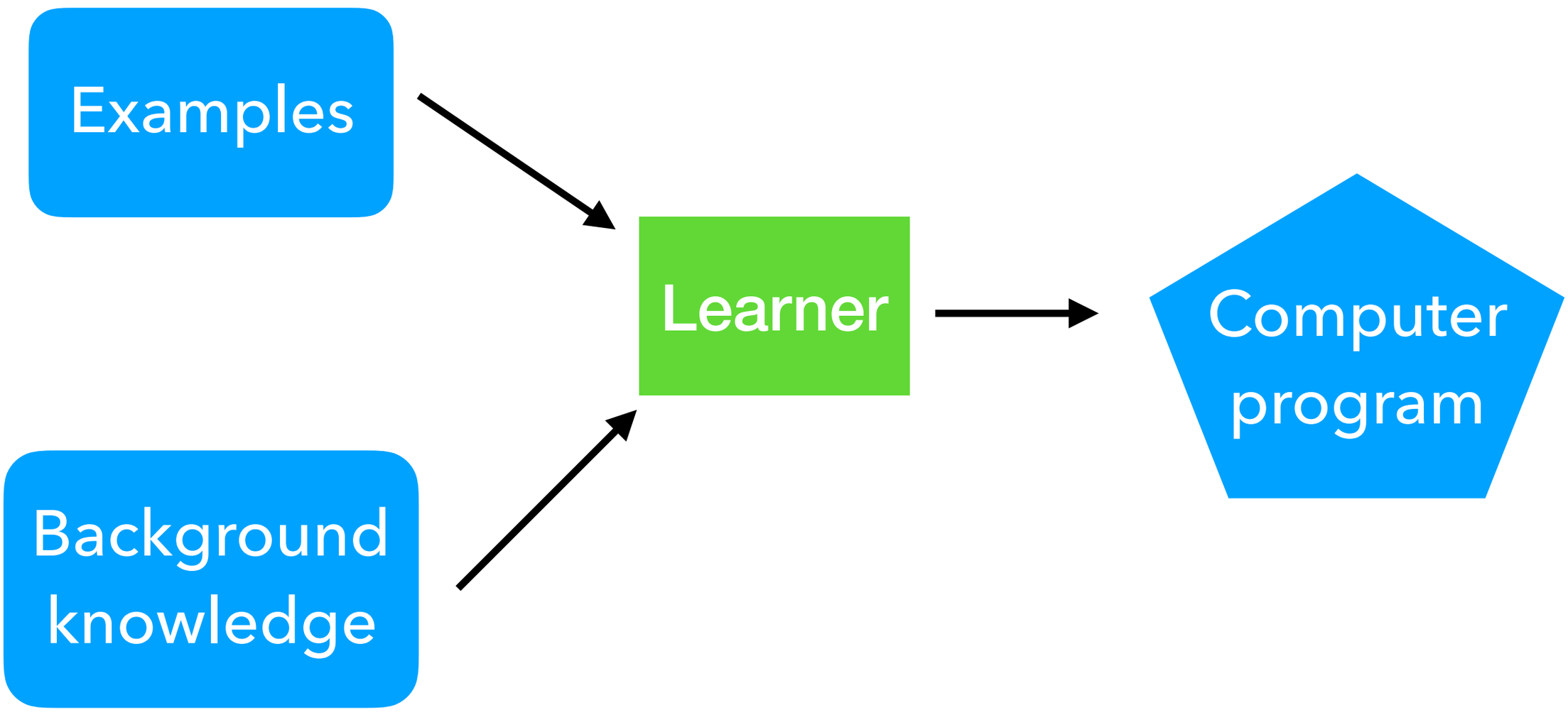
Playgol: learning programs through play

Program induction



Examples

input	output
dog	g
sheep	p
chicken	**?**

Examples

input	output
dog	g
sheep	p
chicken	**?**

Background knowledge

- head/2
- tail/2
- empty/1

Examples

input	output
dog	g
sheep	p
chicken	**n**

Background knowledge

- head/2
- tail/2
- empty/1

```
f(A,B):-tail(A,C),empty(C),head(A,B).
f(A,B):-tail(A,C),f(C,B).
```

Where do we get background knowledge from?

Hand-crafted rules

Supervised multi-task learning

[Lin et al. ECAI14]

[Ellis et al. NIPS18]

Unsupervised learning

[Dumancic et al. IJCAI19]

Self-supervised play

Playgol

1. Play (self-supervised)
2. Build (supervised)

Playing

1. Sample random tasks from the instance space
2. Learn programs to them
3. Add programs to the BK

```
def play(instance_space,bk,playtime,max_depth):
  play_tasks = sample(instance_space,p)
  for d to max_depth:
    solved,new_bk = learn(play_tasks,bk,max_depth)
    play_tasks = play_tasks - solved
    bk = bk + new_bk
return bk
```

```
def learn(play_tasks, bk, depth):
  solved = []
  programs = []
  for task in play_tasks:
    program = metagol(bk, task, max_depth)
    if program != null:
      programs = programs + program
      solved = solved + task
return solved, programs
```

Building

Solve user-supplied tasks using the augmented BK

Why should it work?

