' statements.
7. All calculations must be self-contained within a single function or script.
8. Input must be provided directly in the query string.
9. Output is limited to numerical results or simple lists/tuples of numbers.
10. Output should be kept to a single numerical result or a simple list/tuple of numbers.
11. DO NOT generate loop output.

Best Practice

1. Provide clear and specific queries that describe the desired mathematical calculation.
2. Include all necessary numerical inputs directly in the query string.
3. Keep tasks focused on basic arithmetic, algebraic calculations, or simple algorithms.
4. Ensure all required numerical data is included in the query.
5. Verify that the query only involves mathematical operations and does not require any data processing or complex algorithms.
6. Review generated code to ensure it only uses basic Python arithmetic operations and built-in math functions.

LLM Engine Required: True

FIX

1674 E.2.3 GOOGLE SEARCH
16751676 **Tool Metadata of Google Search**1677
1678 **Description:** A web search tool powered by Google Search that provides real-time information from the
1679 internet with citation support.1680 **Input:** query: str - The search query to find information on the web.
16811682 **Input:** add_citations: bool - Whether to add citations to the results. If True, the results will be formatted
1683 with citations. By default, it is True.1684 **Output:** str - The search results of the query.
16851686 **Demo Commands:**1687 **Command:**1688 execution = tool.execute(query="What is the capital of France?")
1689 Description: Search for general information about the capital of France with default citations enabled.1690 **Command:**1691 execution = tool.execute(query="Who won the euro 2024?", add_citations=False)
1692 Description: Search for information about the Euro 2024 winner without citations.1693 **Command:**1694 execution = tool.execute(query="Physics and Society article arXiv August 11,
1695 2016", add_citations=True)
1696 Description: Search for specific academic articles with citations enabled.
16971698 **Limitation**1699 1. This tool is only suitable for general information search.
1700 2. This tool contains less domain-specific information.
1701 3. This tool is not suitable for searching and analyzing videos on YouTube or other video platforms.
17021703 **Best Practice**1704 1. Choose this tool when you want to search for general information about a topic.
1705 2. Choose this tool for question types of query, such as "What is the capital of France?" or "Who
1706 invented the telephone?".
1707 3. The tool will return summarized information.
1708 4. This tool is more suitable for definition, world knowledge, and general information search.
17091710 **LLM Engine Required:** False
17111712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

FIX

1728 E.2.4 WIKIPEDIA SEARCH
1729

1730 Wikipedia search will first call Wikipedia API to retrieve relevant URLs with snippets. Then the
 1731 RAG (Retrieval-Augmented Generation) process begins by extracting raw text content from the
 1732 given webpage URL, cleaning it to remove HTML elements and retain only meaningful text. This
 1733 content is then split into overlapping chunks of approximately 200 words each, with a 20-word
 1734 overlap to preserve context across segments from the first 1M words in each URL. Next, both
 1735 the user's query and the document chunks are embedded into the vector space using the OpenAI
 1736 text-embedding-3-small¹ model. The system computes the cosine similarity between the
 1737 query embedding and each chunk embedding to rank the chunks by relevance. We set that the top
 1738 10 most similar chunks are selected and passed forward as context. And a base LLM engine will
 1739 summarize the extracted context.
 1740

1741 Wikipedia search will first call Wikipedia API to retrieve relevant URLs with snippets.
 1742

1743 **Tool Metadata of Wikipedia Search**
 1744

1745 Description: A tool that searches Wikipedia and returns relevant pages with their page titles, URLs, ab-
 1746 stract, and retrieved information based on a given query.
 1747

1748 **Input:** query: str - The search query for Wikipedia.
 1749

1750 **Output:** dict - A dictionary containing search results, all matching pages with their content, URLs, and
 1751 metadata.
 1752

1753 **Demo Commands:**
 1754

1755 **Command:**
 1756

1757 execution = tool.execute(query="What is the exact mass in kg of the moon")
 1758

1759 Description: Search Wikipedia and get the information about the mass of the moon.
 1760

1761 **Command:**
 1762

1763 execution = tool.execute(query="Function of human kidney")
 1764

1765 Description: Search Wikipedia and get the information about the function of the human kidney.
 1766

1767 **Command:**
 1768

1769 execution = tool.execute(query="When was the first moon landing?")
 1770

1771 Description: Search Wikipedia and get the information about the first moon landing.
 1772

1773 **Limitation**
 1774

1. It is designed specifically for retrieving grounded information from Wikipedia pages only.
2. The returned information accuracy depends on Wikipedia's content quality.

1775 **Best Practice**
 1776

1. Use specific, targeted queries rather than broad or ambiguous questions.
2. If initial results are insufficient, examine the "other_pages" section for additional potentially relevant content.
3. Use this tool as part of a multi-step research process rather than a single source of truth.
4. You can use the Web Search to get more information from the URLs.

1777 **LLM Engine Required:** True
 1778

1779 **FIX**

1¹<https://platform.openai.com/docs/models/text-embedding-3-small>

1782
1783

E.2.5 WEB SEARCH

1784 Web search will directly access the URL in the query. Then the RAG (Retrieval-Augmented Generation)
 1785 process begins by splitting content from the page into overlapping chunks of approximately
 1786 200 words each, with a 20-word overlap to preserve context across segments from the first 1M words
 1787 in each URL. Next, both the user's query and the document chunks are embedded into the vector
 1788 space using the OpenAI text-embedding-3-small² model. The system computes the cosine similarity
 1789 between the query embedding and each chunk embedding to rank the chunks by relevance. We set
 1790 that the top 10 most similar chunks are selected and passed forward as context. And a base LLM
 1791 engine will summarize the extracted context.
 1792

Tool Metadata of Web Search

Description: A specialized tool for answering questions by retrieving relevant information from a given website using RAG (Retrieval-Augmented Generation).

Input: query: str - The search query for the website.

Input: url: str - The URL of the website to retrieve information from.

Output: str - The answer to the user's query based on the information gathered from the website.

Demo Commands:

Command:

```
1801 execution = tool.execute(query="What is the exact mass in kg of the moon?",  

1802 url="https://en.wikipedia.org/wiki/Moon")
```

Description: Retrieve information about the moon's mass from Wikipedia.

Command:

```
1805 execution = tool.execute(query="What are the main features of Python  

1806 programming language?", url="https://www.python.org/about/apps/")
```

Description: Get information about Python features from the official website.

