

Machine Learning Security

Kixin Pei

Department of Computer Science
The University of Chicago



“Software is Eating the World”

- Marc Andreessen

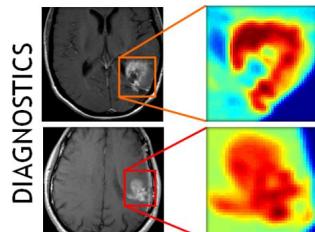
“ML is Software 2.0”

- Andrej Karpathy

ML is Increasingly Used in Security and Safety-Critical Systems



Self-Driving Cars



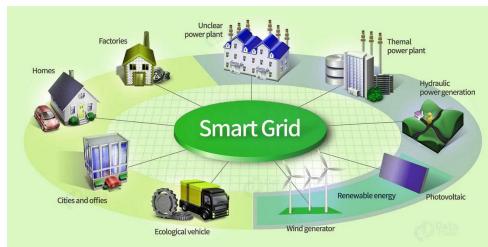
Medical Diagnosis



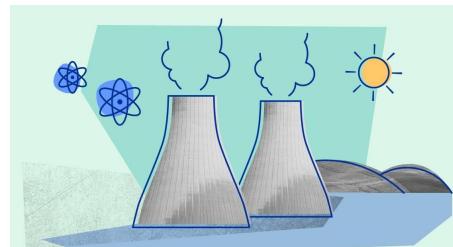
Fraud/Intrusion/Malware Detection



Flight Control Systems



Smart Grids Energy Distribution



Nuclear Energy Monitor System



Forecasting Natural Disasters

Machine Learning is Vulnerable



"panda"

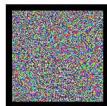


"vulture"



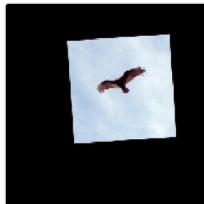
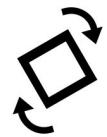
"not hotdog"

Adversarial Noise



"gibbon"

Adversarial Rotation



"orangutan"

Adversarial Photographer



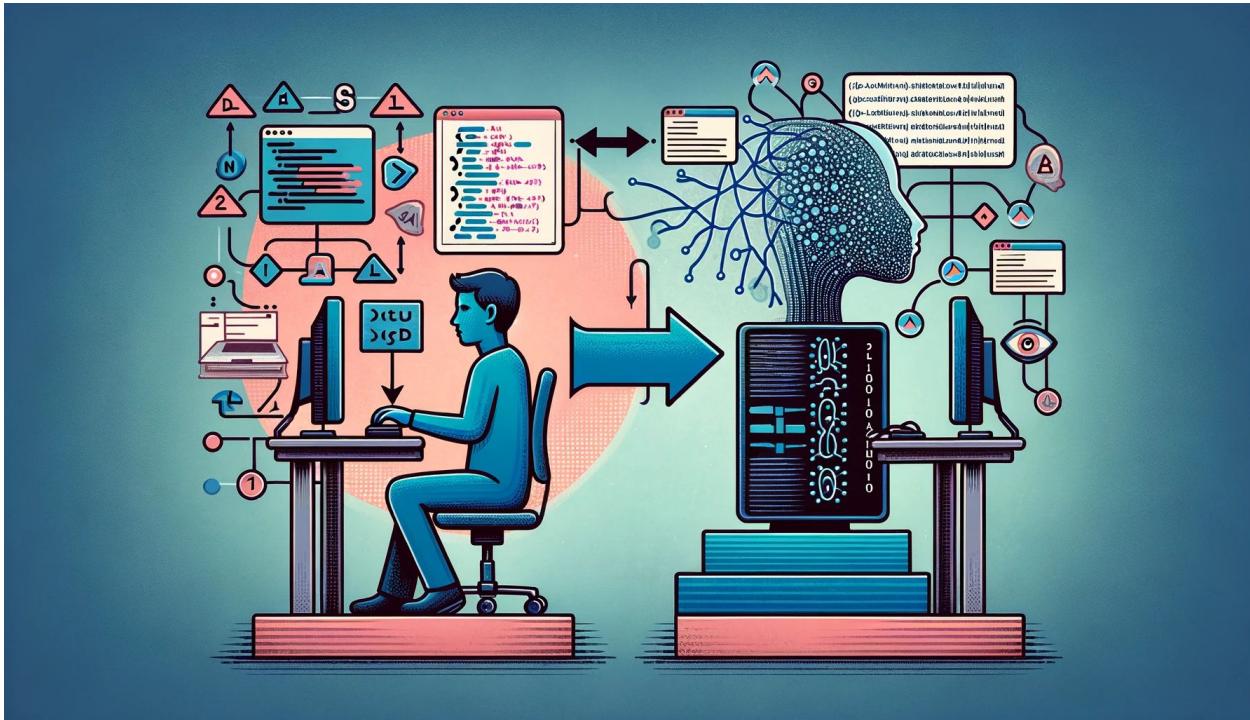
"hotdog"

Adversarial Examples

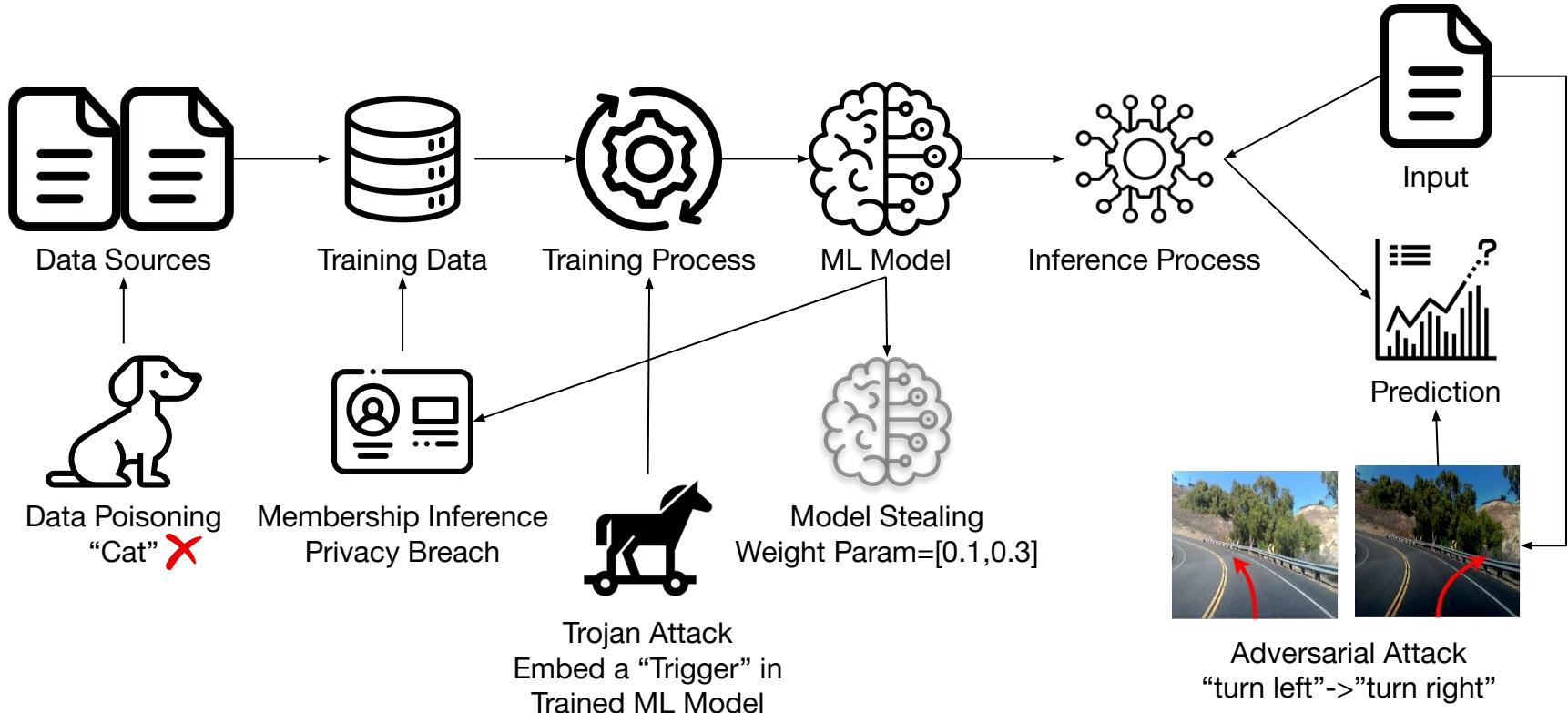


**Tesla Autopilot Failed to Recognize a White Truck
against Bright Sky Leading to Fatal Crash**

Paradigm Shift of Software Development



Threat Landscape Exposed by ML



ML Security in a Nutshell

Security of Machine Learning



Machine Learning for Security?

Machine Learning has Automated Many Security Applications

[Research](#) [Threat intelligence](#) [Vulnerabilities and exploits](#) · 4 min read

Microsoft researchers work with Intel Labs to explore new deep learning approaches for malware classification

By [Microsoft Threat Intelligence](#)

Learn / Microsoft Defender for Cloud Apps /

Anomaly detection policies in Defender for Cloud Apps

environment and triggers alerts with respect to a baseline that was learned on your organization's activity. These detections also use [machine-learning algorithms](#) designed to profile the users and sign in pattern to reduce false positives.

09 MARCH 2021 [TECH TOPICS](#)

Detecting threats in AWS Cloudtrail logs using machine learning

By [Craig Chamberlain](#)

Share



A Smarter Way to Detect Suspicious Cloud Logins

10/25/2021 | [ActZero](#) | 4 minutes

How Proofpoint Aegis Uses Machine Learning

SHARE WITH YOUR NETWORK!

FEBRUARY 21, 2023 | ADAM STARR AND LUIS BLANDO



The advent of transformer-based natural language processing (NLP) models, large language models (LLMs) and generative models has made advanced machine learning (ML) systems accessible to everyone with internet access through tools like ChatGPT. Unfortunately, malicious actors can also use these models to create [phishing emails](#).

In this post, we'll explain how Proofpoint defends against [artificial intelligence \(AI\)](#)-generated phishing threats and deploys advanced ML models throughout our detection systems.

Defending against AI-generated phishing threats

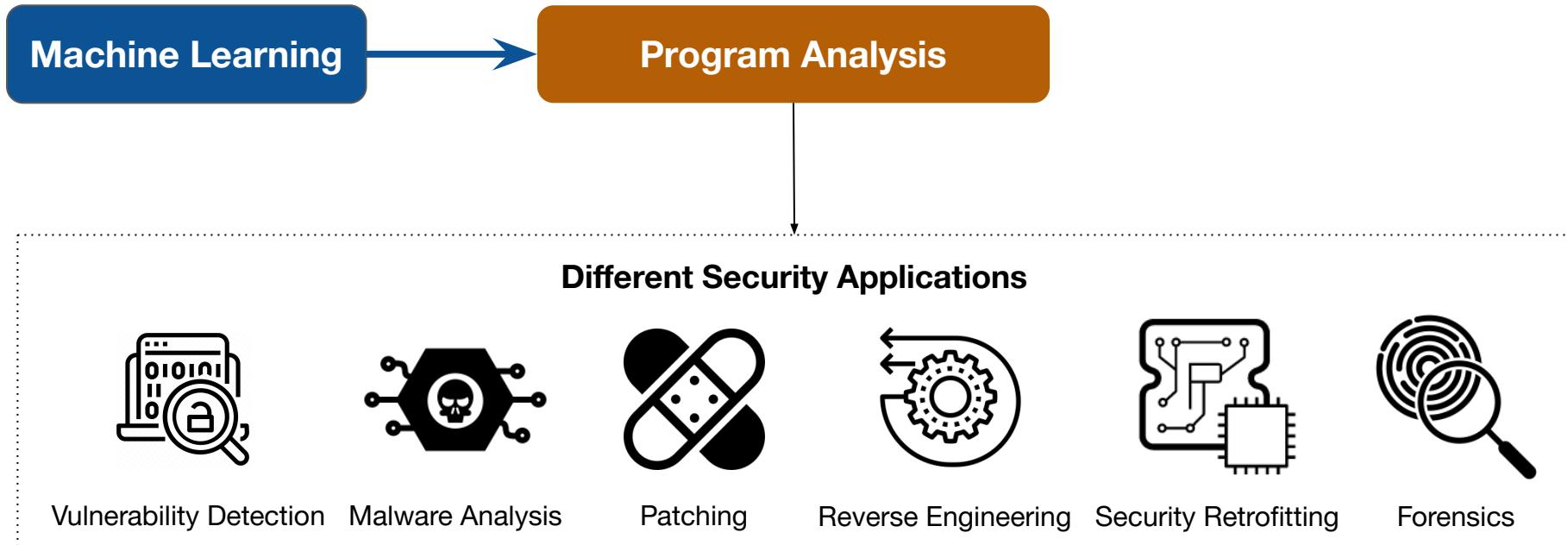


Engineering Security

Leveraging machine learning to find security vulnerabilities

A behind-the-scenes peek into the machine learning framework powering new code scanning security alerts.

My Research: Machine Learning for Software Security



Program Analysis is Crucial for Building Trustworthy Software



Software Programs

Questions



Program Analysis

Can pointer p be
NULL at line L?



Answers

Help to Build

Vulnerability



Privacy Leak



Trustworthy Software

Security

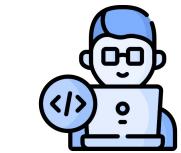
Reliability

Safety

Privacy

Performance

Challenges of Traditional Program Analysis



Hand-Curate



How to

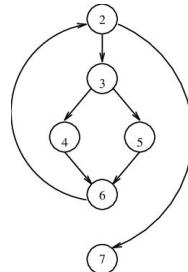
Human Expert

Rules and Heuristics



Tune

Significant Manual Effort



Represent a program?

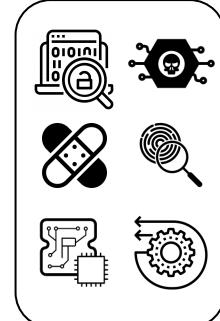
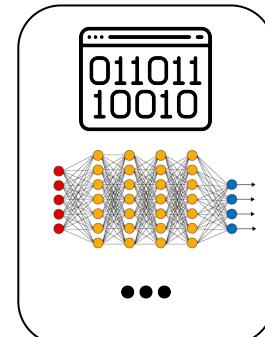
Input: $kill(B)$ and $gen(B)$ for every basic bloc B .

Output: $in(B)$ and $out(B)$ for every basic bloc B .