We increase branching but reduce depth

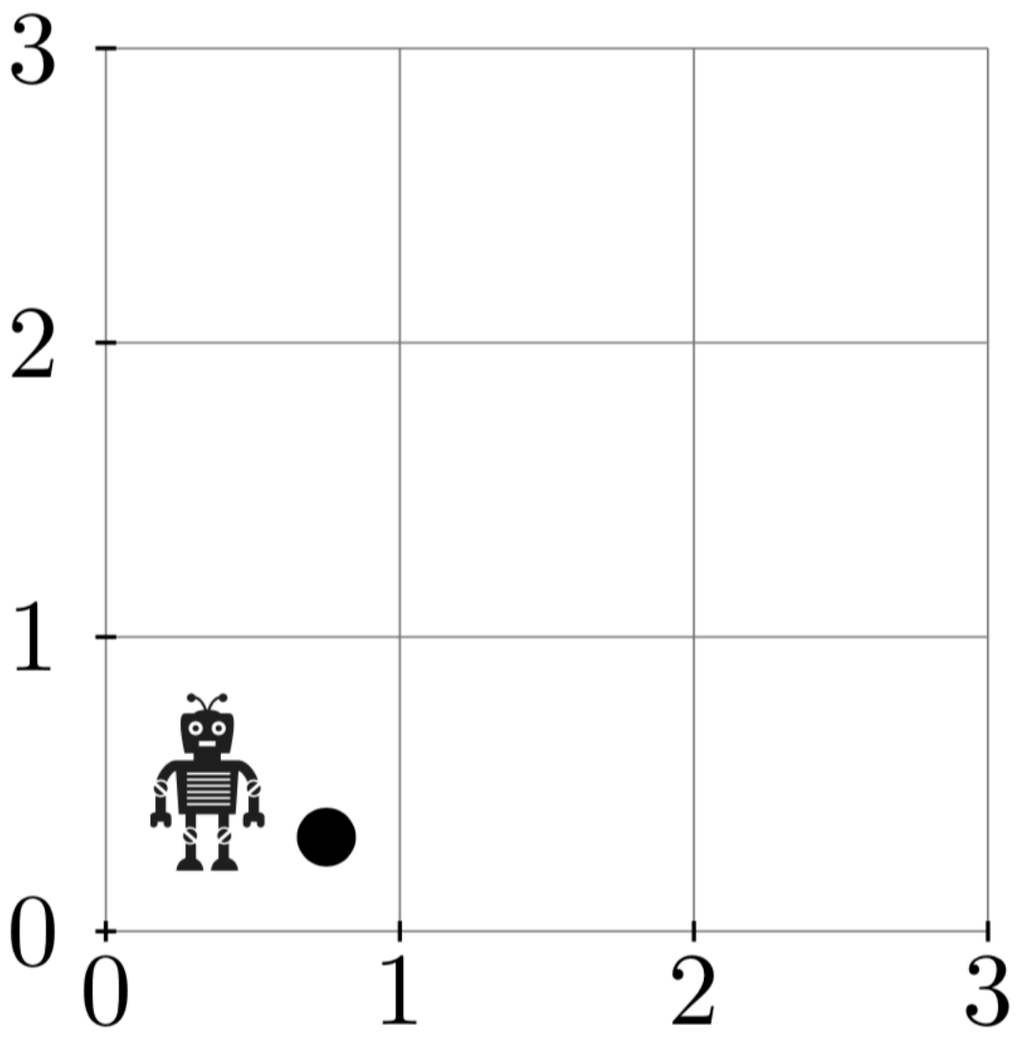
Does it work?

Q1. Can playing improve performance?