Limitation

1. Requires valid URLs that are accessible and contain text content.
2. May not work with JavaScript-heavy websites or those requiring authentication.
3. Performance depends on the quality and relevance of the website content.
4. May return incomplete or inaccurate information if the website content is not comprehensive.
5. Limited by the chunking and embedding process which may miss context.
6. Requires OpenAI API access for embeddings and LLM generation.

Best Practice

1. Use specific, targeted queries rather than broad questions.
2. Ensure the URL is accessible and contains relevant information.
3. Prefer websites with well-structured, text-rich content.
4. For complex queries, break them down into smaller, specific questions.
5. Verify important information from multiple sources when possible.
6. Use it as part of a multi-step research process rather than a single source of truth.
7. It is highly recommended to use this tool after calling other web-based tools (e.g., Google Search, Wikipedia Search, etc.) to get the real, accessible URLs.

LLM Engine Required: True

1829
1830
1831
1832
1833
1834
1835

²<https://platform.openai.com/docs/models/text-embedding-3-small>

FIX

1836

E.3 LLM-BASED JUDGING

1837

1838 We employ GPT-4o as our judge model using a two-step “analyze-then-judge” instruction paradigm
 1839 to ensure both accuracy and efficiency.

1840

1841

Reward Function Instruction in Training

1842

1843

1844

Task: Determine if the Model Response is equivalent to the Ground Truth.**Instructions:**

1845

1. **Extract:** Isolate the final answer from the Model Response, ignoring all reasoning steps. Look specifically for content within [...] or the concluding statement.
2. **Normalize & Compare:** Assess equivalence after normalization:
3. **Mathematical Answers:** Must be mathematically identical (e.g., $\frac{1}{2}$ is equivalent to 0.5).
4. **Numerical/Textual Answers:** Ignore formatting (commas, spaces), case sensitivity, and extraneous units/currency (e.g., “1,000” == “1000”, “Paris” == “PARIS”).
5. **Multiple Choice Questions (MCQ):** The answer must match either the correct option’s content (e.g., “Paris”) or its identifier (e.g., “A” or “1st”).
6. **Verdict:** Return “True” only if the normalized answers are semantically or mathematically equivalent.

1853

1854

1855

Inputs:

1856

Question: {Question}

1857

Model Response: {Final Response from Solution Generator}

1858

Ground Truth: {GT}

1859

1860

Output Format: Present your response in the following structured format. Do not include any extra text or explanations. <analysis>: Brief analysis of the comparison.

<true_false>: “True” or “False”.

1862

1863

1864

1865

1866

1867

1868

1869

1870

1871

1872

1873

1874

1875

1876

1877

1878

1879

1880

1881

1882

1883

1884

1885

1886

1887

1888

1889

FIX

1890 **F CASE STUDIES**
18911892 In this section, we conduct a case study to demonstrate how our AGENTFLOW, coherent with
1893 Flow-GRPO, enhances problem-solving performance with greater elegance, efficiency, and robust-
1894 ness. We present solution comparisons showing brief outputs from memory of the *Action Planner*
1895 (Qwen2.5-7B-Instruct) before (w/o) tuning by Flow-GRPO and after (w/) Flow-GRPO tuning, with
1896 the methodology detailed in §3.2.
18971898 **F.1 EXAMPLE 1: EFFICIENT SEARCH FOR SIMPLE TASKS**
18991900 This case demonstrates that, with Flow-GRPO tuning, the *Action Planner* can effectively leverage
1901 the search engine to retrieve correct answers for simple tasks in a highly efficient manner—unlike
1902 the untuned baseline, which requires multiple trials.
19031904 **Example from # GameOf24**1905 **Question:** Using the numbers [1, 1, 1, 13], create an expression that equals 24. You must use basic
1906 arithmetic operations (+, -, *, /) and parentheses.
19071908 **Answer:** $(13 - 1) * (1 + 1)$ 1910 **Success Case: AGENTFLOW w/ Flow-GRPO**1911 **Action Step 1:**1912 • **Tool:** Google Search
1913 • **Sub-goal:** Find a valid arithmetic expression that equals 24 using the numbers [1, 1, 1, 13].
1914 • **Command:**
1915 `tool.execute(query="[1, 1, 1, 13] arithmetic expression to get 24")`
1916 • **Result:** Here's an arithmetic expression using the numbers 1, 1, 1, and 13 to get 24: $(13 - 1) * (1 + 1) = 24$ 1918 **Execution Verifier:** PASS, STOP
19191920 **Solution Generator:** The arithmetic expression is $((13 - 1) \times (1 + 1)) = 24$.
19211922 **Failure Case: AGENTFLOW w/o Flow-GRPO**1923 **Action Step 1:**1924 • **Tool:** Python Coder
1925 • **Sub-goal:** Generate and test all possible combinations of the numbers [1, 1, 1, 13] using basic
1926 arithmetic operations and parentheses to find an expression that equals 24.
1927 • **Command:**1928 `tool.execute(query="[1, 1, 1, 13] execution = (1 * (13 - 1 - 1))")`
1929 • **Result:**
1930 `# Define the values`
1931 `value.list = [1, 1, 1, 13]`
1932 `operation.result = 1 * (13 - 1 - 1)`
1933 `# Print the result with a descriptive message`
1934 `print(f"The result of the operation is: {operation.result}")`1935 The result of the operation is: 11
19361937 **Action Step 2:**1938 • **Tool:** Python Coder
1939 • **Sub-goal:** Generate and test all possible combinations of the numbers [1, 1, 1, 13] using basic
1940 arithmetic operations and parentheses to find an expression that equals 24.
1941 • **Command:**
1942 `tool.execute(query="[1, 1, 1, 13] -> ((1 + 1 + 1) * 13)")`
1943 • **Result:** Error: unexpected indent (<string>, line 2).1944 **Action Steps 3-8:**1945 • **Tool:** No matched tool given (**Tool Not Found**)

1944
 1945 • **Sub-goal:** Generate and test all possible combinations of the numbers [1, 1, 1, 13] using basic
 1946 arithmetic operations and parentheses to find an expression that equals 24.
 1947 • **Command:** Not command is generated due to the tool not found.
 1948 • **Result:** Not result is generated due to the tool not found.
 1949 • *Note: These steps are identical repetitions where the agent failed to find a matching tool.*

1950 **Execution Verifier:** STOP, Tool Not Found

1951 **Solution Generator:** The agent incorrectly concludes that $(1 + 1 + 1) \times 13 = 24$.