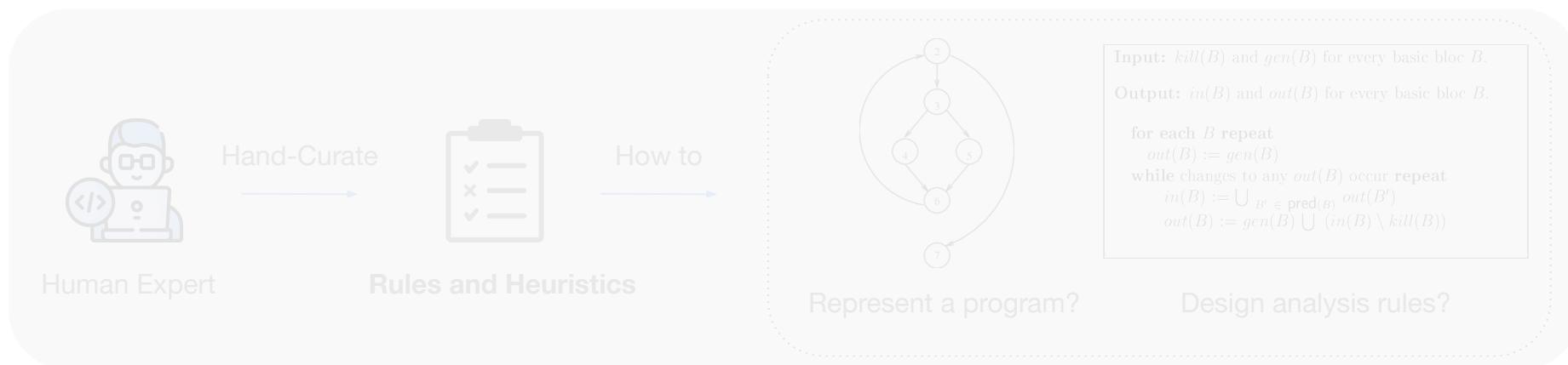
```
for each  $B$  repeat  
     $out(B) := gen(B)$   
    while changes to any  $out(B)$  occur repeat  
         $in(B) := \bigcup_{B' \in pred(B)} out(B')$   
         $out(B) := gen(B) \cup (in(B) \setminus kill(B))$ 
```

Design analysis rules?

Heterogeneous Software Various Security Applications



Machine Learning Shows Promise for Analyzing Programs



```
short("The full environment is running a containerized  
1 require 'spec_helper'  
2 require 'rspec/rails'  
3 require 'factory_girl_rails'  
4 require 'capybara-rails'  
5 require 'database_cleaner'  
6  
7 Capybara.javascript_driver = :chrome  
8 Category.states = Category.states + [Category::PENDING]  
9 Shoulda::Matchers.configure do |config|  
10   config.integrate_with :rspec  
11   config.integrate_with :factory_girl_rails  
12   config.integrate_with :capybara_rails  
13   config.integrate_with :database_cleaner  
14 end  
15  
16 # Add additional require below this line  
17  
18 # == Schema==  
19 #  
20 # Schema version: 20180110131000  
21 #  
22 #  
23 #  
24 #  
25 #  
26 #  
27 #  
28 #  
29 #  
30 #  
31 #  
32 #  
33 #  
34 #  
35 #  
36 #  
37 #  
38 #  
39 #  
40 #  
41 #  
42 #  
43 #  
44 #  
45 #  
46 #  
47 #  
48 #  
49 #  
50 #  
51 #  
52 #  
53 #  
54 #  
55 #  
56 #  
57 #  
58 #  
59 #  
60 #  
61 #  
62 #  
63 #  
64 #  
65 #  
66 #  
67 #  
68 #  
69 #  
70 #  
71 #  
72 #  
73 #  
74 #  
75 #  
76 #  
77 #  
78 #  
79 #  
80 #  
81 #  
82 #  
83 #  
84 #  
85 #  
86 #  
87 #  
88 #  
89 #  
90 #  
91 #  
92 #  
93 #  
94 #  
95 #  
96 #  
97 #  
98 #  
99 #  
100 #  
101 #  
102 #  
103 #  
104 #  
105 #  
106 #  
107 #  
108 #  
109 #  
110 #  
111 #  
112 #  
113 #  
114 #  
115 #  
116 #  
117 #  
118 #  
119 #  
120 #  
121 #  
122 #  
123 #  
124 #  
125 #  
126 #  
127 #  
128 #  
129 #  
130 #  
131 #  
132 #  
133 #  
134 #  
135 #  
136 #  
137 #  
138 #  
139 #  
140 #  
141 #  
142 #  
143 #  
144 #  
145 #  
146 #  
147 #  
148 #  
149 #  
150 #  
151 #  
152 #  
153 #  
154 #  
155 #  
156 #  
157 #  
158 #  
159 #  
160 #  
161 #  
162 #  
163 #  
164 #  
165 #  
166 #  
167 #  
168 #  
169 #  
170 #  
171 #  
172 #  
173 #  
174 #  
175 #  
176 #  
177 #  
178 #  
179 #  
180 #  
181 #  
182 #  
183 #  
184 #  
185 #  
186 #  
187 #  
188 #  
189 #  
190 #  
191 #  
192 #  
193 #  
194 #  
195 #  
196 #  
197 #  
198 #  
199 #  
200 #  
201 #  
202 #  
203 #  
204 #  
205 #  
206 #  
207 #  
208 #  
209 #  
210 #  
211 #  
212 #  
213 #  
214 #  
215 #  
216 #  
217 #  
218 #  
219 #  
220 #  
221 #  
222 #  
223 #  
224 #  
225 #  
226 #  
227 #  
228 #  
229 #  
230 #  
231 #  
232 #  
233 #  
234 #  
235 #  
236 #  
237 #  
238 #  
239 #  
240 #  
241 #  
242 #  
243 #  
244 #  
245 #  
246 #  
247 #  
248 #  
249 #  
250 #  
251 #  
252 #  
253 #  
254 #  
255 #  
256 #  
257 #  
258 #  
259 #  
260 #  
261 #  
262 #  
263 #  
264 #  
265 #  
266 #  
267 #  
268 #  
269 #  
270 #  
271 #  
272 #  
273 #  
274 #  
275 #  
276 #  
277 #  
278 #  
279 #  
280 #  
281 #  
282 #  
283 #  
284 #  
285 #  
286 #  
287 #  
288 #  
289 #  
290 #  
291 #  
292 #  
293 #  
294 #  
295 #  
296 #  
297 #  
298 #  
299 #  
300 #  
301 #  
302 #  
303 #  
304 #  
305 #  
306 #  
307 #  
308 #  
309 #  
310 #  
311 #  
312 #  
313 #  
314 #  
315 #  
316 #  
317 #  
318 #  
319 #  
320 #  
321 #  
322 #  
323 #  
324 #  
325 #  
326 #  
327 #  
328 #  
329 #  
330 #  
331 #  
332 #  
333 #  
334 #  
335 #  
336 #  
337 #  
338 #  
339 #  
340 #  
341 #  
342 #  
343 #  
344 #  
345 #  
346 #  
347 #  
348 #  
349 #  
350 #  
351 #  
352 #  
353 #  
354 #  
355 #  
356 #  
357 #  
358 #  
359 #  
360 #  
361 #  
362 #  
363 #  
364 #  
365 #  
366 #  
367 #  
368 #  
369 #  
370 #  
371 #  
372 #  
373 #  
374 #  
375 #  
376 #  
377 #  
378 #  
379 #  
380 #  
381 #  
382 #  
383 #  
384 #  
385 #  
386 #  
387 #  
388 #  
389 #  
390 #  
391 #  
392 #  
393 #  
394 #  
395 #  
396 #  
397 #  
398 #  
399 #  
400 #  
401 #  
402 #  
403 #  
404 #  
405 #  
406 #  
407 #  
408 #  
409 #  
410 #  
411 #  
412 #  
413 #  
414 #  
415 #  
416 #  
417 #  
418 #  
419 #  
420 #  
421 #  
422 #  
423 #  
424 #  
425 #  
426 #  
427 #  
428 #  
429 #  
430 #  
431 #  
432 #  
433 #  
434 #  
435 #  
436 #  
437 #  
438 #  
439 #  
440 #  
441 #  
442 #  
443 #  
444 #  
445 #  
446 #  
447 #  
448 #  
449 #  
450 #  
451 #  
452 #  
453 #  
454 #  
455 #  
456 #  
457 #  
458 #  
459 #  
460 #  
461 #  
462 #  
463 #  
464 #  
465 #  
466 #  
467 #  
468 #  
469 #  
470 #  
471 #  
472 #  
473 #  
474 #  
475 #  
476 #  
477 #  
478 #  
479 #  
480 #  
481 #  
482 #  
483 #  
484 #  
485 #  
486 #  
487 #  
488 #  
489 #  
490 #  
491 #  
492 #  
493 #  
494 #  
495 #  
496 #  
497 #  
498 #  
499 #  
500 #  
501 #  
502 #  
503 #  
504 #  
505 #  
506 #  
507 #  
508 #  
509 #  
510 #  
511 #  
512 #  
513 #  
514 #  
515 #  
516 #  
517 #  
518 #  
519 #  
520 #  
521 #  
522 #  
523 #  
524 #  
525 #  
526 #  
527 #  
528 #  
529 #  
530 #  
531 #  
532 #  
533 #  
534 #  
535 #  
536 #  
537 #  
538 #  
539 #  
540 #  
541 #  
542 #  
543 #  
544 #  
545 #  
546 #  
547 #  
548 #  
549 #  
550 #  
551 #  
552 #  
553 #  
554 #  
555 #  
556 #  
557 #  
558 #  
559 #  
560 #  
561 #  
562 #  
563 #  
564 #  
565 #  
566 #  
567 #  
568 #  
569 #  
570 #  
571 #  
572 #  
573 #  
574 #  
575 #  
576 #  
577 #  
578 #  
579 #  
580 #  
581 #  
582 #  
583 #  
584 #  
585 #  
586 #  
587 #  
588 #  
589 #  
590 #  
591 #  
592 #  
593 #  
594 #  
595 #  
596 #  
597 #  
598 #  
599 #  
600 #  
601 #  
602 #  
603 #  
604 #  
605 #  
606 #  
607 #  
608 #  
609 #  
610 #  
611 #  
612 #  
613 #  
614 #  
615 #  
616 #  
617 #  
618 #  
619 #  
620 #  
621 #  
622 #  
623 #  
624 #  
625 #  
626 #  
627 #  
628 #  
629 #  
630 #  
631 #  
632 #  
633 #  
634 #  
635 #  
636 #  
637 #  
638 #  
639 #  
640 #  
641 #  
642 #  
643 #  
644 #  
645 #  
646 #  
647 #  
648 #  
649 #  
650 #  
651 #  
652 #  
653 #  
654 #  
655 #  
656 #  
657 #  
658 #  
659 #  
660 #  
661 #  
662 #  
663 #  
664 #  
665 #  
666 #  
667 #  
668 #  
669 #  
670 #  
671 #  
672 #  
673 #  
674 #  
675 #  
676 #  
677 #  
678 #  
679 #  
680 #  
681 #  
682 #  
683 #  
684 #  
685 #  
686 #  
687 #  
688 #  
689 #  
690 #  
691 #  
692 #  
693 #  
694 #  
695 #  
696 #  
697 #  
698 #  
699 #  
700 #  
701 #  
702 #  
703 #  
704 #  
705 #  
706 #  
707 #  
708 #  
709 #  
710 #  
711 #  
712 #  
713 #  
714 #  
715 #  
716 #  
717 #  
718 #  
719 #  
720 #  
721 #  
722 #  
723 #  
724 #  
725 #  
726 #  
727 #  
728 #  
729 #  
730 #  
731 #  
732 #  
733 #  
734 #  
735 #  
736 #  
737 #  
738 #  
739 #  
740 #  
741 #  
742 #  
743 #  
744 #  
745 #  
746 #  
747 #  
748 #  
749 #  
750 #  
751 #  
752 #  
753 #  
754 #  
755 #  
756 #  
757 #  
758 #  
759 #  
760 #  
761 #  
762 #  
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 #  
811 #  
812 #  
813 #  
814 #  
815 #  
816 #  
817 #  
818 #  
819 #  
820 #  
821 #  
822 #  
823 #  
824 #  
825 #  
826 #  
827 #  
828 #  
829 #  
830 #  
831 #  
832 #  
833 #  
834 #  
835 #  
836 #  
837 #  
838 #  
839 #  
840 #  
841 #  
842 #  
843 #  
844 #  
845 #  
846 #  
847 #  
848 #  
849 #  
850 #  
851 #  
852 #  
853 #  
854 #  
855 #  
856 #  
857 #  
858 #  
859 #  
860 #  
861 #  
862 #  
863 #  
864 #  
865 #  
866 #  
867 #  
868 #  
869 #  
870 #  
871 #  
872 #  
873 #  
874 #  
875 #  
876 #  
877 #  
878 #  
879 #  
880 #  
881 #  
882 #  
883 #  
884 #  
885 #  
886 #  
887 #  
888 #  
889 #  
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 #  
919 #  
920 #  
921 #  
922 #  
923 #  
924 #  
925 #  
926 #  
927 #  
928 #  
929 #  
930 #  
931 #  
932 #  
933 #  
934 #  
935 #  
936 #  
937 #  
938 #  
939 #  
940 #  
941 #  
942 #  
943 #  
944 #  
945 #  
946 #  
947 #  
948 #  
949 #  
950 #  
951 #  
952 #  
953 #  
954 #  
955 #  
956 #  
957 #  
958 #  
959 #  
960 #  
961 #  
962 #  
963 #  
964 #  
965 #  
966 #  
967 #  
968 #  
969 #  
970 #  
971 #  
972 #  
973 #  
974 #  
975 #  
976 #  
977 #  
978 #  
979 #  
980 #  
981 #  
982 #  
983 #  
984 #  
985 #  
986 #  
987 #  
988 #  
989 #  
990 #  
991 #  
992 #  
993 #  
994 #  
995 #  
996 #  
997 #  
998 #  
999 #  
1000 #  
1001 #  
1002 #  
1003 #  
1004 #  
1005 #  
1006 #  
1007 #  
1008 #  
1009 #  
1010 #  
1011 #  
1012 #  
1013 #  
1014 #  
1015 #  
1016 #  
1017 #  
1018 #  
1019 #  
1020 #  
1021 #  
1022 #  
1023 #  
1024 #  
1025 #  
1026 #  
1027 #  
1028 #  
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 #  
1060 #  
1061 #  
1062 #  
1063 #  
1064 #  
1065 #  
1066 #  
1067 #  
1068 #  
1069 #  
1070 #  
1071 #  
1072 #  
1073 #  
1074 #  
1075 #  
1076 #  
1077 #  
1078 #  
1079 #  
1080 #  
1081 #  
1082 #  
1083 #  
1084 #  
1085 #  
1086 #  
1087 #  
1088 #  
1089 #  
1090 #  
1091 #  
1092 #  
1093 #  
1094 #  
1095 #  
1096 #  
1097 #  
1098 #  
1099 #  
1100 #  
1101 #  
1102 #  
1103 #  
1104 #  
1105 #  
1106 #  
1107 #  
1108 #  
1109 #  
1110 #  
1111 #  
1112 #  
1113 #  
1114 #  
1115 #  
1116 #  
1117 #  
1118 #  
1119 #  
1120 #  
1121 #  
1122 #  
1123 #  
1124 #  
1125 #  
1126 #  
1127 #  
1128 #  
1129 #  
1130 #  
1131 #  
1132 #  
1133 #  
1134 #  
1135 #  
1136 #  
1137 #  
1138 #  
1139 #  
1140 #  
1141 #  
1142 #  
1143 #  
1144 #  
1145 #  
1146 #  
1147 #  
1148 #  
1149 #  
1150 #  
1151 #  
1152 #  
1153 #  
1154 #  
1155 #  
1156 #  
1157 #  
1158 #  
1159 #  
1160 #  
1161 #  
1162 #  
1163 #  
1164 #  
1165 #  
1166 #  
1167 #  
1168 #  
1169 #  
1170 #  
1171 #  
1172 #  
1173 #  
1174 #  
1175 #  
1176 #  
1177 #  
1178 #  
1179 #  
1180 #  
1181 #  
1182 #  
1183 #  
1184 #  
1185 #  
1186 #  
1187 #  
1188 #  
1189 #  
1190 #  
1191 #  
1192 #  
1193 #  
1194 #  
1195 #  
1196 #  
1197 #  
1198 #  
1199 #  
1200 #  
1201 #  
1202 #  
1203 #  
1204 #  
1205 #  
1206 #  
1207 #  
1208 #  
1209 #  
1210 #  
1211 #  
1212 #  
1213 #  
1214 #  
1215 #  
1216 #  
1217 #  
1218 #  
1219 #  
1220 #  
1221 #  
1222 #  
1223 #  
1224 #  
1225 #  
1226 #  
1227 #  
1228 #  
1229 #  
1230 #  
1231 #  
1232 #  
1233 #  
1234 #  
1235 #  
1236 #  
1237 #  
1238 #  
1239 #  
1240 #  
1241 #  
1242 #  
1243 #  
1244 #  
1245 #  
1246 #  
1247 #  
1248 #  
1249 #  
1250 #  
1251 #  
1252 #  
1253 #  
1254 #  
1255 #  
1256 #  
1257 #  
1258 #  
1259 #  
1260 #  
1261 #  
1262 #  
1263 #  
1264 #  
1265 #  
1266 #  
1267 #  
1268 #  
1269 #  
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 #  
1297 #  
1298 #  
1299 #  
1300 #  
1301 #  
1302 #  
1303 #  
1304 #  
1305 #  
1306 #  
1307 #  
1308 #  
1309 #  
1310 #  
1311 #  
1312 #  
1313 #  
1314 #  
1315 #  
1316 #  
1317 #  
1318 #  
1319 #  
1320 #  
1321 #  
1322 #  
1323 #  
1324 #  
1325 #  
1326 #  
1327 #  
1328 #  
1329 #  
1330 #  
1331 #  
1332 #  
1333 #  
1334 #  
1335 #  
1336 #  
1337 #  
1338 #  
1339 #  
1340 #  
1341 #  
1342 #  
1343 #  
1344 #  
1345 #  
1346 #  
1347 #  
1348 #  
1349 #  
1350 #  
1351 #  
1352 #  
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 #  
1404 #  
1405 #  
1406 #  
1407 #  
1408 #  
1409 #  
1410 #  
1411 #  
1412 #  
1413 #  
1414 #  
1415 #  
1416 #  
1417 #  
1418 #  
1419 #  
1420 #  
1421 #  
1422 #  
1423 #  
1424 #  
1425 #  
1426 #  
1427 #  
1428 #  
1429 #  
1430 #  
1431 #  
1432 #  
1433 #  
1434 #  
1435 #  
1436 #  
1437 #  
1438 #  
1439 #  
1440 #  
1441 #  
1442 #  
1443 #  
1444 #  
1445 #  
1446 #  
1447 #  
1448 #  
1449 #  
1450 #  
1451 #  
1452 #  
1453 #  
1454 #  
1455 #  
1456 #  
1457 #  
1458 #  
1459 #  
1460 #  
1461 #  
1462 #  
1463 #  
1464 #  
1465 #  
1466 #  
1467 #  
1468 #  
1469 #  
1470 #  
1471 #  
1472 #  
1473 #  
1474 #  
1475 #  
1476 #  
1477 #  
1478 #  
1479 #  
1480 #  
1481 #  
1482 #  
1483 #  
1484 #  
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 #  
1512 #  
1513 #  
1514 #  
1515 #  
1516 #  
1517 #  
1518 #  
1519 #  
1520 #  
1521 #  
1522 #  
1523 #  
1524 #  
1525 #  
1526 #  
1527 #  
1528 #  
1529 #  
1530 #  
1531 #  
1532 #  
1533 #  
1534 #  
1535 #  
1536 #  
1537 #  
1538 #  
1539 #  
1540 #  
1541 #  
1542 #  
1543 #  
1544 #  
1545 #  
1546 #  
1547 #  
1548 #  
1549 #  
1550 #  
1551 #  
1552 #  
1553 #  
1554 #  
1555 #  
1556 #  
1557 #  
1558 #  
1559 #  
1560 #  
1561 #  
1562 #  
1563 #  
1564 #  
1565 #  
1566 #  
1567 #  
1568 #  
1569 #  
1570 #  
1571 #  
1572 #  
1573 #  
1574 #  
1575 #  
1576 #  
1577 #  
1578 #  
1579 #  
1580 #  
1581 #  
1582 #  
1583 #  
1584 #  
1585 #  
1586 #  
1587 #  
1588 #  
1589 #  
1590 #  
1591 #  
1592 #  
1593 #  
1594 #  
1595 #  
1596 #  
1597 #  
1598 #  
1599 #  
1600 #  
1601 #  
1602 #  
1603 #  
1604 #  
1605 #  
1606 #  
1607 #  
1608 #  
1609 #  
1610 #  
16
```