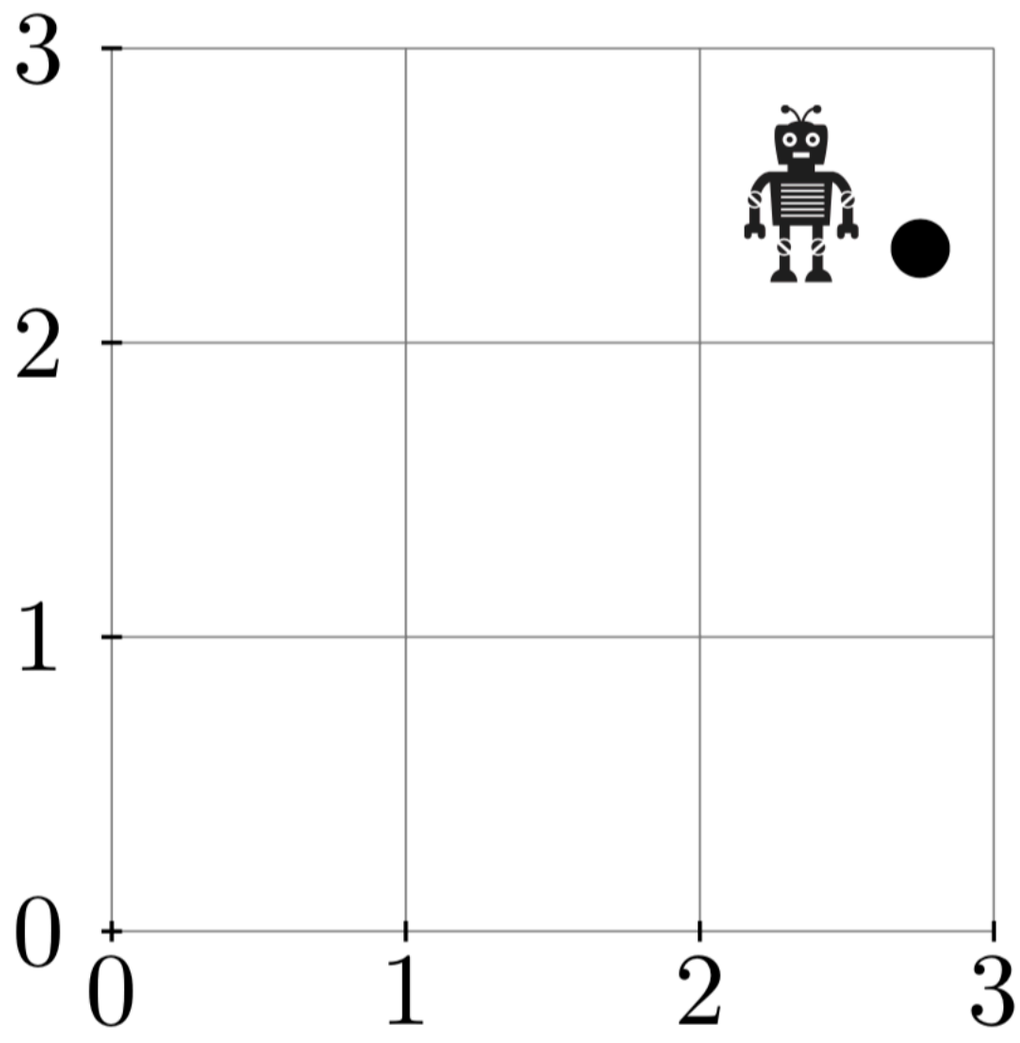
Q2. Can playing improve performance without many play tasks?

Q3. Can predicate invention improve performance?