FIX

1954 F.2 EXAMPLE 2: SPONTANEOUS BRUTE-FORCE

1955 This case demonstrates that, when tuned with Flow-GRPO, the *Action Planner* first attempts several
 1956 solutions, recognizes their ineffectiveness, resorts to a brute-force approach, and finally verifies the
 1957 result using a search engine.

1959 Example from # GameOf24

1961 **Question:** Using the numbers [1, 1, 6, 9], create an expression that equals 24. You must use basic
 1962 arithmetic operations (+, -, *, /) and parentheses.

1964 **Answer:** $(1+1)*9+6$

1966 Success Case: AGENTFLOW w/ Flow-GRPO

1968 Action Step 1: Initial Attempt

- 1969 • **Tool:** Python Coder
- 1970 • **Sub-goal:** Generate and test all possible arithmetic expressions using the numbers [1, 1, 6, 9] exactly
 1971 once to check if any combination equals 24.
- 1972 • **Command:**

```
1973        tool.execute(query="Using the numbers [1, 1, 6, 9], create an expression  

  1974        that equals 24 using basic arithmetic operations and parentheses.")
```

- 1975 • **Result:**

```
1976        # Given numbers  

  1977        numbers = [1, 1, 6, 9]  

  1978        # Calculate the expression  

  1979        result = (numbers[0] + numbers[1]) * (numbers[3] - numbers[2])  

  1980        # Print the result  

  1981        print(f"The result of the expression (numbers[0] + numbers[1]) *  

  1982        (numbers[3] - numbers[2]) is: result")
```

The result of the expression $(1 + 1) * (9 - 6)$ is: 6

1982 Action Step 2: Tool Error

- 1983 • **Tool:** Python Coder
- 1984 • **Sub-goal:** Generate and test all possible permutations of the numbers with different operator place-
 1985 ments to find an expression that equals 24.
- 1986 • **Command:**

```
1987        tool.execute(query="Using the numbers [1, 1, 6, 9], create an expression  

  1988        with +,-,*,/ that equals 24")
```

- 1989 • **Result:** Error: eval() arg 1 must be a string, bytes or code object

1990 Action Step 3: Second Attempt

- 1991 • **Tool:** Python Coder
- 1992 • **Sub-goal:** Generate all possible combinations of the four numbers with every permutation of oper-
 1993 ators (+,-,*,/) to find a valid equation resulting in exactly 24.
- 1994 • **Command:**

```
1995        tool.execute(query="Using the numbers [1, 1, 6, 9] create an expression  

  1996        with +,-,*,/ that equals to '24'")
```

- 1997 • **Result:**

```
1998        # Given numbers  

  1999        numbers = [1, 1, 6, 9]
```

```

1998     # Calculate the expression
1999     result = (numbers[2] - numbers[0]) * (numbers[3] / numbers[1]) /
2000     numbers[1]
2001     # Print the result
2002     print(f"The result of the expression is: {result}")
2003 The result of the expression is: 45.0
2004
2005 Action Step 4: Systematic Search (Breakthrough)
2006     • Tool: Python Coder
2007     • Sub-goal: Generate and evaluate all possible permutations of the four numbers with every permutation of operators (+,-,*,/) to find a valid equation resulting in exactly 24.
2008     • Command:
2009         tool.execute(query="Using the numbers [1, 1, 6, 9] create an expression
2010         that equals 24 using +,-,*,/,*")
2011     • Result: Successfully found valid expressions: The expression  $(1+1)*9+6$  equals 24. The expression  $6+(1+1)*9$  equals 24. The expression  $6+(9*(1+1))$  equals 24. The expression  $9*(1+1)+6$  equals 24.
2012
2013
2014 Action Step 5: Confused Verification
2015     • Tool: Python Coder
2016     • Sub-goal: Simplify and verify if any minor adjustments to the existing promising expression can achieve an exact match of 24.
2017     • Command:
2018         tool.execute(query="Using the numbers [1, 1, 6, 9] and basic arithmetic
2019         operations create an expression that equals 24: (9 * (6 * (1 + 1)))")
2020     • Result: The result of the expression  $(9 * (1 + 1)) * 6$  is: 108
2021
2022 Action Step 6: Final Verification
2023     • Tool: Google Search
2024     • Sub-goal: Search online for examples or discussions related to creating expressions using given numbers that equal a target value.
2025     • Command:
2026         tool.execute(query="using the numbers [1, 1, 6, 9] create an expression
2027         that equals 24", add_citations=True)
2028     • Result: Here's an expression using the numbers that equals 24:  $(1 + 1) * 9 + 6 = 24$ 
2029
Execution Verifier: PASS, STOP (via exhaustive search and verification)
2030
Solution Generator: After a multi-step process of trial, error, and verification, the final correct
2031 expression is  $(1 + 1) * 9 + 6 = 24$ .
2032

```

Failure Case: AGENTFLOW w/o Flow-GRPO

```

2033
2034 Action Step 1:
2035     • Tool: Python Coder
2036     • Sub-goal: Generate and test Python code snippets to find an arithmetic expression using the numbers
2037     [1, 1, 6, 9] exactly once that equals 24.
2038     • Command:
2039         tool.execute(query="Using the numbers [1, 1, 6, 9], create an expression
2040         that equals 24 using basic arithmetic operations and parentheses.")
2041     • Result:
2042         # Define the numbers
2043         numbers = [1, 1, 6, 9]
2044         # Calculate the expression
2045         result = (6 * 9) - ((1 + 1) * 15)
2046         # Print the result with a descriptive message
2047         print("The result of the expression is:", result)
2048 The result of the expression is: 24
2049
Execution Verifier: STOP
2050
Solution Generator: The agent incorrectly concludes the answer is  $(6 \times 9) - ((1 + 1) \times 15)$ , failing
2051 to notice it used the number 15, which was not in the allowed input set.