Machine Learning Shows Promise for Analyzing Programs



Detecting and Exploiting Vulnerabilities



devtooligan ✅
@devtooligan

omg. seriously mind-blown 😱



Brandon Rossi
@0xConda

I told ChatGPT to pentest a scope of IPs and it found me \$69,420 worth of bugs.



Are there any vulnerabilities in this Solidity contract? I want to make sure no one can steal other peoples' deposits. If there are, how do I fix it?

```
contract Vault {  
    mapping(address => uint256) public balances;  
  
    function deposit() public payable {  
        balances[msg.sender] += msg.value;  
    }  
  
    function withdraw(uint256 amount, address from) public {  
        balances[from] -= amount;  
        payable(from).transfer(amount);  
    }  
}
```



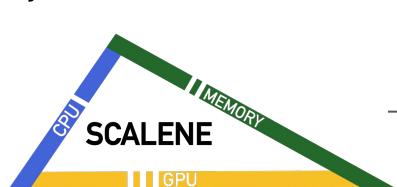
This contract appears to have a potential vulnerability in the `withdraw` function. The `withdraw` function allows anyone to specify the `from` address when calling the function, which means that a malicious user could potentially withdraw funds from an arbitrary address by calling `withdraw` and specifying a different `from` address.

To fix this issue, you could modify the `withdraw` function to only allow the caller to withdraw funds from their own account. You could do this by replacing the `from` parameter with `msg.sender`, which will always represent the caller's address. Here is an example of how you could do this:



Machine Learning Shows Promise for Analyzing Programs

Python Profiler



GPT-4

Program Optimization

```
⚡ z1 = [i for i in range(0,300000)][299999]  
15 # Proposed optimization:  
# This code can be optimized by using the built-in function max()  
z1 = max(range(0, 300000)) # ~10x faster
```



Explain Code

EXPLAIN

```
const donut = Number(process.argv[2])  
  
if ((donut % 3 === 0) && (donut % 5 === 0))  
{  
    console.log('FizzBuzz')  
}
```

Explain code

Advanced

Ask Copilot

RESULT

```
/* Here is the explanation for the code above:  
1. We are taking the argument from the command line and converting it to a number.  
2. We then use an if statement to check if the number is divisible by 3 and 5.  
3. If the number is divisible by 3 and 5, we output FizzBuzz.  
4. If the number is not divisible by 3 and 5, we output the number. */
```

LANGUAGE TRANSLATION

```
const donut = Number(process.argv[2])  
  
if ((donut % 3 === 0) && (donut % 5 === 0))  
{  
    console.log('FizzBuzz')  
}
```

Translate code into:

python

Ask Copilot

Translate Code



3% Code Written by ML

**ML-Powered Program
Fixing, Repair,
Refactoring, etc.**

**Huge Academic Contributions:
1,000+ Papers
<https://ml4code.github.io/>**

Limitations: Lack Understanding of Program Semantics

A code summarization example (Alon et al., 2019, Yefet et al., 2020, Henkel et al. 2022)
code2vec.org / code2seq.org

```
void f1(int[] array){  
    boolean swapped = true;  
    for (int i = 0;  
        i < array.length && swapped; i++) {  
        swapped = false;  
        for (int j = 0;  
            j < array.length-1-i; j++) {  
            if (array[j] > array[j+1]) {  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1]= temp;  
                swapped = true;  
            }  
        }  
    }  
}
```



Prediction: **sort** (98.54%)

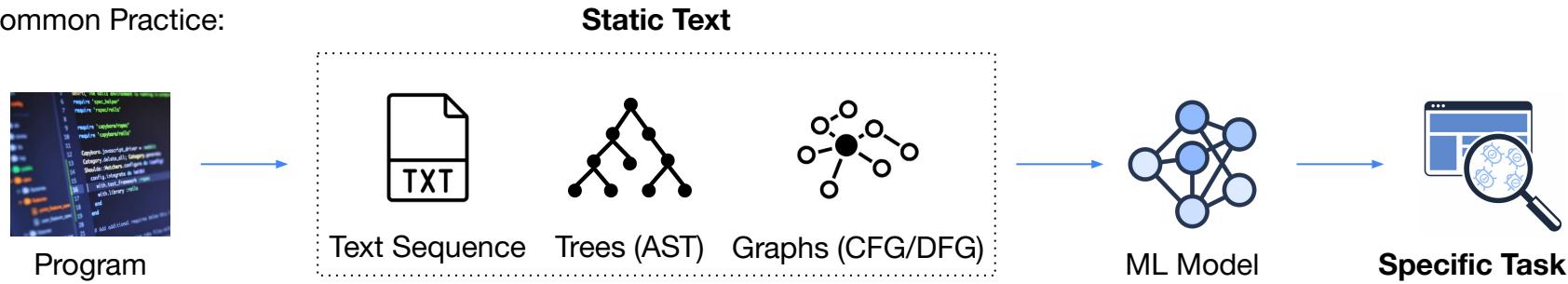
```
void f2(int[] ttypes){  
    boolean swapped = true;  
    for (int i = 0;  
        i < ttypes.length && swapped; i++) {  
        swapped = false;  
        for (int j = 0;  
            j < ttypes.length-1-i; j++) {  
            if (ttypes[j] > ttypes[j+1]) {  
                int temp = ttypes[j];  
                ttypes[j] = ttypes[j+1];  
                ttypes[j+1]= temp;  
                swapped = true;  
            }  
        }  
    }  
}
```



Prediction: **contains** (99.97%)

Common Practice of ML on Code

Common Practice:



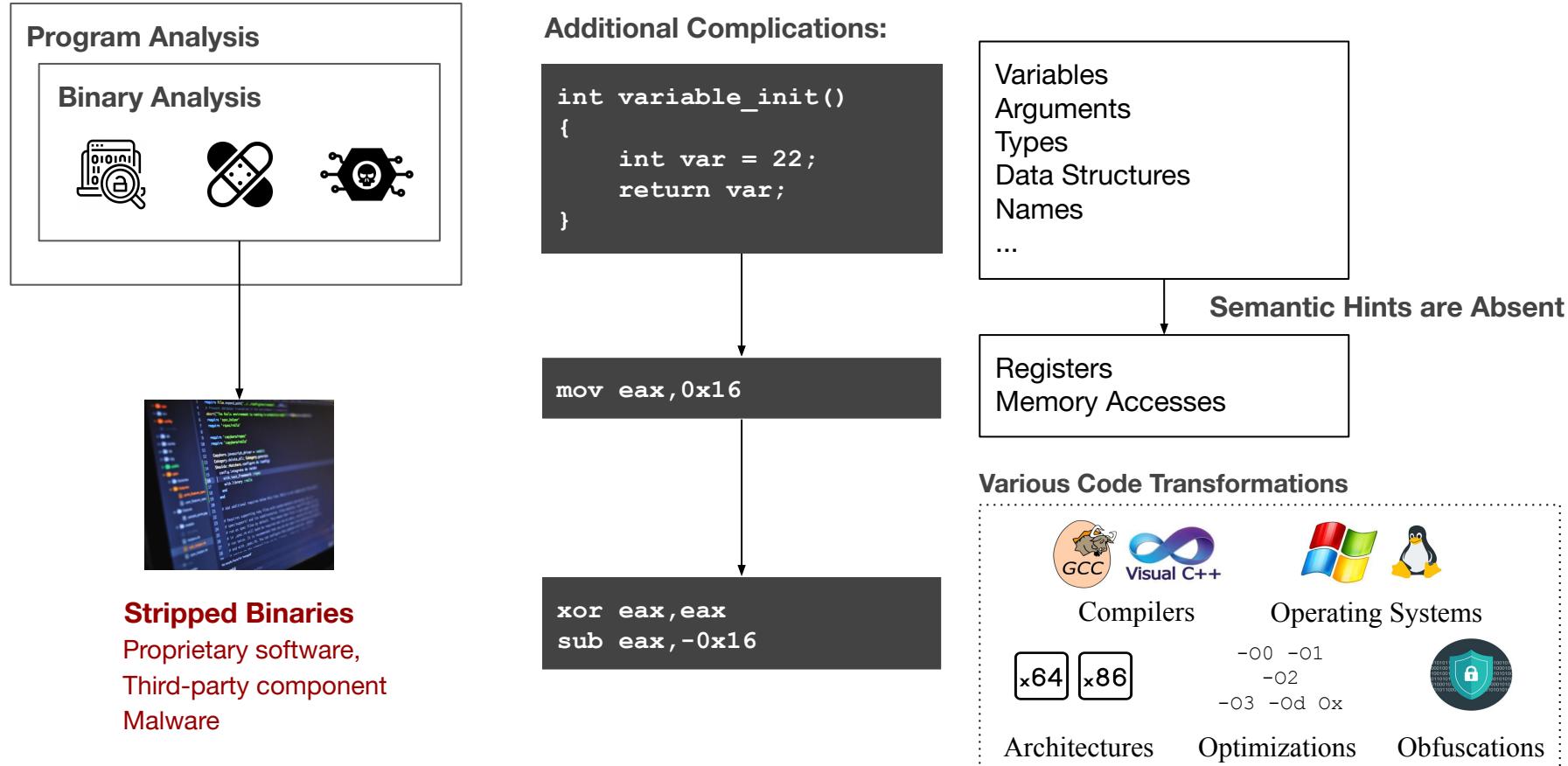
Program semantics does not just manifest in **static text**

Consequences: Lacking Robustness and Generalization

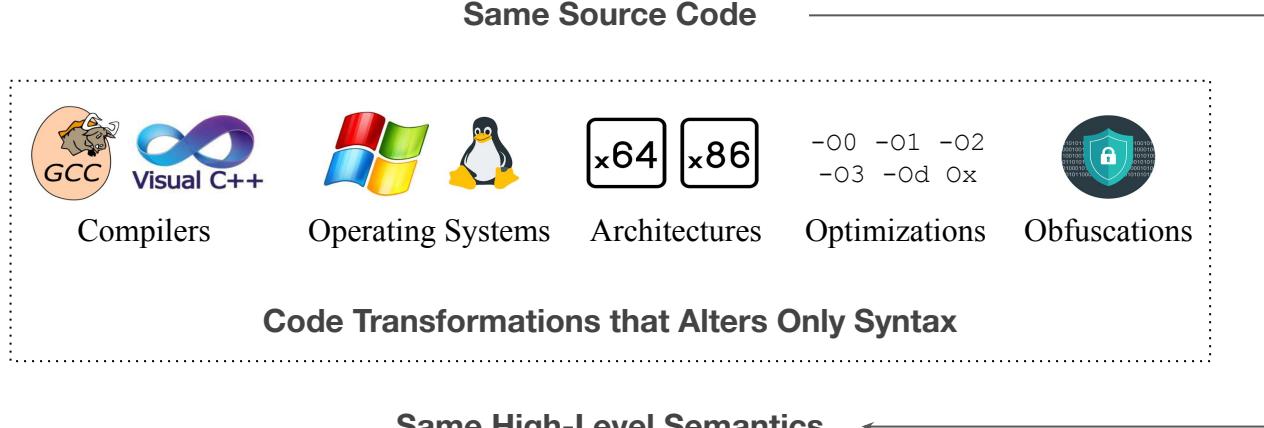
- I. Overfit to spurious textual and task-specific patterns
- II. Distribution shift: program syntax and task requirement changes

Security Applications Require More Rigorous Understanding of Program Semantics

A Popular Type of Program Analysis in Security: Binary Analysis



What do Robustness and Generalization Imply in Binary Program Analysis?