Robot planning



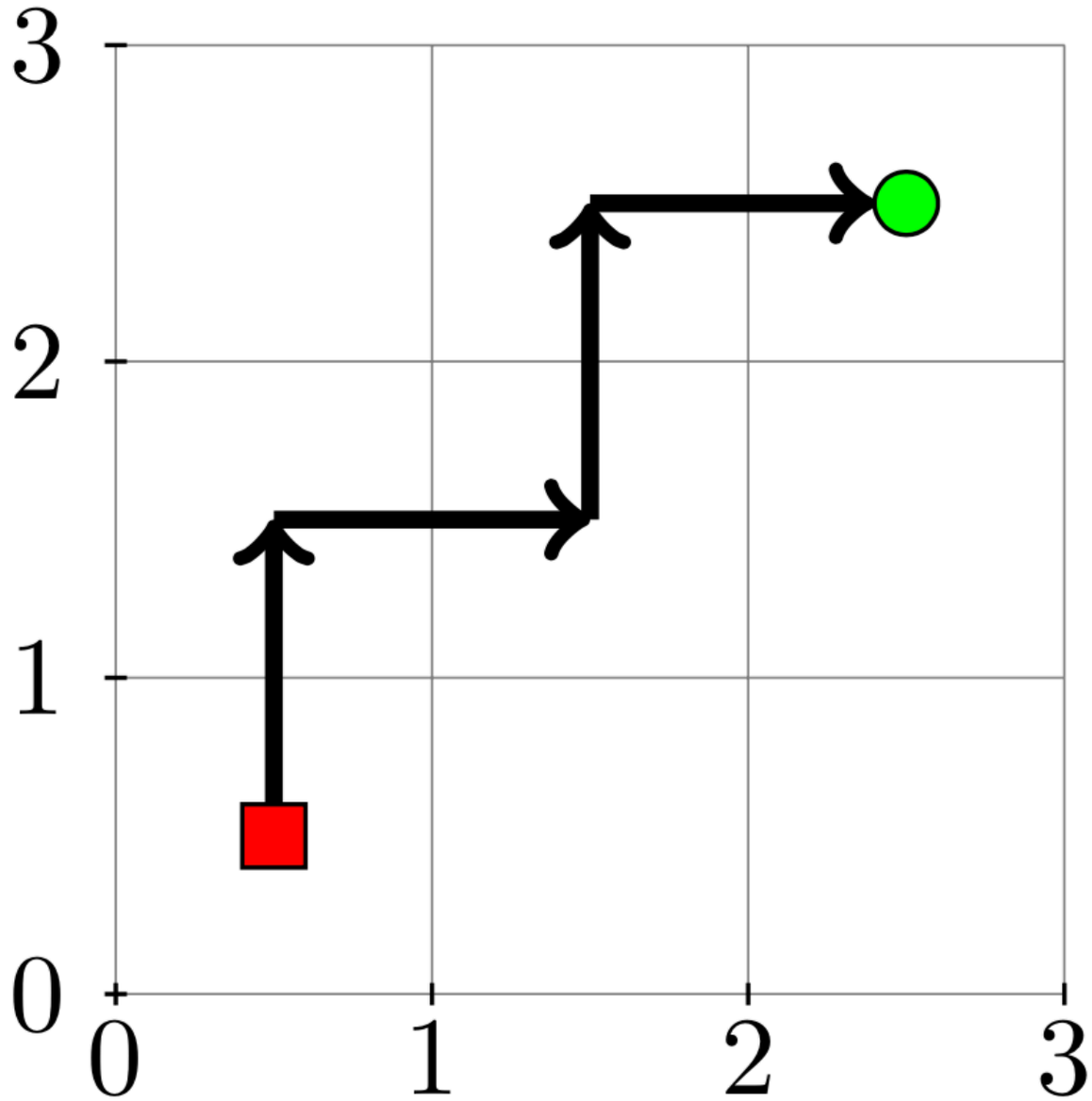
(a) Initial state



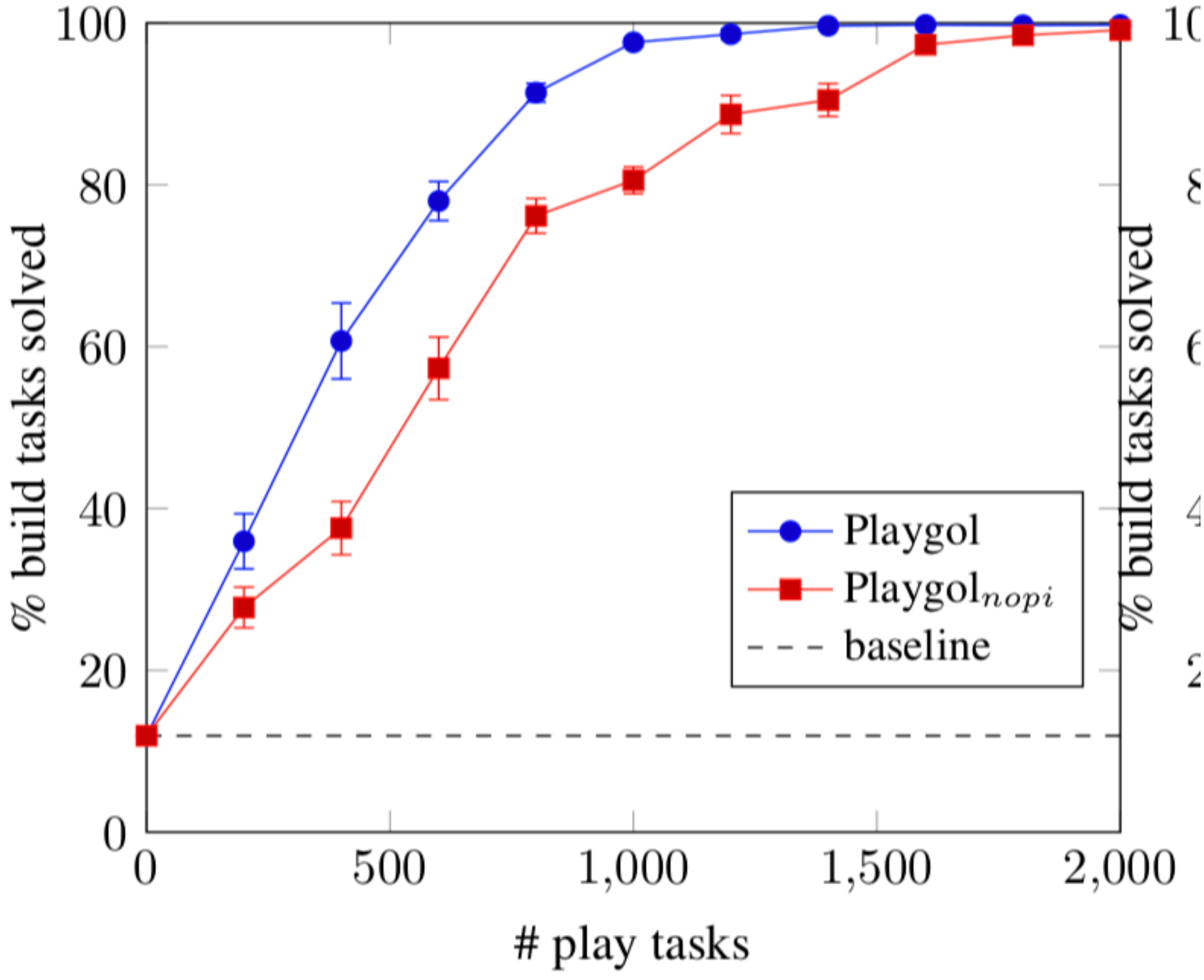
(b) Final state