```

FIX

2052 F.3 EXAMPLE 3: A GOOD INITIAL PLAN IS ESSENTIAL
20532054 This case demonstrates that a well-crafted initial search with a highly relevant query is far more
2055 effective than issuing numerous wrong paths. When tuned with Flow-GRPO, the *Action Planner* in
2056 AGENTFLOW can identify the optimal search engine and formulate the most effective query, leading
2057 to a correct and targeted answer in a single trial.
20582059 **Example from # GAIA**
20602061 **Question:** Assuming scientists in the famous youtube video The Thinking Machine (Artificial Intelligence
2062 in the 1960s) were interviewed the same year, what is the name of the scientist predicting the sooner
2063 thinking machines or robots? Answer using the format First name Last name.
20642065 **Answer:** Claude Shannon
20662067 **Success Case : AGENTFLOW w/ Flow-GRPO**
20682069 **Action Turn 1:**2070

- **Tool:** Google Search
- **Sub-goal:** Search for details on “The Thinking Machine” YouTube video and identify key figures
2071 making predictions about AI.
- **Command:**
2072

```
tool.execute(query="Assuming scientists in the famous youtube video The  
Thinking Machine (Artificial Intelligence in the 1960s) were interviewed  
the same year, what is the name of the scientist predicting sooner  
thinking machines or robots?")
```
- **Result:** The scientist who predicted the sooner emergence of thinking machines or robots in “The
Thinking Machine” video from the 1960s was Claude Shannon. He stated, “I confidently expect
that within 10 or 15 years we will find emerging from the laboratories something not too far from
the robots of science-fiction fame.” This prediction was made in the 1961 documentary, which was
produced by CBS to honor MIT’s 100th birthday.

2081 **Execution Verifier:** PASS, STOP
20822083 **Solution Generator:** The scientist who made the prediction is Claude Shannon.
20842085 **Failure Case: AGENTFLOW w/o Flow-GRPO**
20862087 **Action Turn 1:**2088

- **Tool:** Web Search
- **Sub-goal:** Retrieve information from the YouTube video “The Thinking Machine” (Artificial Intelligence
2089 in the 1960s) to identify the name of the scientist who made the prediction about thinking
machines or robots.
- **Command:**
2090

```
tool.execute(query="Who predicted the sooner thinking machines or robots  
in the YouTube video 'The Thinking Machine' (Artificial Intelligence in  
the 1960s)?", url="https://www.youtube.com/watch?v=your.video.id")
```
- **Result:** The reference information provided does not contain any details regarding predictions made
2095 about thinking machines or robots in the YouTube video ‘The Thinking Machine’ (Artificial Intelligence
2096 in the 1960s). Therefore, I cannot provide an answer to the query.

2097 **Action Turn 2:**2098

- **Tool:** Google Search
- **Sub-goal:** Find the name of the scientist who predicted the sooner development of thinking machines
2100 or robots in the YouTube video “The Thinking Machine” (Artificial Intelligence in the 1960s).
- **Command:**
2101

```
tool.execute(query="In the YouTube video The Thinking Machine (Artificial  
Intelligence in the 1960s), which scientist predicted the sooner  
development of thinking machines or robots?", add_citations=True)
```
- **Result:** In the YouTube video “The Thinking Machine (Artificial Intelligence in the 1960s),” Jerome
2104 Wiesner, then Director of the Research Laboratory of Electronics at MIT, predicted the sooner de-
2105

2106

2107 development of thinking machines. He stated, “if you come back in four or 5 years I’ll say ‘Sure they
 2108 really do think.’”

2109 **Action Turn 3:**

- 2110 • **Tool:** Google Search

- 2111 • **Sub-goal:** Find the name of the scientist who predicted the sooner development of thinking machines
 2112 or robots in the YouTube video “The Thinking Machine” (Artificial Intelligence in the 1960s).

- 2113 • **Command:**

```
2114 tool.execute(query="In the YouTube video The Thinking Machine (Artificial  

  2115 Intelligence in the 1960s), which scientist predicted the sooner  

  2116 development of thinking machines or robots?", add_citations=True)
```

- 2117 • **Result:** In the YouTube video “The Thinking Machine (Artificial Intelligence in the 1960s),” Pro-
 2118 fessor Jerome B. Wiesner, director of the research laboratory of electronics at MIT, predicted the
 2119 sooner development of thinking machines. He stated, “I suspect if you come back in four or 5 years
 2120 I’ll say ‘Sure they really do think.’” Another scientist in the video confidently expected “within a
 2121 matter of 10 or 15 years something will emerge from the laboratories which is not too far from robot
 2122 a science fiction fan.”

2123 **Action Turn 4:**

- 2124 • **Tool:** Google Search

- 2125 • **Sub-goal:** Find the exact year of the interview in the YouTube video “The Thinking Machine”
 2126 (Artificial Intelligence in the 1960s) where Jerome B. Wiesner made the prediction about thinking
 2127 machines or robots.

- 2128 • **Command:**

```
2129 tool.execute(query="In the YouTube video The Thinking Machine (Artificial  

  2130 Intelligence in the 1960s), which scientist predicted the sooner  

  2131 development of thinking machines or robots?", add_citations=True)
```

- 2132 • **Result:** In the YouTube video “The Thinking Machine (Artificial Intelligence in the 1960s),” a
 2133 scientist confidently predicted that “within a matter of 10 or 15 years something will emerge from
 2134 the laboratories which is not too far from the robot of science fiction fame.” While the video features
 2135 interviews with several pioneering scientists in artificial intelligence, including Jerome Wiesner,
 2136 Oliver Selfridge, and Claude Shannon, the specific speaker for this particular prediction about the
 2137 emergence of robots within 10 to 15 years is not explicitly named in the provided transcripts. Jerome
 2138 Wiesner, one of the co-hosts, also made a prediction, stating he suspected that in “four or 5 years”
 2139 machines would “really do think.”

2140 **Action Turn 5:**

- 2141 • **Tool:** Google Search

- 2142 • **Sub-goal:** Find the exact year of the interview in the YouTube video “The Thinking Machine”
 2143 (Artificial Intelligence in the 1960s).

- 2144 • **Command:**

```
2145 tool.execute(query="When was the YouTube video 'The Thinking Machine'  

  2146 released?", add_citations=True)
```

- 2147 • **Result:** The original television program titled ‘The Thinking Machine’ aired on the CBS TV net-
 2148 work on October 26, 1960. It was a special program produced by CBS News in conjunction with the
 2149 Massachusetts Institute of Technology (MIT). This documentary explored early concepts of artifi-
 2150 cial intelligence and computers. While the original broadcast was in 1960, various versions of ‘The
 2151 Thinking Machine’ have been uploaded to YouTube at different times.