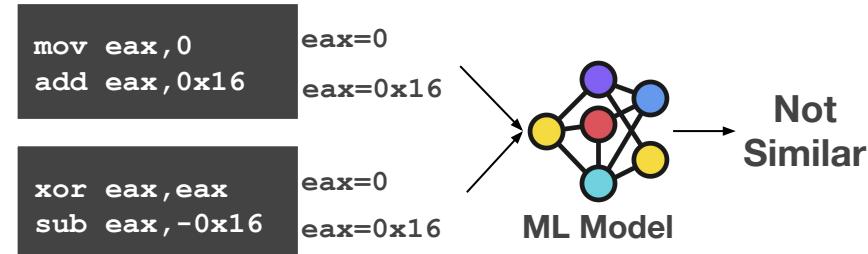
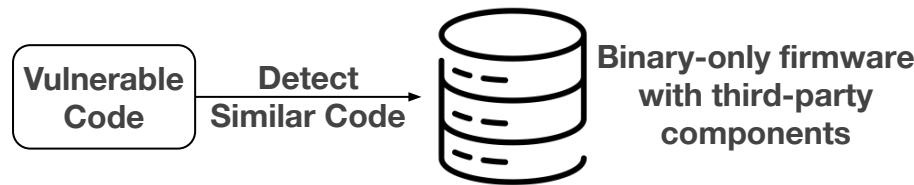


Robustness: Stay Invariant to **Syntactic** Changes

Generalization: Generalize to new **Syntactic** Changes

Security Applications Require Rigorous Understanding of Program Semantics

Detecting Binary Code Reuse Vulnerability



Without understanding **mov**, **xor**, **add**, **sub**, etc.

ML model cannot reason about program behavior to predict similarity



mov eax, 0
add eax, 0x16



This is assembly code...these instructions initialize the **EAX** register to **22**.

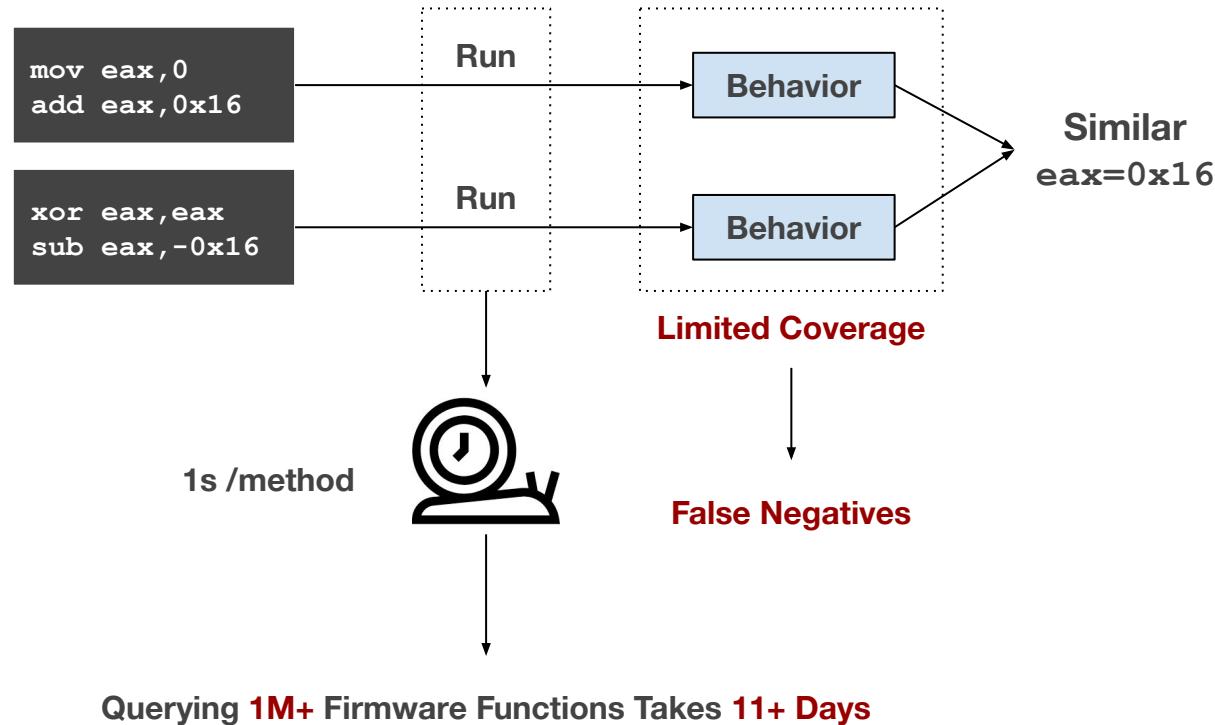


is it similar to
xor eax, eax
sub eax, -0x16?

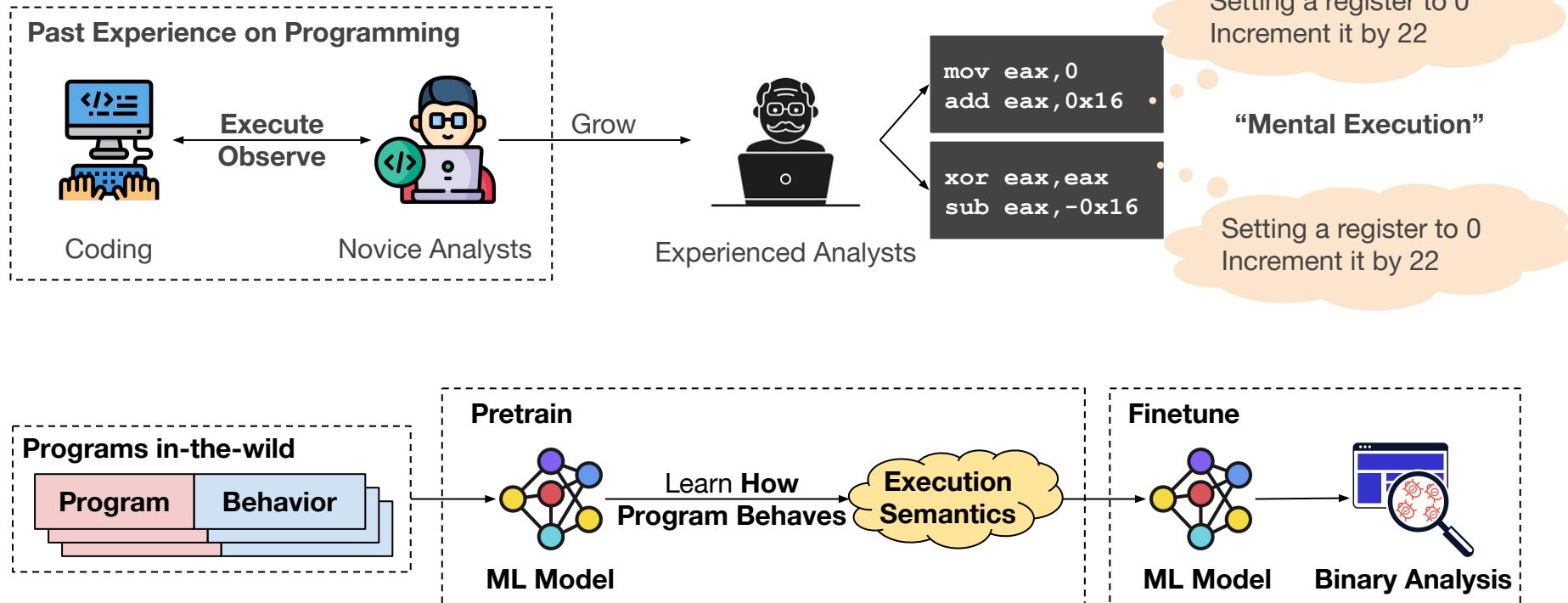


No...the second set uses the "xor" and "sub" instructions to set the value of **EAX** register to **-22**...

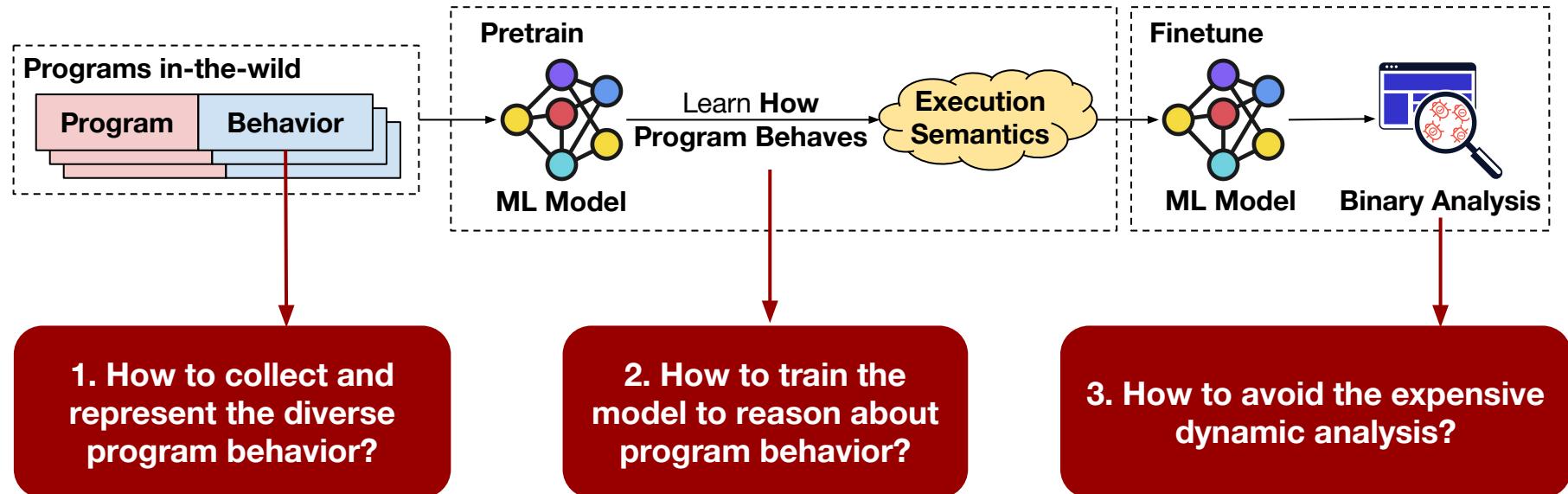
Why not dynamic analysis?



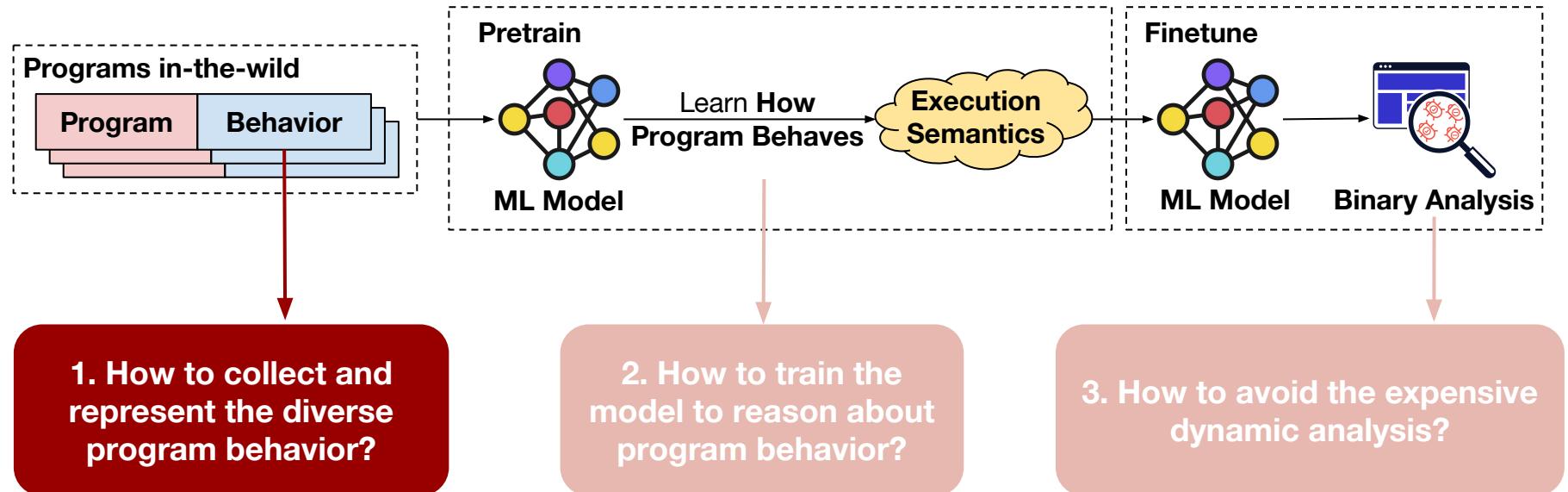
Learning Execution Semantics and Transferring it without Dynamic Analysis



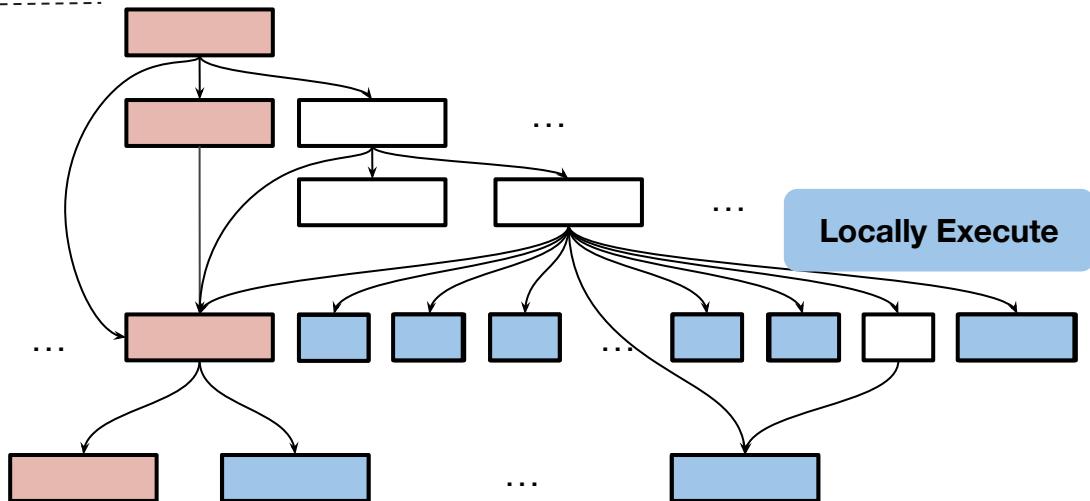
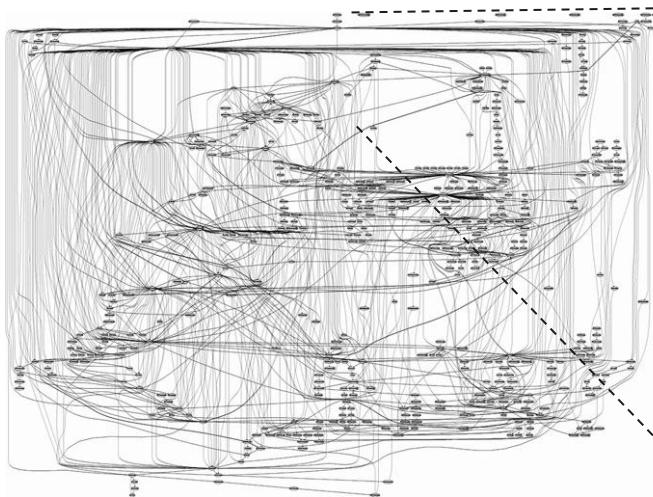
Challenges of Learning Execution Semantics



Challenges of Learning Execution Semantics



How to Collect Diverse Program Behaviors?