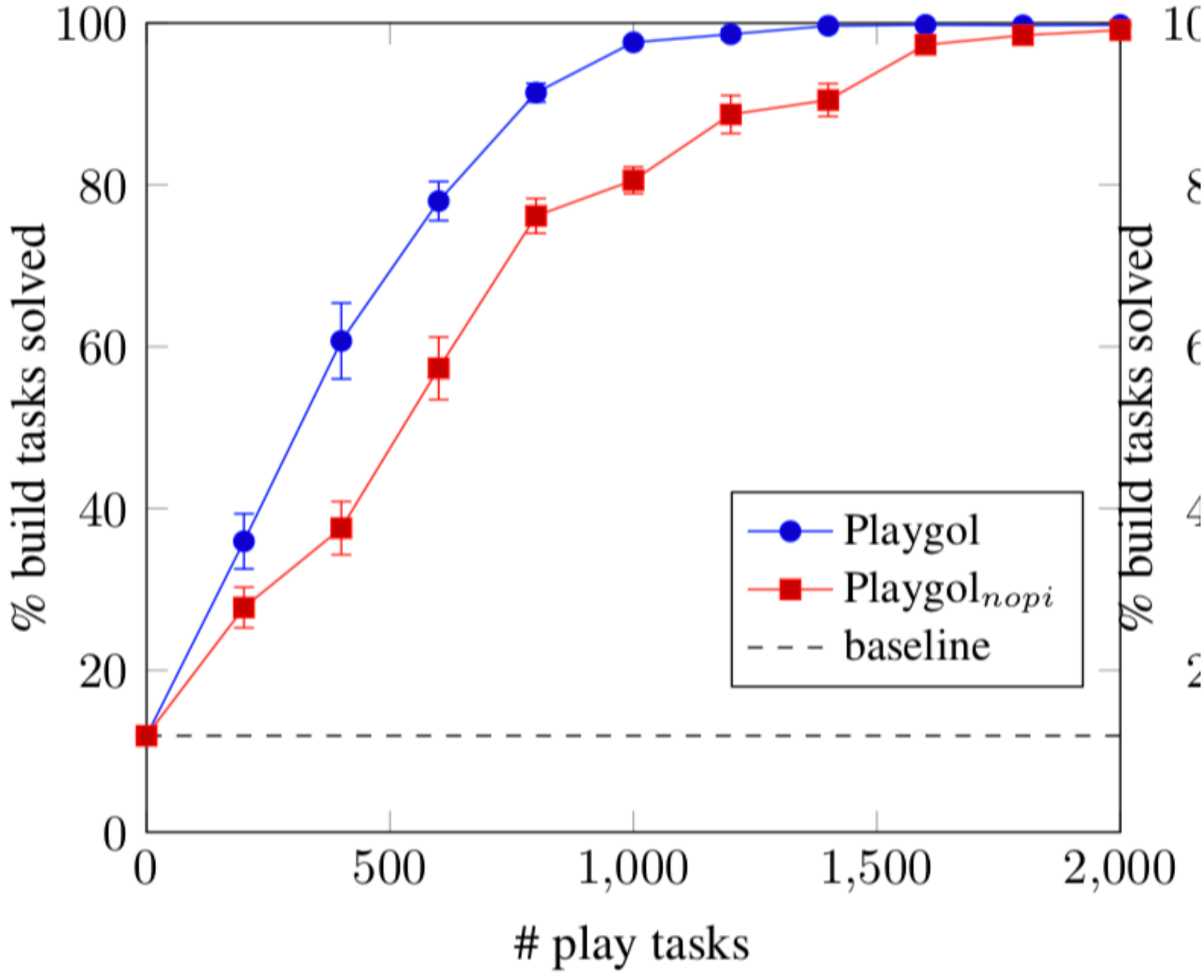
```
f(A,B):-
      grab(A,C),
      f1(C,D),
      f1(D,E),
      drop(E,B).
f1(A,B):-
      up(A,C),
      right(C,B).
```