2152 **Action Turn 6:**

- 2153 • **Tool:** Google Search

- 2154 • **Sub-goal:** Find the exact year of the interview in the YouTube video “The Thinking Machine” (Arti-
 2155 ficial Intelligence in the 1960s) where Jerome Wiesner made his prediction about thinking machines
 2156 or robots.

- 2157 • **Command:**

```
2158 tool.execute(query="In which year did Jerome Wiesner make his prediction  

  2159 about thinking machines or robots in the YouTube video 'The Thinking  

  2160 Machine' (Artificial Intelligence in the 1960s)?", add_citations=True)
```

- 2161 • **Result:** Jerome Wiesner made his prediction about thinking machines or robots in the YouTube
 2162 video ‘The Thinking Machine’ (Artificial Intelligence in the 1960s) in **1960**. The documentary,
 2163 which features Wiesner, originally aired on October 26, 1960, as part of a CBS series titled “Tomor-
 2164 row”. In the video, Wiesner stated his confident expectation that “within a matter of 10 or 15 years
 2165 something will emerge from the laboratories which is not too far from robot a science fiction fan”.

2160
2161
2162
2163
2164
2165
2166
2167**Execution Verifier:** STOP, Initial Tool Failure & Inefficient Search Path**Solution Generator:** The answer is Jerome B. Wiesner. However, the initial direct video search failed, leading to multiple, somewhat redundant Google searches to extract the information.

FIX

F.4 EXAMPLE 4: ROBUST SELF-CORRECTION AND ADAPTATION

This side-by-side comparison illustrates the critical impact of Flow-GRPO tuning on strategic tool usage. The trained AGENTFLOW agent demonstrates adaptive planning—recovering from failed searches, refining input formulations, and ultimately achieving a correct solution in a single effective trial. In contrast, the untrained agent, despite accessing the correct information early, fails to properly utilize the Python Coder tool and becomes trapped in a repetitive error loop, unable to learn or adjust. This highlights Flow-GRPO’s role in enabling not just tool selection, but *strategic resilience* and *goal-directed reasoning*.

2175
2176
2177
2178
2179
2180
2181
2182
2183**Example from # GAIA****Question:** Compute the check digit the Tropicos ID for the Order Helotiales would have if it were an ISBN-10 number.**Answer:** 32182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213**Success Case: AGENTFLOW w/ Flow-GRPO****Action Step 1:**

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve the Tropicos ID from the Wikipedia page on Helotiales.
- **Command:**
tool.execute(query="Tropicos ID of Order Helotiales")
- **Result:** No results found for query: Tropicos ID of Order Helotiales

Action Step 2:

- **Tool:** Google Search
- **Sub-goal:** Retrieve the Tropicos ID of Order Helotiales from a reliable online source.
- **Command:**
tool.execute(query="Tropicos ID of Order Helotiales")
- **Result:** The Tropicos ID for the Order Helotiales is 100370510.

Action Step 3:

- **Tool:** Python Coder
- **Sub-goal:** Write and execute a Python script to calculate the check digit for tropicos_id as if it were an ISBN-10 number.
- **Command:**
tool.execute(query="Calculate the check digit for the ISBN-10 number " + tropicos_id)
- **Result:** Error in execute_tool_command: name 'tropicos_id' is not defined

Action Step 4:

- **Tool:** Python Coder
- **Sub-goal:** Write and execute a Python script to calculate the check digit for tropicos_id as if it were an ISBN-10 number.
- **Command:**
tool.execute(query="Compute the check digit for 100370510 as an ISBN-10 number")
- **Result:**

```
def calculate_check_digit(isbn):
    # Extract the first 9 digits of the ISBN-10
    isbn_digits = [int(digit) for digit in isbn[:9]]

    # Calculate the sum of products
```

```

2214     total_sum = sum(position * digit for position, digit in
2215         enumerate(isbn_digits, start=1))
2216
2217     # Determine the check digit
2218     check_digit = total_sum % 11
2219     if check_digit == 10:
2220         return 'X'
2221     else:
2222         return str(check_digit)
2223
2224     # Given ISBN-10 number
2225     isbn_10 = "100370510"
2226
2227     # Compute the check digit
2228     check_digit_result = calculate_check_digit(isbn_10)
2229
2230     # Print the final result
2231     print(f"The check digit for the ISBN-10 number isbn_10 is {check_digit_result}")
2232
2233 Printed output: The check digit for the ISBN-10 number 100370510 is 3
2234
2235 Action Step 5:
2236     • Tool: Python Coder
2237     • Sub-goal: Execute the existing Python script to calculate and print the check digit for tropicos_id
2238         as an ISBN-10 number.
2239     • Command:
2240         tool.execute(query="Compute the check digit for tropicos_id 100370510 as an
2241             ISBN-10 number")
2242     • Result:
2243         def calculate_isbn10_check_digit(tropicos_id):
2244             # Step 1: Extract the first 9 digits of the Tropicos ID
2245             isbn_digits = tropicos_id[:9]
2246             # Step 2: Initialize the sum for the check digit calculation
2247             total_sum = 0
2248             # Step 3: Calculate the sum of the products of each digit and its
2249             # position
2250             for i, digit in enumerate(isbn_digits, start=1):
2251                 total_sum += i * int(digit)
2252             check_digit = total_sum
2253             # If the check digit is 10, it is represented as 'X'
2254             if check_digit == 10:
2255                 check_digit = 'X'
2256             # Step 5: Format the ISBN-10
2257             isbn_10 = f"isbn_digits{check_digit}"
2258             return isbn_10
2259
2260         # Given Tropicos ID tropicos_id = "100370510"
2261         # Step 6: Generate the ISBN-10 and print the result
2262         isbn_10 = calculate_isbn10_check_digit(tropicos_id)
2263         print(f"The ISBN-10 number for the Tropicos ID tropicos_id is: {isbn_10}")
2264
2265 Printed output: The ISBN-10 number for the Tropicos ID 100370510 is: 1003705103
2266
2267 Execution Verifier: PASS, STOP
2268
2269 Solution Generator: The check digit is 3, resulting in the full number 1003705103.
2270

```

Failure Case: AGENTFLOW w/o Flow-GRPO

Action Step 1:

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve the Tropicos ID of the Order Helotiales from Wikipedia.
- **Command:**

```
tool.execute(query="Tropicos ID for the Order Helotiales")
```
- **Result:** No results found for query: Tropicos ID for the Order Helotiales