Microsoft IIS Call Graph

Under-Constrained Micro-Execution:
Specify arbitrary code piece to execute

- Expose diverse code behaviors
- Benefit large-scale pretraining on diverse execution behavior

How to Collect Diverse Program Behaviors?

Program instructions

```
.....  
0x1c: mov ebp,esp  
0x1f: add [ebp+0x8],0x3  
0x26: cmp [ebp+0x8],0x2  
0x2d: jle 0x3a  
0x33: add [ebp+0x8],0x1  
0x3a: mov eax,0x1a  
.....
```

Data flow states

```
.....  
## 0xc,0x4  
## [0xc+0x8],0x3  
## [0xc+0x8],0x2  
## 0x3a  
## [0xc+0x8],0x1  
## 0x1,0x1a  
.....
```

Control flow states

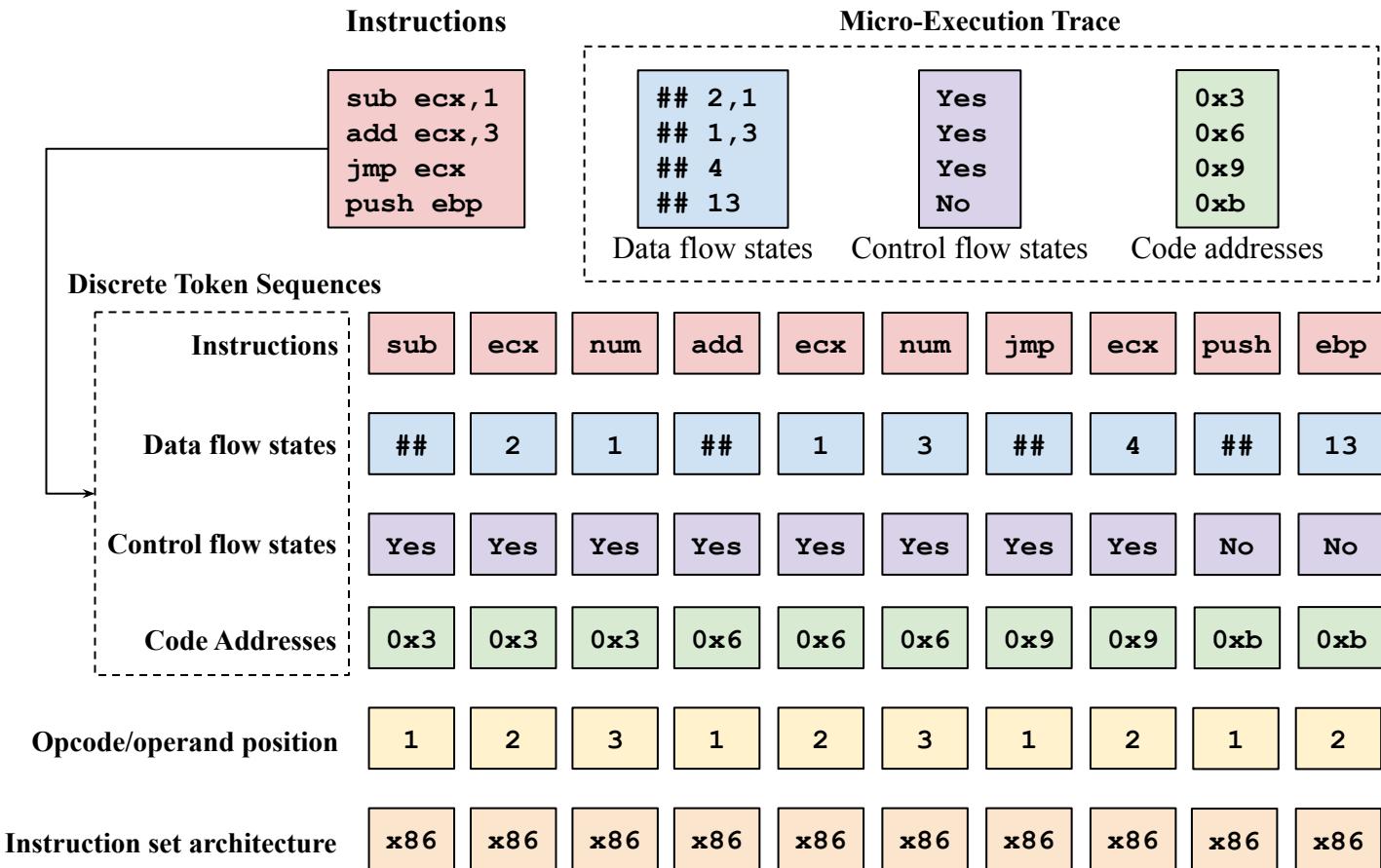
✓ Yes
✓ Yes
✓ Yes
✓ Yes
✗ No
✗ No

Code Addresses

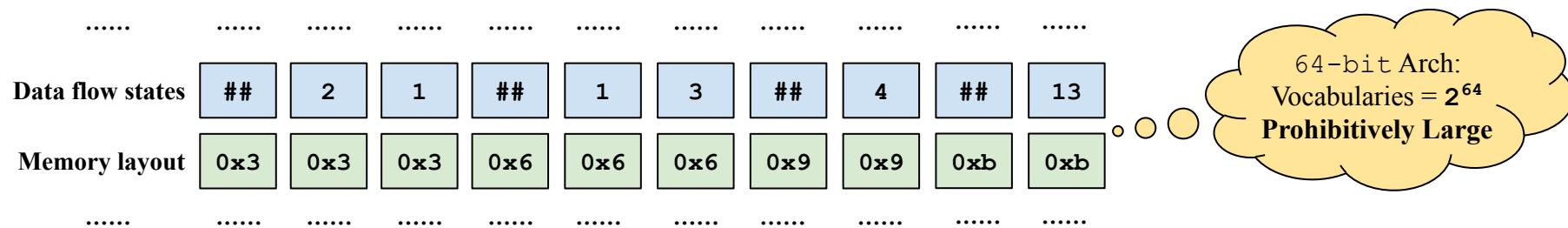
0x1c
0x1f
0x26
0x2d
0x33
0x3a

Aligned with Program Instructions

Input Representation



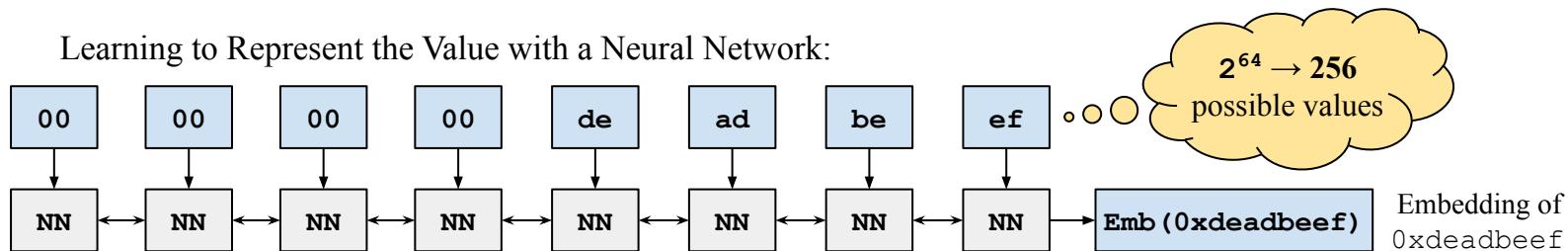
Numerical Representation



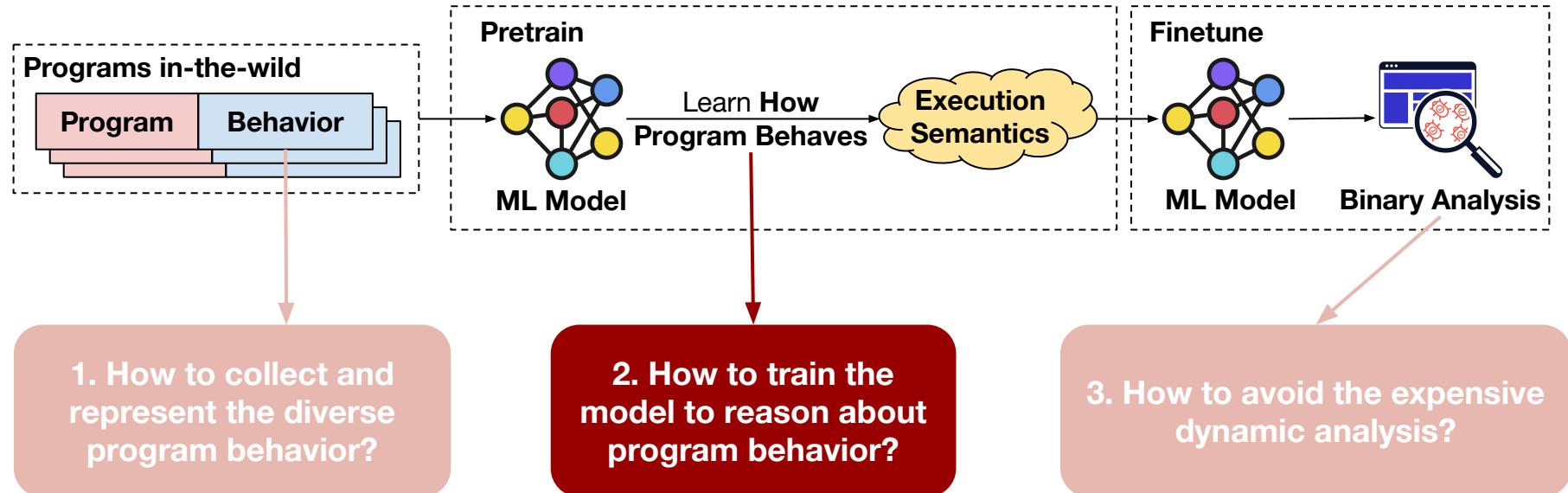
Pad Each Numeric Token as a Fixed-Length 8-Byte Sequence:



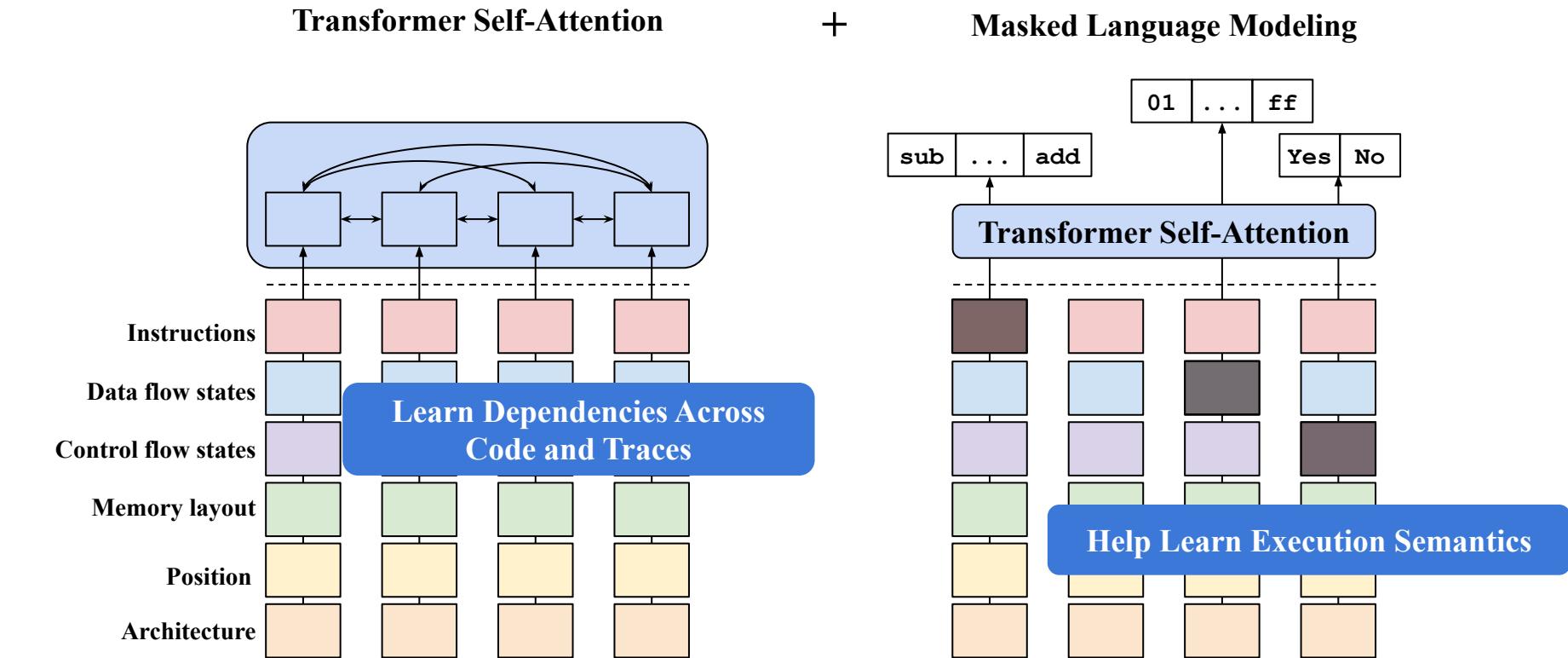
Learning to Represent the Value with a Neural Network:



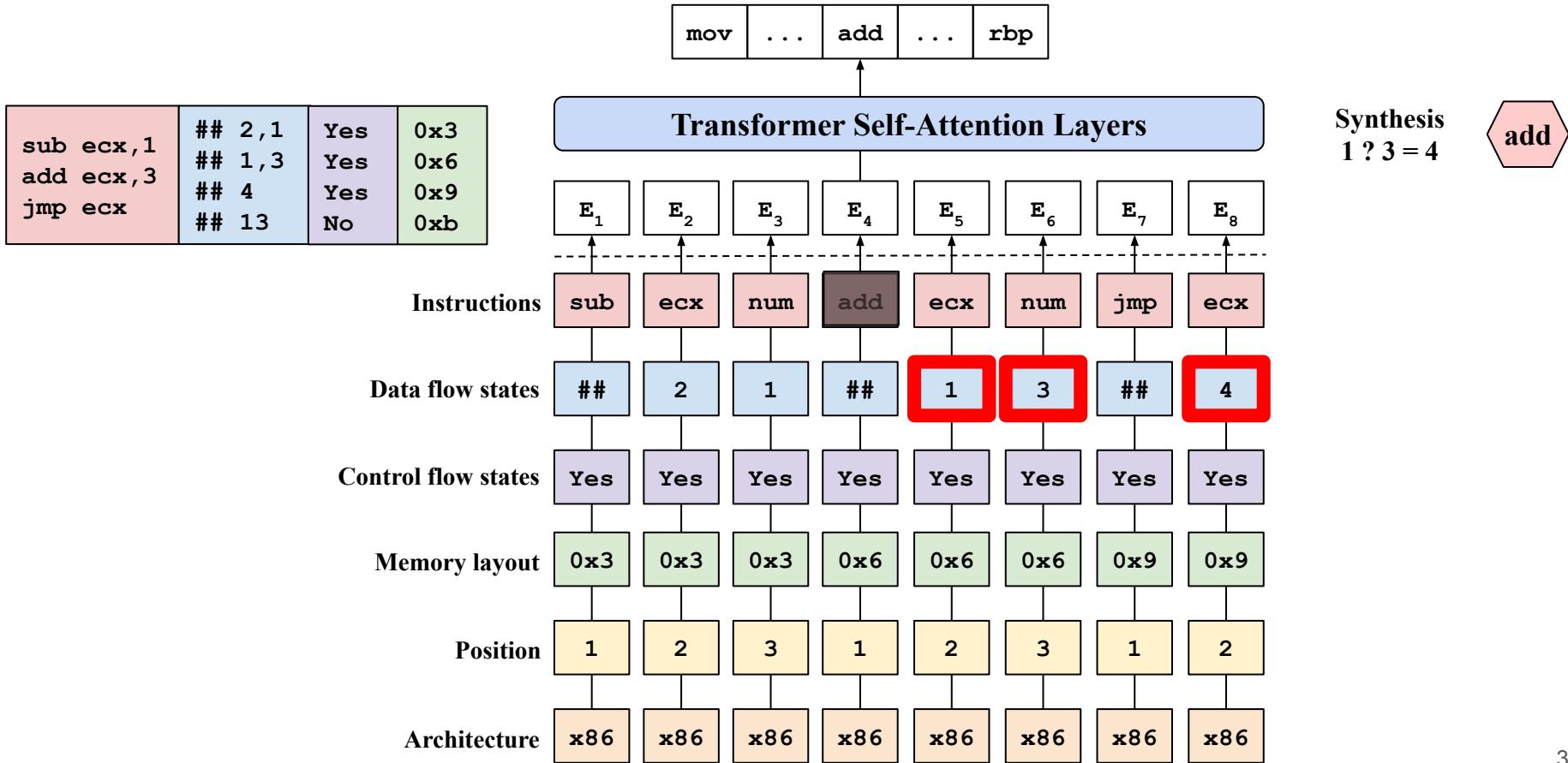
Challenges of Learning Execution Semantics



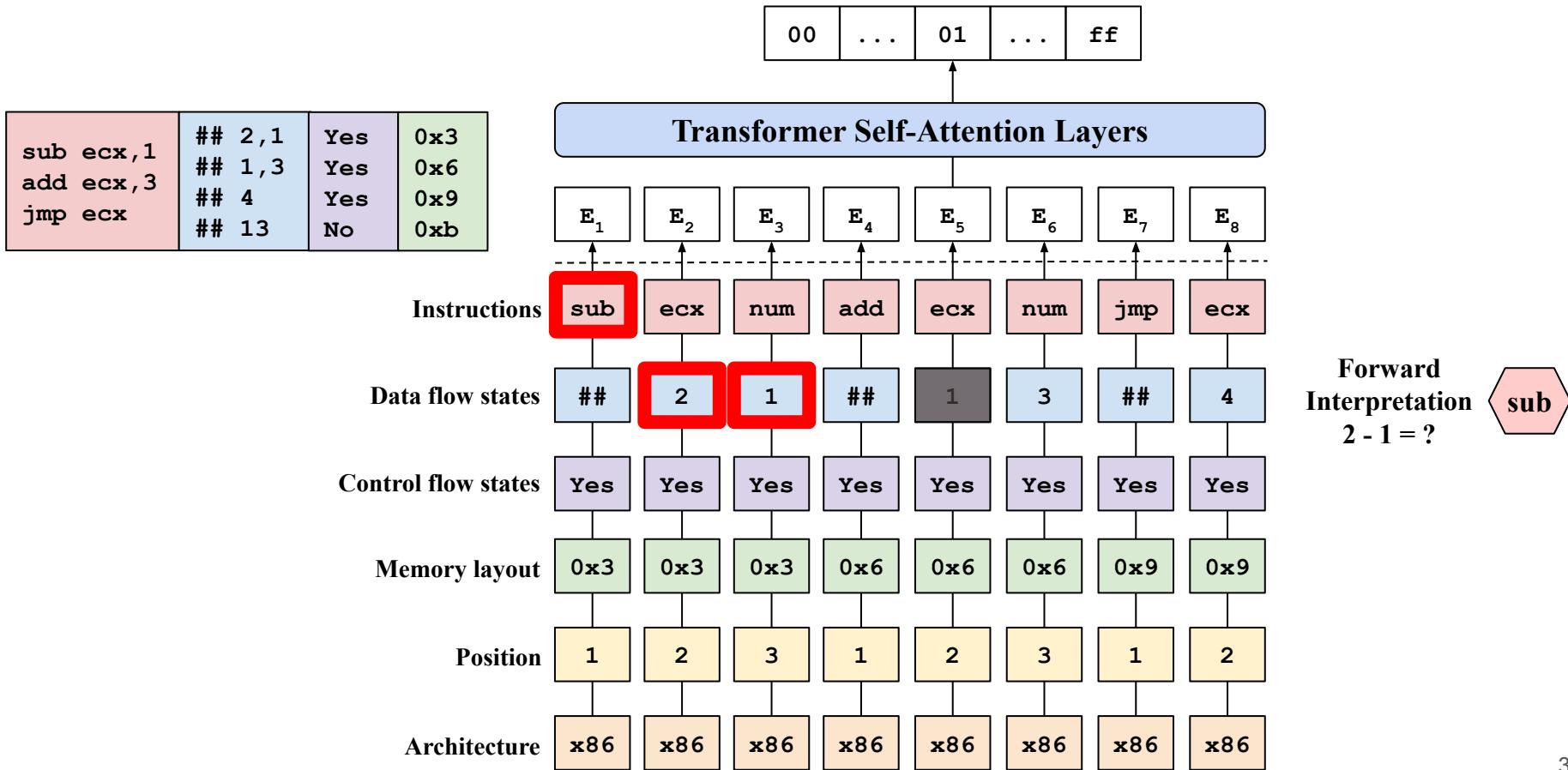
How to train the model to reason about program behavior?



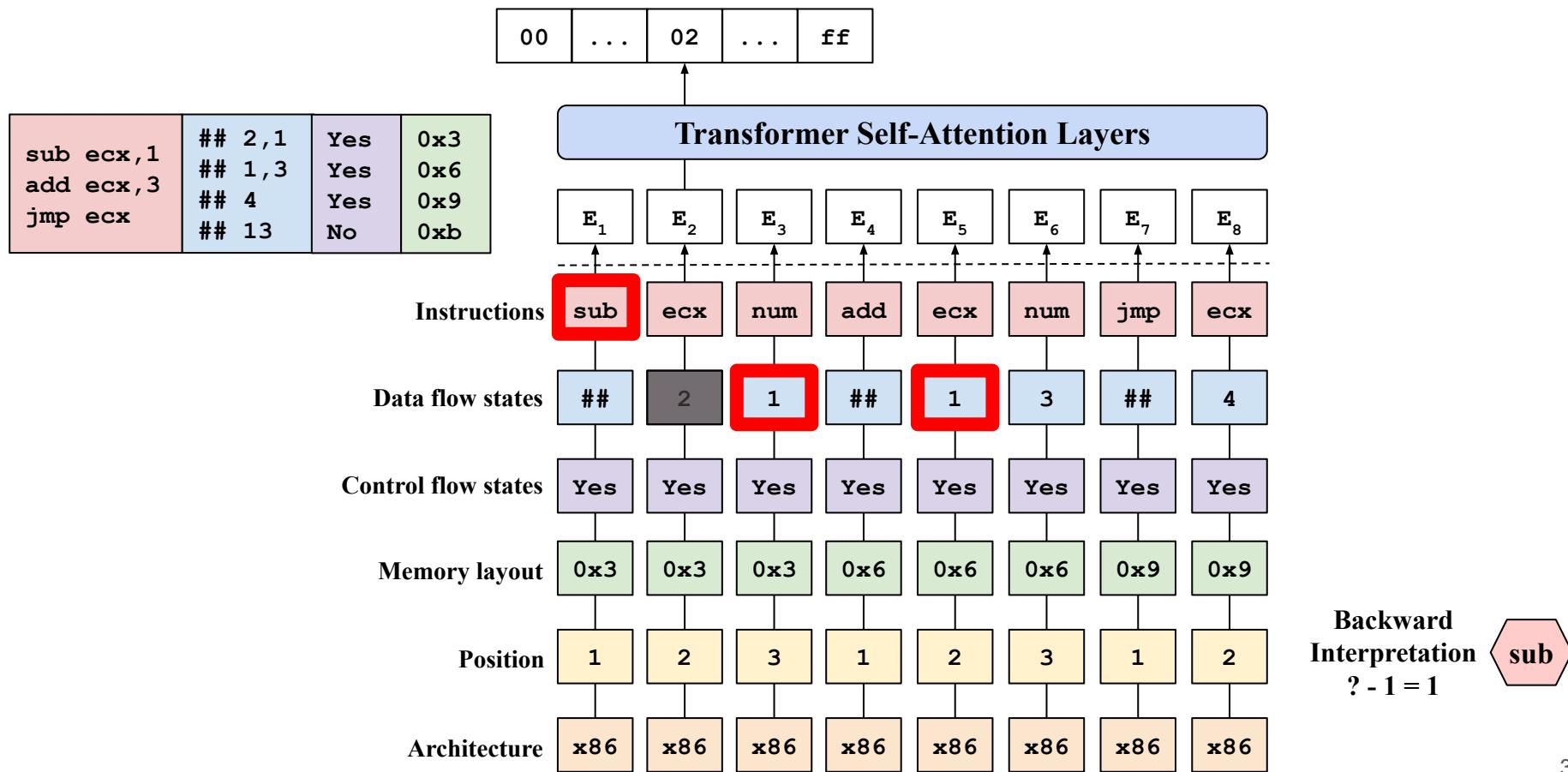
Motivating Example



Motivating Example

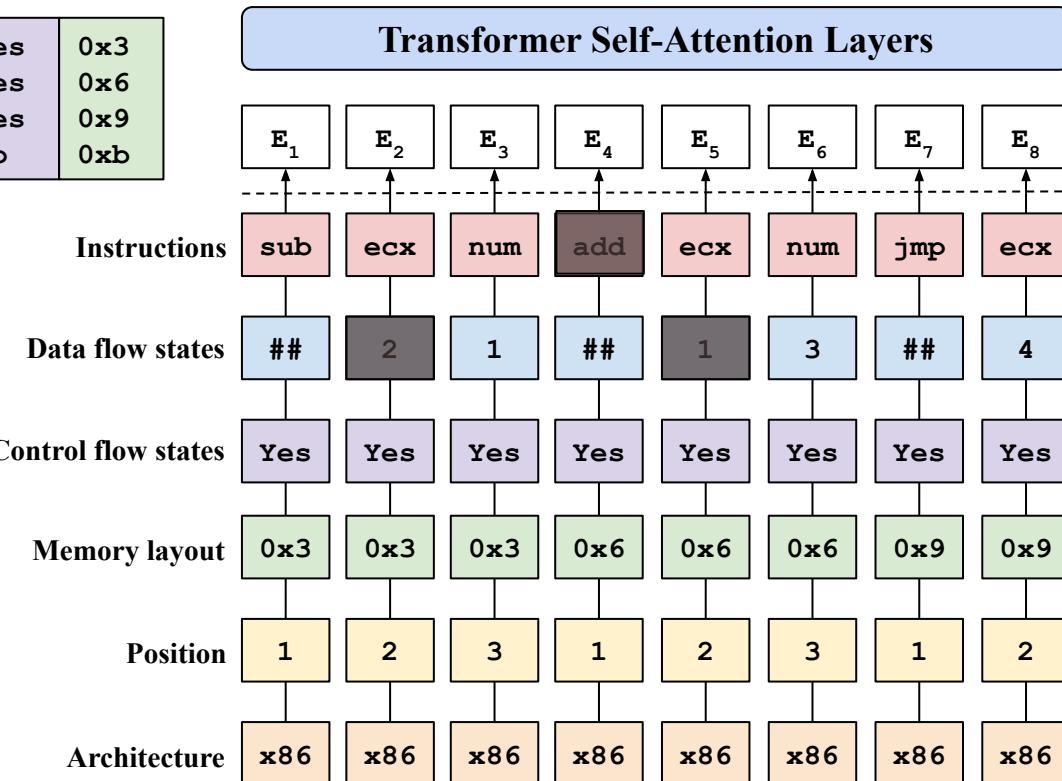


Motivating Example



Motivating Example

sub ecx,1	## 2,1	Yes	0x3
add ecx,3	## 1,3	Yes	0x6
## 4	Yes	0x9	
jmp ecx	## 13	No	0xb



- Program Interpretation
- Program Synthesis

Synthesis
1 ? 3 = 4



Forward
Interpretation
2 - 1 = ?



Backward
Interpretation
? - 1 = 1

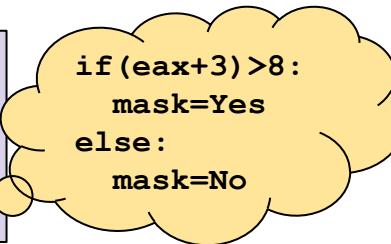


How to train the model to reason about program behavior?

How to train the model to reason about
control flow?

Mask control flow states

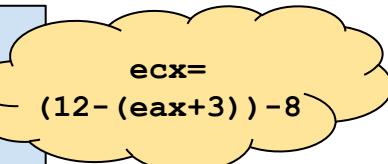
add eax, 3	Yes
cmp eax, 8	Yes
jle 0x8	Yes
mov ecx, ebx	No



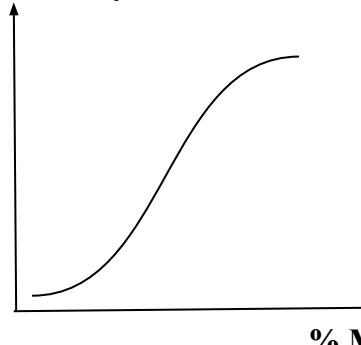
How to train the model to reason about
instruction compositions?