(a) Program

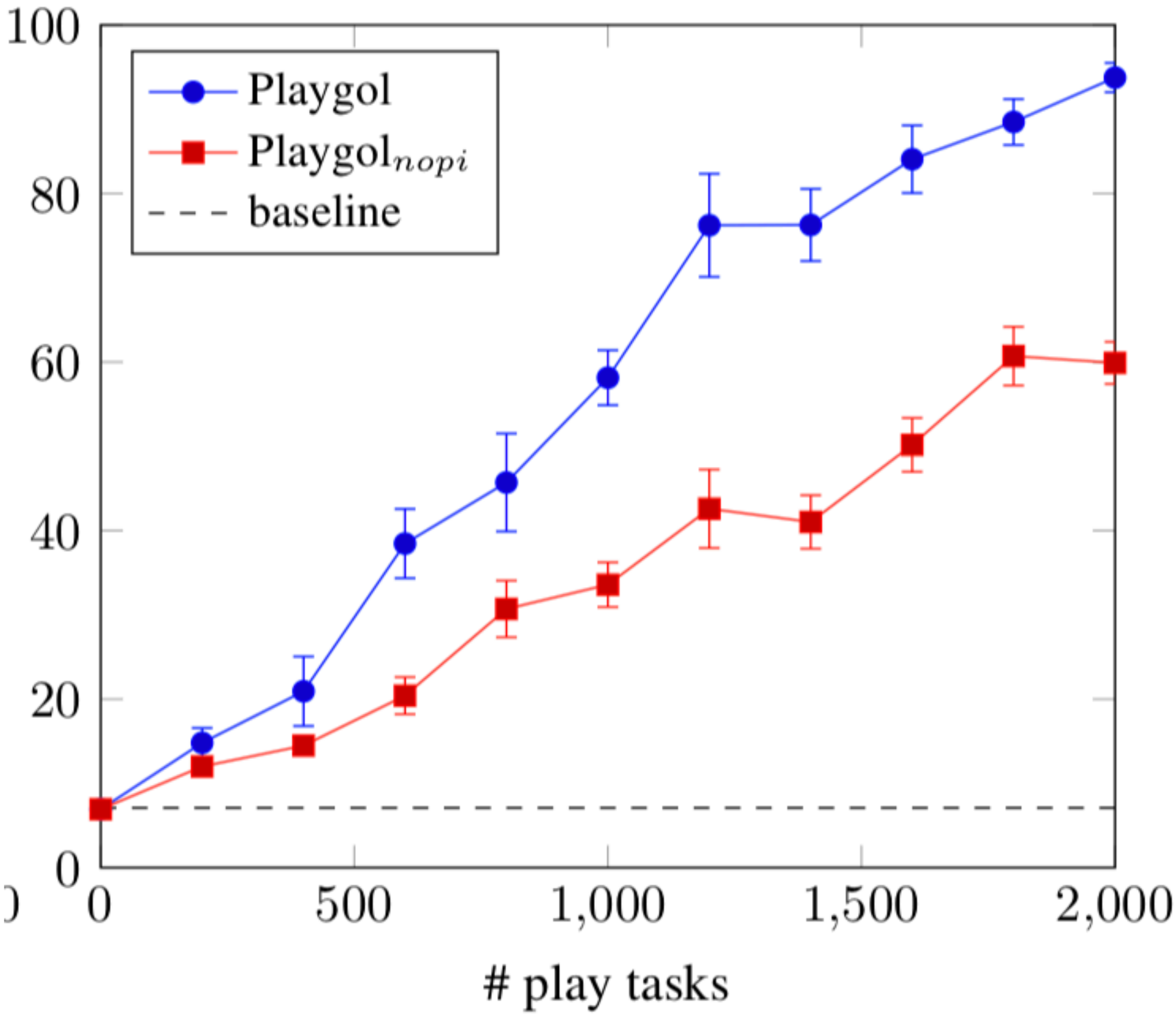


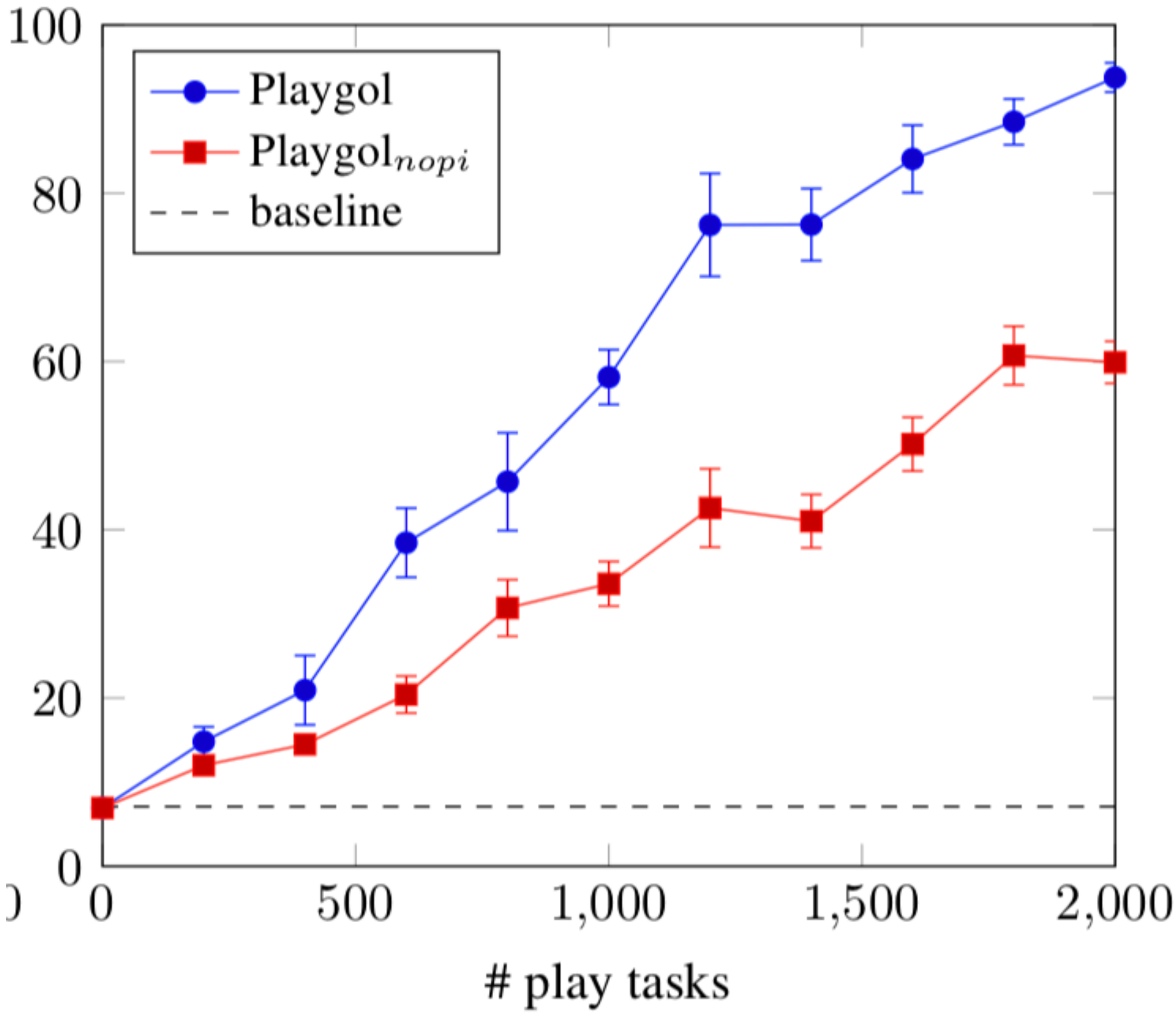
(b) Plan


% build tasks solved
100
80
60
40
20
0
0
500
1,000
1,500
2,000
play tasks
Playgol
Playgol$_{nopi}$
baseline



2000 << 5,000,000


100
80
60
40
20
0
0
500
1,000
1,500
2,000
Playgol
Playgol$_{nopi}$
baseline
play tasks