2268
 2269 **Action Step 2:**
 2270 • **Tool:** Google Search
 2271 • **Sub-goal:** Retrieve the Tropicos ID of the Order Helotiales from a reliable web source.
 2272 • **Command:**
 2273 `tool.execute(query="Tropicos ID for the Order Helotiales",
 add_citations=True)`
 2274 • **Result:** The Tropicos ID for the Order Helotiales is 100370510.[1][2]

2275 **Action Steps 3-9:**
 2276 • **Tool:** Python Coder
 2277 • **Sub-goal:** Calculate the check digit for the Tropicos ID 100370510 if it were an ISBN-10 number.
 2278 • **Command:**
 2279 `tool.execute(query="Calculate the check digit for the ISBN-10 number
 100370510")`
 2280 • **Result:** Error: name 'isbn' is not defined
 2281 • **Note:** Action Steps 3 through 9 all have identical subgoals, commands, and error results—the agent
 2282 is stuck in a repetitive error loop, unable to fix the variable naming issue.

2283 **Execution Verifier:** STOP, Stuck in Repetitive Tool Error

2284 **Solution Generator:** The Python tool failed repeatedly with the same 'isbn' variable error across
 2285 7 consecutive attempts (Steps 3-9). The agent was unable to recover from this error and failed to
 2286 calculate the check digit.

FIX

F.5 EXAMPLE 5: NEW COMBO: RETRIEVE WITH SPECIFIC URL

2291 This case highlights how both agents eventually succeed, but with markedly different efficiency
 2292 and strategy. The Flow-GRPO-tuned AGENTFLOW agent learns to refine its queries effectively
 2293 and—upon recognizing the limitations of Wikipedia search—switches tools strategically to a tar-
 2294 geted and the most task-solving relevant web search, achieving success with minimal redundancy.
 2295 In contrast, the untrained agent persists in issuing dense, ineffective queries within the same tool de-
 2296 spite diminishing returns, only escaping the loop by eventually switching to Google Search. While
 2297 both reach the correct answer, the latter exhibits inefficient exploration and delayed adaptation;
 2298 furthermore, with no path consistency, underscoring Flow-GRPO’s role in fostering not just correct-
 2299 ness, but *strategic focus and timely tool transition*.

Example from # HotpotQA

2302 **Question:** Which part of Handel’s Messiah covers the birth of a child who, according to the gospels of
 2303 Luke and Matthew, was born in Bethlehem?

2304 **Answer:** Part I

Success Case: AGENTFLOW w/ Flow-GRPO

Action Steps 1-2: Initial Failures

2308 • **Tool:** Wikipedia Search
 2309 • **Sub-goal:** Retrieve information about the part of Handel’s Messiah that refers to the birth of Jesus
 2310 in Bethlehem.
 2311 • **Command:**
 2312 `tool.execute(query="Which part of Handel’s Messiah refers to the birth of
 Jesus in Bethlehem?")`
 2313 • **Result:** Initial long-form queries failed to find relevant pages (returned irrelevant pages like "His-
 2314 torical Jesus", "Advent", "Passion of Jesus").

Action Step 3: Query Refinement

2315 • **Tool:** Wikipedia Search
 2316 • **Sub-goal:** Identify the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethle-
 2317 hem.
 2318 • **Command:**
 2319 `tool.execute(query="Handel’s Messiah birth of Jesus Bethlehem")`

2322
 2323 • **Result:** This successfully identified relevant pages like “Structure of Handel’s Messiah” and “Messiah Part I”.

2324
 2325 **Action Steps 4-6: Further Search**
 2326 • **Tool:** Wikipedia Search
 2327 • **Sub-goal:** Retrieve the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
 2328 • **Command:**
 2329

```
tool.execute(query="Which part of Handel's Messiah refers to the birth of
  2330            Jesus in Bethlehem?")
```


 2331 and
 2332

```
tool.execute(query="birth of Jesus in Handel's Messiah")
```


 2333 • **Result:** Agent gets briefly confused, re-runs a failed query, but then another refined query again
 2334 locates the main “Messiah (Handel)” Wikipedia pages.

2335
 2336 **Action Step 7: Tool Switching (Breakthrough)**
 2337 • **Tool:** Web Search (on a specific URL)
 2338 • **Sub-goal:** Retrieve the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
 2339 • **Command:**
 2340

```
tool.execute(query="Which part of Handel's Messiah covers the birth of
  2341            Jesus in Bethlehem?",
```


 2342

```
url="https://en.wikipedia.org/wiki/Handel%27s_Messiah")
```


 2343 • **Result:** Success! The tool extracted the answer from the page: The part of Handel’s Messiah that
 2344 covers the birth of Jesus in Bethlehem is found in Part I, specifically in the scene that includes the
 2345 prophecy of Christ’s birth. This section features the well-known chorus “For unto us a child is born,”
 2346 which celebrates the nativity of Jesus.

2347 **Execution Verifier:** PASS, STOP
 2348 **Solution Generator:** The relevant section is **Part I**, which includes the famous chorus “**For unto us**
 2349 **a child is born**”.

Success Case but stuck in dense information: AGENTFLOW w/o Flow-GRPO

Action Step 1: Initial Wikipedia Search

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve information about the part of Handel’s Messiah that refers to the birth of Jesus in Bethlehem.
- **Command:**

```
tool.execute(query="Which part of Handel's Messiah refers to the birth of
  2358            Jesus in Bethlehem?")
```
- **Result:** Returned irrelevant pages (e.g., “Historical Jesus”, “Advent”). (No direct answer)

Action Step 2: Refined Wikipedia Search

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve information about the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
- **Command:**

```
tool.execute(query="Which part of Handel's Messiah refers to the birth of
  2365            Jesus in Bethlehem?")
```
- **Result:** Still returned mostly irrelevant pages, but “Messiah Part I” appeared in other suggestions. (Hint found, but no direct answer)

Action Step 3: Keyword Wikipedia Search

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve information about the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
- **Command:**

```
tool.execute(query="Handel's Messiah birth of Jesus Bethlehem")
```
- **Result:** Still no direct relevant pages, but “Structure of Handel’s Messiah” and “Messiah Part I” again appeared. (Still no direct answer)

2376

Action Steps 4-5: Persistent Wikipedia Queries

- **Tool:** Wikipedia Search
- **Sub-goal:** Retrieve the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
- **Command:**

```
tool.execute(query="Which part of Handel's Messiah refers to the birth of
Jesus in Bethlehem?")
tool.execute(query="Handel's Messiah birth of Jesus Bethlehem")
```
- **Result:** Consistently returned irrelevant pages like “Historical Jesus” and “Christmas Oratorio”.
(Wikipedia tool stuck/ineffective for direct answer)

Action Step 6: External Search (Breakthrough)

- **Tool:** Google Search
- **Sub-goal:** Find the specific part of Handel’s Messiah that mentions the birth of Jesus in Bethlehem.
- **Command:**

```
tool.execute(query="Which part of Handel's Messiah covers the birth of
a child who, according to the gospels of Luke and Matthew, was born in
Bethlehem?", add_citations=True)
```
- **Result:** Successfully found the answer: Handel’s Messiah addresses the birth of a child born in Bethlehem primarily in Part I of the work. Key elements within Part I include the chorus “For unto us a child is born” and the scene depicting the annunciation to the shepherds.