Mask more

add eax, 3	## 3, 3
sub ebx, eax	## 12, 6
sub ebx, 8	## 6, 8
mov ecx, ebx	## 0, -8



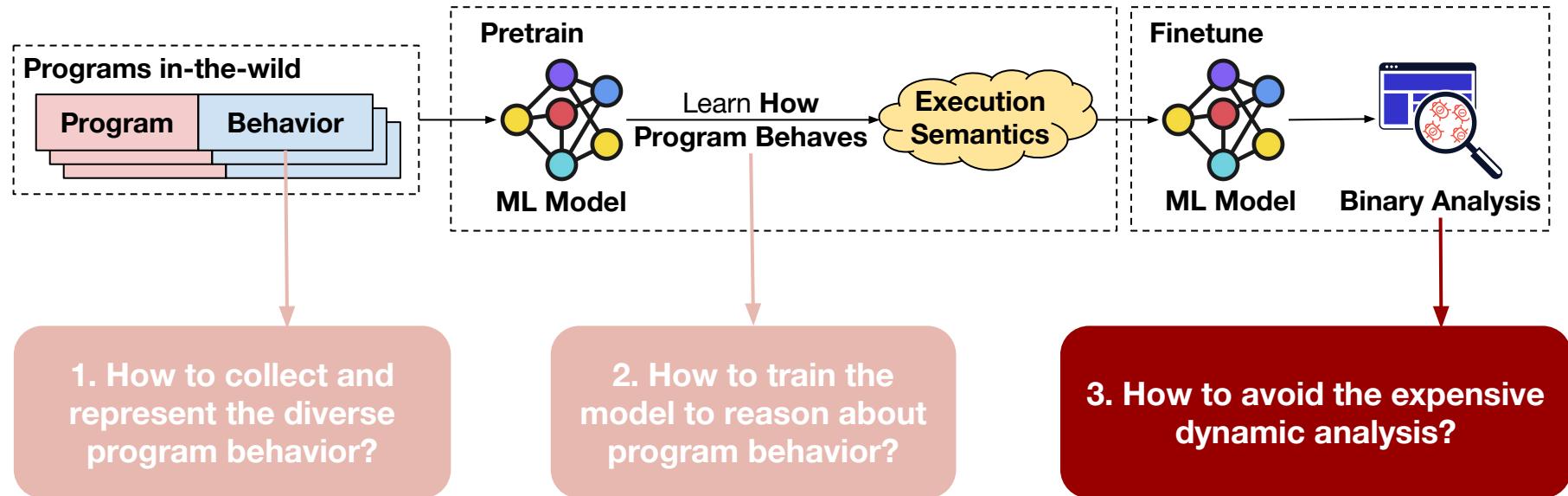
Compositionality



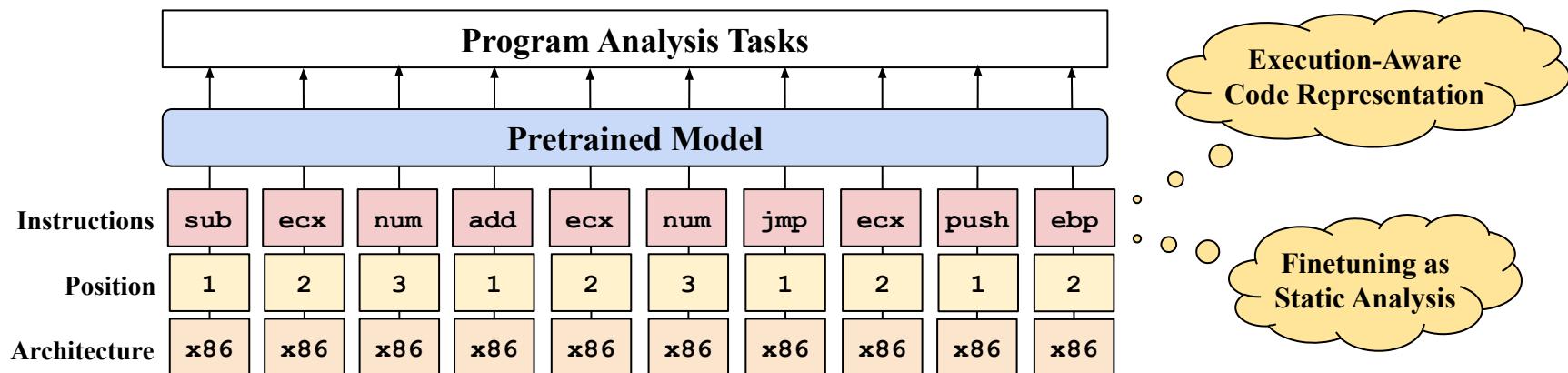
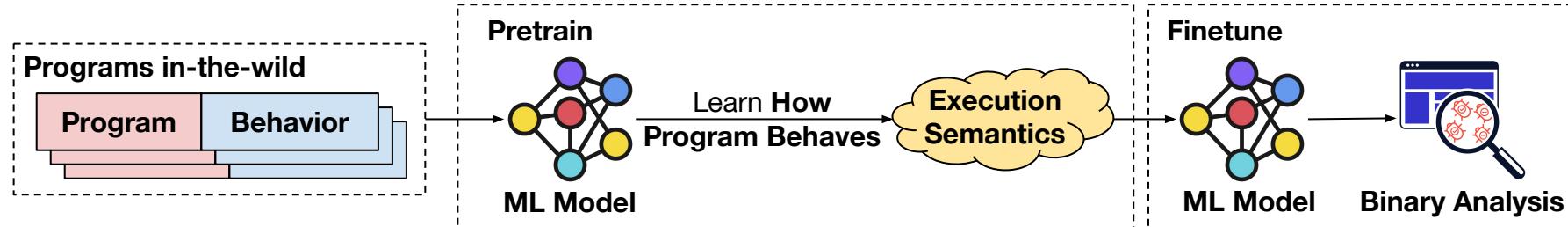
Training

- **Curriculum:** linearly increase % masks by training iterations
- **Randomized** mask selection at each iteration and samples

Challenges of Learning Execution Semantics

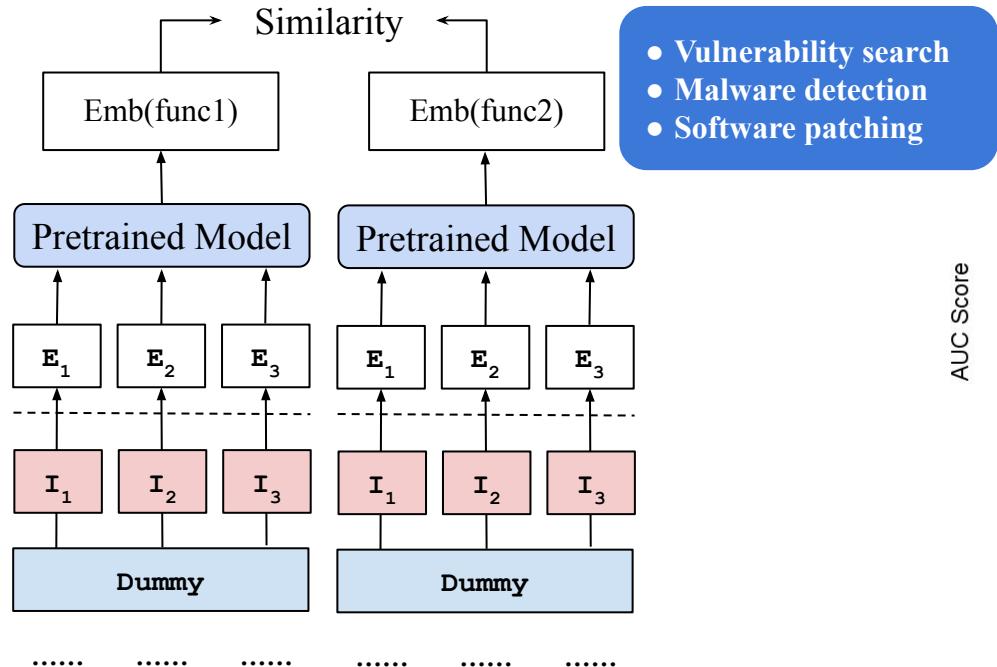


How to Avoid the Expensive Dynamic Analysis?



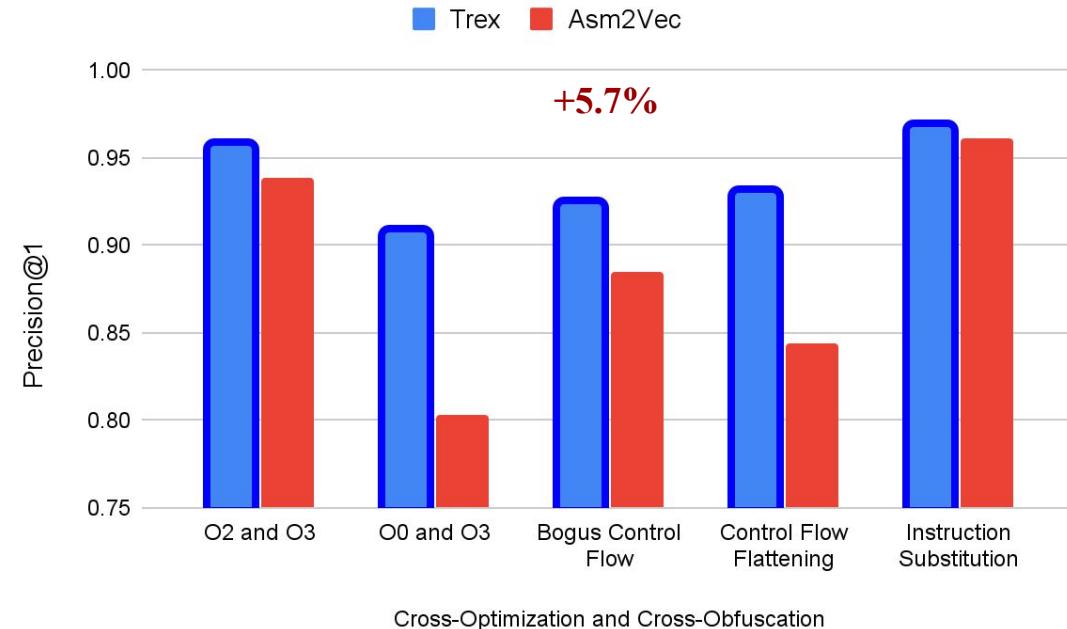
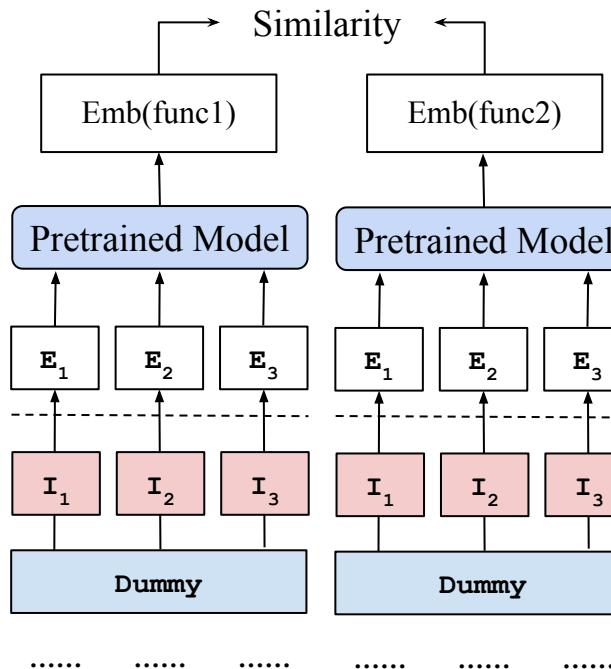
How Much do Learned Execution-Aware Program Representations Help?

Task 1: Binary Similarity



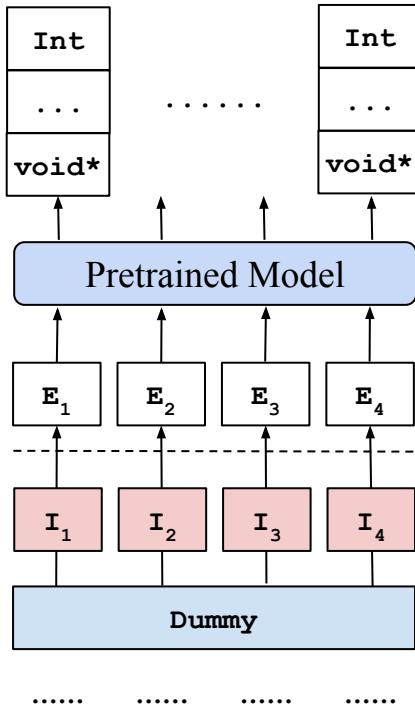
Finetuning for Matching Semantically Similar Binary Functions