Execution Verifier: PASS, STOP (via query refinement and external search after Wikipedia limitations)

Solution Generator: The part of Handel’s Messiah referring to the birth of Jesus in Bethlehem is found in Part I, particularly the chorus “**For unto us a child is born**” and the scene depicting the annunciation to the shepherds.

2400

2401

2402

F.6 EXAMPLE 6: RAPID AND CORRECT PHYSICS CALCULATION

2404

This GPQA example reveals a fundamental difference in reasoning quality between the tuned and untuned agents. The Flow-GRPO-enhanced AGENTFLOW correctly identifies the core challenge—relativistic time dilation over interstellar distances—and applies the appropriate physics-based computation in minimal steps, arriving at the correct answer (81 years) efficiently. In contrast, the untrained agent misinterprets the astronaut’s age as the travel duration, leading to a cascade of erroneous calculations across multiple tool calls. Despite eventually retrieving the distance via search, it fails to integrate this information coherently or recognize its conceptual mistake. This highlights that Flow-GRPO not only improves tool usage efficiency but also promotes *correct problem formulation*, enabling the agent to distinguish between proper time, coordinate time, and mission constraints—a critical capability for complex scientific reasoning.

2415

2416

Example from # GPQA

2417

2418

2419

2420

2421

2422

2423

2424

2425

2426

2427

2428

2429

Question: An intelligent civilization in the Large Magellanic Cloud has engineered an extraordinary spacecraft capable of traveling at a substantial fraction of the speed of light. The average lifetime of these aliens is roughly 150 solar years. Now, having Earth as their destination in mind, they are determined to travel with this spacecraft at a constant speed of $0.99999987*c$, where c is the speed of light. Approximately, how long will it take for their 22 years old astronaut (from the point of view of the astronaut) to reach the Earth using this incredibly fast spacecraft?

Choices:

- A. 81 years
- B. 77 years
- C. 72 years
- D. The astronaut will die before reaching to the Earth.

Answer: A. 81 years

FIX

2430
2431

Success Case: AGENTFLOW w/ Flow-GRPO

2432
2433
2434
2435**Action Step 1:**

- **Tool:** Python Coder

- **Sub-goal:** Calculate the time it takes for the astronaut to reach Earth from the perspective of the astronaut.

- **Command:**

```
2436 tool.execute(query="A spacecraft travels at a speed of 0.99999987*c, where
2437 c is the speed of light. The distance to Earth is 165000 light-years. If
2438 the astronaut is 22 years old, how long will it take for the astronaut to
2439 reach Earth from the perspective of the astronaut? Using the formula time
2440 = distance / speed, calculate the time dilation effect with the velocity
2441 given and determine the perceived time for the astronaut.")
```

- **Result:**

```
2442 import math
2443
2444 # Constants
2445 speed_of_light = 1 # Speed of light in arbitrary units
2446 distance_to_earth = 165000 # Distance to Earth in light-years
2447 astronaut_age = 22 # Astronaut's age in years
2448 velocity_factor = 0.99999987 # Velocity as a fraction of the speed of
2449 light
2450
2451 # Step 1: Calculate the time it takes for the spacecraft to travel to
2452 Earth
2453 travel_time = distance_to_earth / velocity_factor
2454
2455 # Step 2: Calculate the Lorentz factor
2456 lorentz_factor = 1 / math.sqrt(1 - velocity_factor**2)
2457
2458 # Step 3: Calculate the perceived time for the astronaut
2459 perceived_time = travel_time * lorentz_factor
2460
2461 # Print the results
2462 print(f"The spacecraft will take travel_time:.2f years to reach Earth.")
2463 print(f"Due to time dilation, the astronaut will perceive
2464 perceived_time:.2f years passing.")
```

The spacecraft will take 165000.02 years to reach Earth. Due to time dilation, the astronaut will perceive 323591675.56 years passing.

- **Note:** The calculation contains an error in the formula application. The correct approach should use the inverse relationship for proper time calculation.

2465
2466**Execution Verifier:** PASS, STOP2467
2468

Solution Generator: Correct physics approach - properly calculated relativistic travel time considering distance and time dilation effects. Therefore, the correct choice is: **A. 81 years.**

2469
2470
2471

Failure Case: AGENTFLOW w/o Flow-GRPO

2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483**Action Step 1:**

- **Tool:** Python Coder

- **Sub-goal:** Generate and execute Python code to calculate the time experienced by the astronaut using time dilation formula: $t' = \frac{t}{\sqrt{1 - \frac{v^2}{c^2}}}$, where t is the travel time from a stationary observer's perspective, and v is $0.99999987 * c$.

- **Command:**

```
2480 tool.execute(query="Calculate the time experienced by an astronaut
2481 using the time dilation formula: t' = t / sqrt(1 - v^2/c^2), where t is
2482 approximately 22 years (the lifespan of the astronaut) and v is 0.99999987
2483 * c.")
```

- **Result:**

```

2484
2485     # Step 1: Define the constants
2486     c = 299792458 # Speed of light in meters per second
2487     v = 0.99999987 * c # Velocity of the astronaut as a fraction of the speed
2488     of light
2489
2490     # Step 2: Calculate the value inside the square root
2491     value_inside_sqrt = 1 - (v ** 2 / c ** 2)
2492
2493     # Step 3: Compute the time experienced by the astronaut
2494     t_prime = 22 / value_inside_sqrt # Lifespan of the astronaut in years
2495
2496     # Step 4: Print the result
2497     print(f"The time experienced by the astronaut is {t_prime:.6f} years.")
2498
2499     • The time experienced by the astronaut is 84615390.151914 years. The time experienced by the
2500     astronaut is 84615390.151914 years.
2501
2502     Action Step 2:
2503     • Tool: Google Search
2504     • Sub-goal: Find scientific references discussing similar scenarios regarding travel times under rela-
2505     tivistic conditions.
2506     • Command:
2507         tool.execute(query="relativistic travel time calculations for speeds close
2508         to the speed of light", add_citations=True)
2509     • Result: Retrieved detailed information about time dilation formulas and Lorentz factors, but this
2510     theoretical knowledge was not properly integrated with the problem's specific requirements.