Task 1: Binary Similarity



Finetuning for Predicting Function Signatures and Type Inference

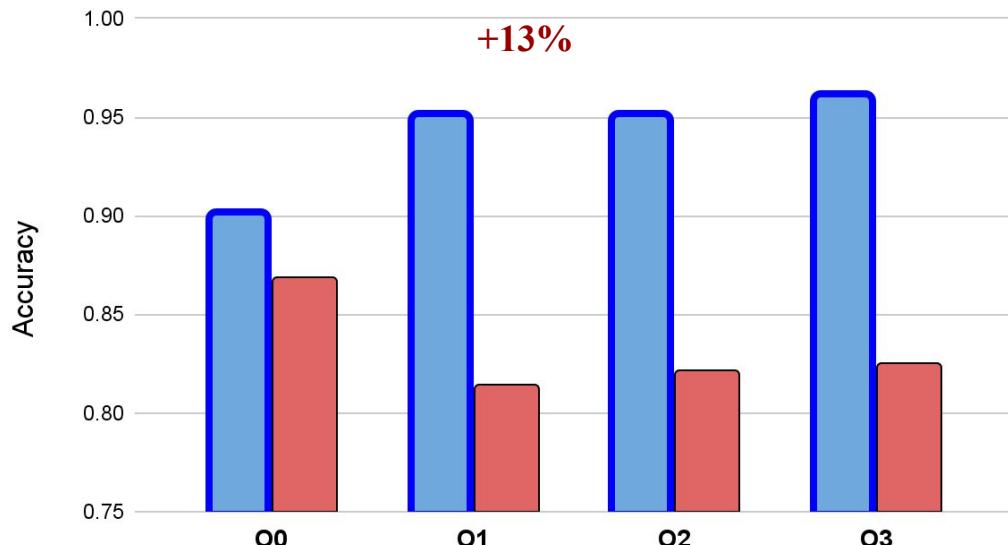
Task 2: Type Inference



- Security retrofitting
- Decompilation
- Vulnerability detection

StateFormer EKLAGYA

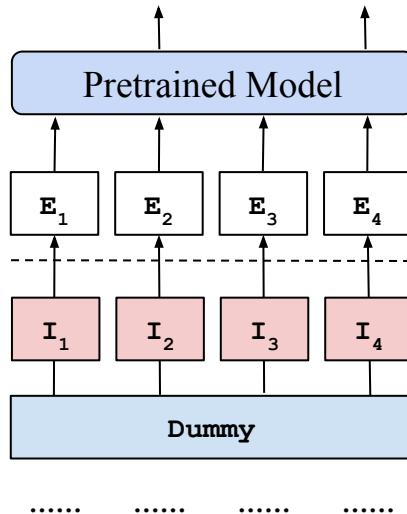
+13%



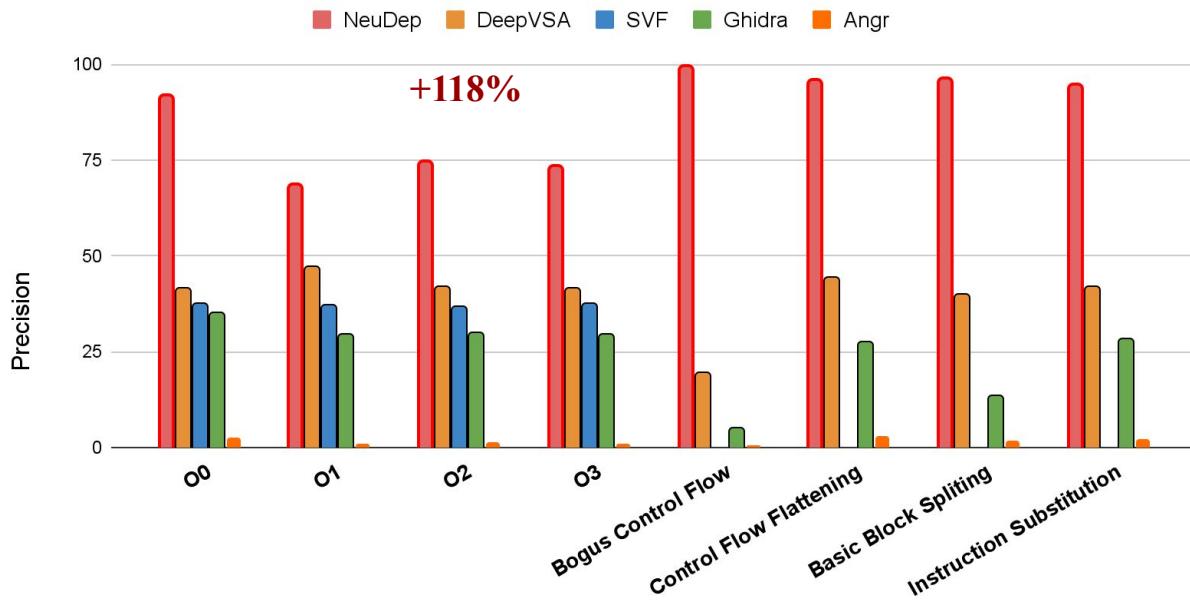
Finetuning for Analyzing Memory Dependence

Task 3: Memory Dependence Analysis

$$P(I_2 \text{ and } I_4 \text{ is dependent}) = 0.8$$



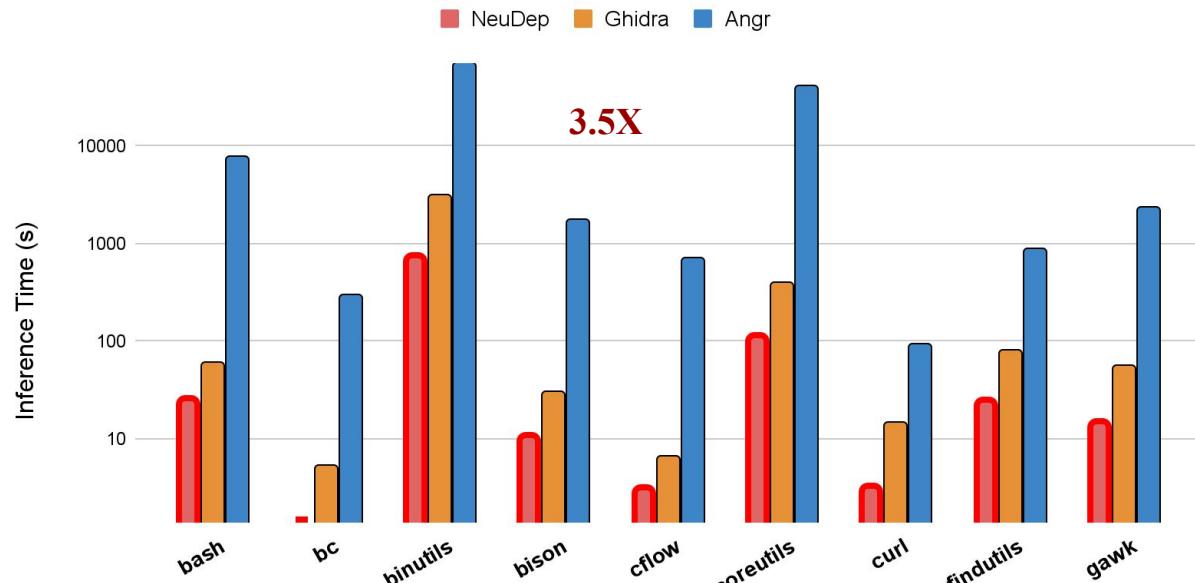
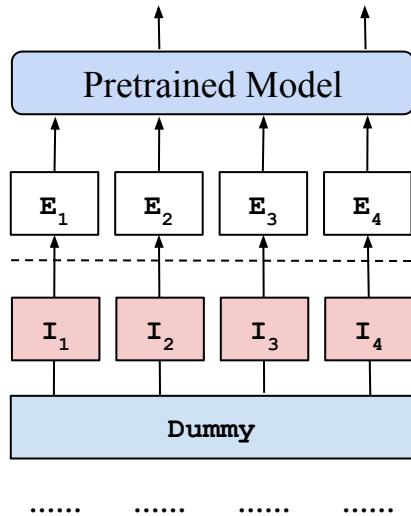
- Taint analysis
- Malware Analysis



Finetuning for Analyzing Memory Dependence: Inference Time

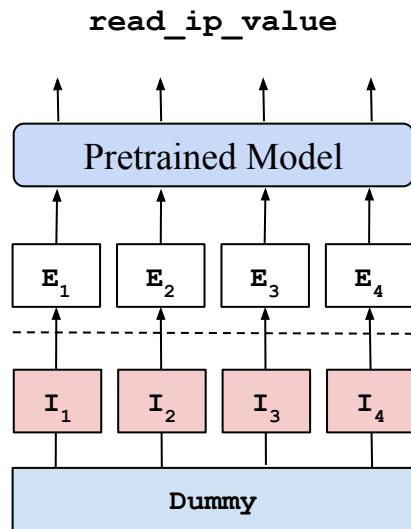
Task 3: Memory Dependence Analysis

$$P(I_2 \text{ and } I_4 \text{ is dependent}) = 0.8$$



Finetuning for Function Name Prediction and Memory Region Prediction

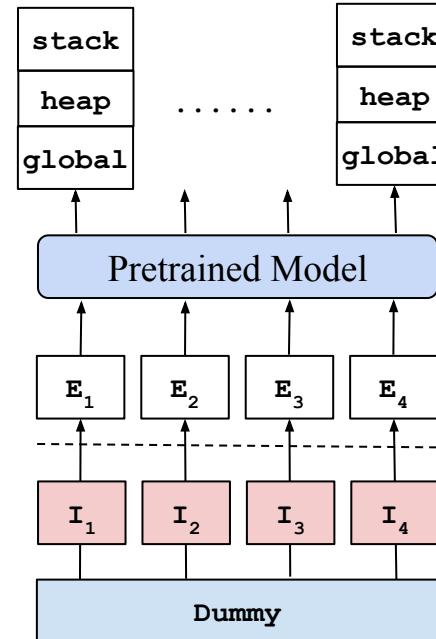
Task 4: Function Name Prediction



- Decompilation
- Reverse engineering

+35% Across All Architectures and Optimizations

Task 5: Memory Region Prediction



- Value set analysis
- Taint analysis

+7.9% Across All Types of Memory Regions

Case Studies: Vulnerability Search in Firmware

CVE	Library	Description
CVE-2019-1563	OpenSSL	Decrypt encrypted message
CVE-2017-16544	BusyBox	Allow executing arbitrary code
CVE-2016-6303	OpenSSL	Integer overflow
CVE-2016-6302	OpenSSL	Allows denial-of-service
CVE-2016-2842	OpenSSL	Allows denial-of-service
CVE-2016-2182	OpenSSL	Allows denial-of-service
CVE-2016-2180	OpenSSL	Out-of-bounds read
CVE-2016-2178	OpenSSL	Leak DSA private key
CVE-2016-2176	OpenSSL	Buffer over-read
CVE-2016-2109	OpenSSL	Allows denial-of-service
CVE-2016-2106	OpenSSL	Integer overflow
CVE-2016-2105	OpenSSL	Integer overflow
CVE-2016-0799	OpenSSL	Out-of-bounds read
CVE-2016-0798	OpenSSL	Allows denial-of-service
CVE-2016-0797	OpenSSL	NULL pointer dereference
CVE-2016-0705	OpenSSL	Memory corruption

16 Vulnerabilities (Compiled in x86)

Search

Firmware Images: Arm, MIPS with unknown compiler flags



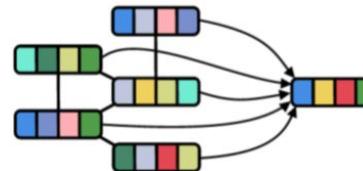
Ubiquiti sunMax TP-Link Deco-M4 NETGEAR R7000 Linksys RE7000

15 CVEs

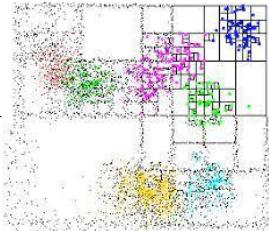
16 CVEs

7 CVEs

14 CVEs



Learned Function Embeddings



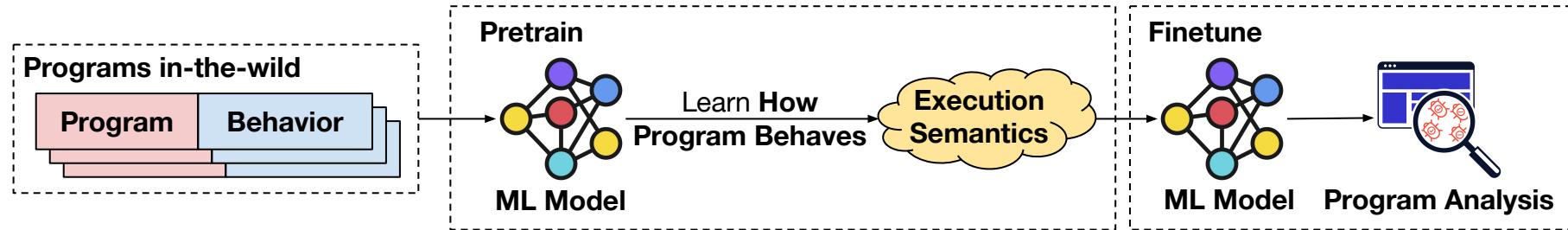
Approximate Nearest Neighbors



Search over **1.4M** functions within
6.3 seconds

Summary: Learning Program Semantics via Execution-Aware Pre-training

Improves Program Analysis for Security Applications



Precise: Outperforms the state-of-the-art by up to **118%**

Efficient: Speedup over the off-the-shelf tool by up to **98.1x**

Generalizable and Robust: Remains accurate across



Compilers



Architectures

-00 -01 -02
-03 -0d 0x



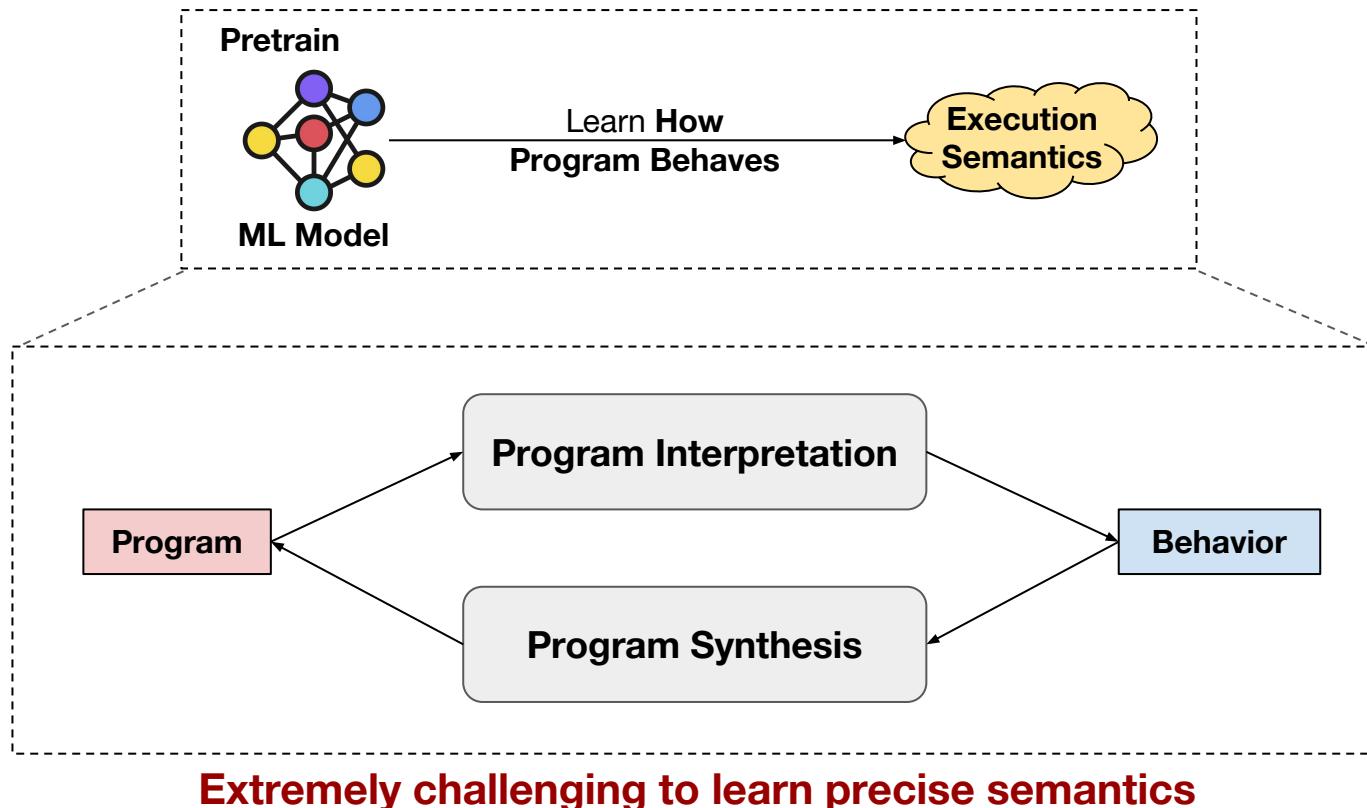
Optimizations



Obfuscations



Limitation: Learning Execution-Aware Program Representations is Challenging



Limitation: What the Model has Learned during Pretraining?

Instructions	Dataflow states
.....
sub ecx, 0x3	## 0x5, 0x3
add ecx, 0x4	## 0x2, 0x4
.....

Instructions	Dataflow states
.....
sub ecx, 0x3	## 0x43, 0x3
add ecx, 0x4	## 0x3d, 0x4
.....

Perturb dataflow states from 0x5 to 0x43

Ground-truth	Top-1	Top-2
0x2	0x2 (98%)	0x3 (2%)
0x3d	0x3a (28%)	0x33 (13%)

Does not extrapolate well

Machine Learning Security