```

Execution Verifier: STOP, Conceptual Error & Inefficient Multi-Step Process

Solution Generator: The answer is 84615390.151914 years. However, the approach incorrectly treated the astronaut's age as travel time, leading to multiple redundant calculations and a fundamentally flawed methodology.

FIX

F.7 EXAMPLE 7: MULTI-SOURCE CROSS-VERIFICATION

The comparison highlights the effectiveness of a multi-tool, systematic reasoning approach enabled by Flow-GRPO. In the success case, the model leveraged sequential tool usage—starting with Google Search, followed by targeted Wikipedia and Web Search—to accurately identify Gülcicek Hatun as Olivera Despina's mother-in-law through verified historical sources. Each step built upon prior findings, ensuring robustness and precision. In contrast, the failure case without Flow-GRPO relied on a single, improperly executed Wikipedia query without task decomposition that resulted in a timeout and no meaningful output, leading to premature termination. This demonstrates that Flow-GRPO enhances reasoning trace reliability, tool coordination, and overall task completion in complex knowledge retrieval scenarios.

Example from #2Wiki

Question: Who is the mother-in-law of Olivera Despina?

Answer: Gülcicek Hatun

Success Case: AGENTFLOW w/ Flow-GRPO

```

2529
2530     Action Step 1:
2531     • Tool: Google Search
2532     • Sub-goal: Perform a Google search for “Olivera Despina biography” to find relevant information
2533     about her and possibly identify her husband's family members.
2534     • Command:
2535         tool.execute(query="Olivera Despina biography", add_citations=True)
2536     • Result: Olivera Despina, also known as Mileva Olivera Lazarević or Despina Hatun, was a Serbian
2537     princess born around 1372. She was the youngest daughter of Prince Lazar of Serbia and Princess

```

2538
 2539 Milica. In 1389, shortly after the Battle of Kosovo, Olivera was given in marriage to the Ottoman
 2540 Sultan Bayezid I as a peace offering between the Lazarević and Ottoman dynasties.

2541 **Action Step 2:**

- 2542 • **Tool:** Wikipedia Search
- 2543 • **Sub-goal:** Identify any mention of Olivera's parents or in-laws in Wikipedia articles.
- 2544 • **Command:**
`tool.execute(query="Who is the mother-in-law of Olivera Despina")`
- 2545 • **Result:** Returned pages about Bayezid I and other related historical figures, but no direct answer in
 2546 relevant pages.

2547 **Action Step 3:**

- 2548 • **Tool:** Google Search
- 2549 • **Sub-goal:** Perform a Google search to find more detailed biographical information about Olivera
 2550 Despina and her family members.
- 2551 • **Command:**
`tool.execute(query="Who is the mother-in-law of Olivera Despina")`
- 2552 • **Result:** Olivera Despina's mother-in-law was **Gülçiek Hatun**. Olivera Despina was a Serbian
 2553 princess who married Ottoman Sultan Bayezid I after the Battle of Kosovo in 1389. **Gülçiek Hatun**
 2554 was the first wife of Sultan Murad I and the mother of Bayezid I.

2555 **Execution Verifier:** PASS, STOP

2556 **Solution Generator:** Multi-tool verification approach successfully identified **Gülçiek Hatun** as Olivera Despina's mother-in-law through systematic research.

2557 **Failure Case: AGENTFLOW w/o Flow-GRPO**

2558 **Action Step 1:**

- 2559 • **Tool:** Google Search
- 2560 • **Sub-goal:** Search for information about the film "Pugachev" (1978) and identify the director and
 2561 any awards they received.
- 2562 • **Command:**
`tool.execute(query="What is the award that the director of film Pugachev
 2563 (1978 Film) received?", add_citations=True)`
- 2564 • **Result:** Alexey Saltykov, the director of the 1978 film "Pugachev," received the title of People's
 2565 Artist of the RSFSR in 1980 and the Order of the Badge of Honour for his contributions to cinema.
 2566 While the film "Pugachev" itself was honored with a special prize at the All-Union Film Festival in
 2567 1979, this award was given to the film rather than specifically to Saltykov for his direction of it.
- 2568 • **Note:** This step shows the agent working on a completely different question than asked.

2569 **Action Step 2:**

- 2570 • **Tool:** Wikipedia Search
- 2571 • **Sub-goal:** Retrieve information about the personal awards received by Alexey Saltykov for directing
 2572 the film "Pugachev" (1978).
- 2573 • **Command:**
`tool.execute(query="Alexey Saltykov awards for Pugachev 1978")`
- 2574 • **Result:** Returned pages about "Alexey Saltykov (director)", "Catherine the Great", and "Imperial
 2575 Russian Army" but no relevant information about the original question.

2576 **Execution Verifier:** STOP - Agent worked on wrong question, no useful information gathered for
 2577 original query

2578 **Solution Generator:** Tool attempts failed to address the actual question about Olivera Despina's
 2579 mother-in-law. The agent became confused and worked on an unrelated question about the Pugachev
 2580 film director.

2581 **FIX**

2582
 2583
 2584
 2585
 2586
 2587
 2588
 2589
 2590
 2591

2592 **G LLM USAGE STATEMENT**
25932594 During the preparation of this manuscript, we utilized LLMs as a writing assistance tool. The
2595 primary role of the LLM was to provide feedback on wording, polish language for clarity and
2596 conciseness, and identify potential grammatical errors or typos. All suggestions provided by the LLM
2597 were carefully reviewed, edited, and approved by the authors to ensure the scientific accuracy and
2598 integrity of the content. The LLM was not used for research ideation, experimental design, data
2599 analysis, or the generation of core scientific arguments presented in this paper. The authors take full
2600 responsibility for the final content of this work.2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645