We should need to sample 24,000,000 play tasks

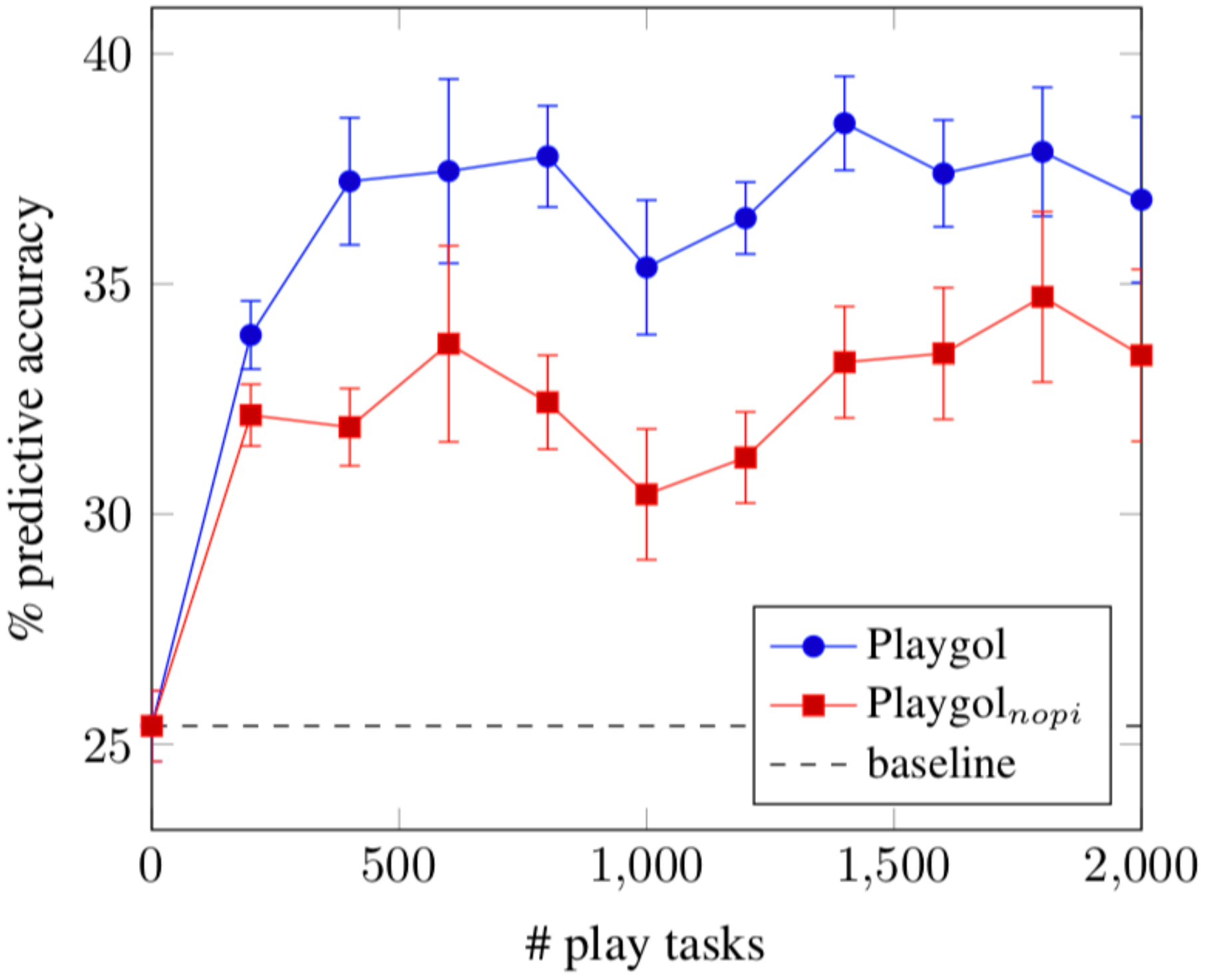
String transformations

Real-world build tasks

Input	Output
22 July,1983 (35 years old)	JUL
30 October,1955 (63 years old)	OCT
2 November,1954 (64 years old)	NOV

Play tasks

Task	Input	Output
play_9	.f\73\R)	F
play_52	@B4\X¿3MjKdyZzC	B
play_136	9pfy”ktfbS1v	99PF
play_228	I6zihQk-	Q


% predictive accuracy
40
35
30
25
0
500
1,000
1,500
2,000
play tasks
Playgol
Playgol$_{nopi}$
baseline

Input	Output
22 July,1983 (35 years old)	JUL
30 October,1955 (63 years old)	OCT
2 November,1954 (64 years old)	NOV

```
build_95(A,B):-play_228(A,C),play_136_1(C,B).
play_228(A,B):-play_52(A,B),uppercase(B).
play_228(A,B):-skip1(A,C),play_228(C,B).
play_136_1(A,B):-play_9(A,C),mk_uppercase(C,B).
play_9(A,B):-skip1(A,C),mk_uppercase(C,B).
play_52(A,B):-skip1(A,C),copy1(C,B).
```

```
build_95(A,B):-play_228(A,C),play_136_1(C,B).
```

Task	Input	Output
228	I6zihQk-	Q

```
play_228(A,B):-play_52(A,B),uppercase(B).
play_228(A,B):-skip1(A,C),play_228(C,B).
```

Task	Input	Output
228	I6zihQk-	Q
52	@B4\X¿3MjKdyZzC	B

```
play_228(A,B):-play_52(A,B),uppercase(B).
play_228(A,B):-skip1(A,C),play_228(C,B).
play_52(A,B):-skip1(A,C),copy1(C,B).
```

Task	Input	Output
228	I6zihQk-	Q
52	@B4\X¿3MjKdyZzC	B

skip_to_uppercase_and_copy

```
build_95(A,B):-
skip_to_uppercase_and_copy(A,C),
play_136_1(C,B).
```

```
play_136_1(A,B):-play_9(A,C),mk_uppercase(C,B).
play_9(A,B):-skip1(A,C),mk_uppercase(C,B).
```

```
play_136_1(A,B):-
    skip1(A,C),
    mk_uppercase(C,D),
    mk_uppercase(D,B).
```

```
play_136_1(A,B):-
    skip1(A,C),
    mk_uppercase(C,D),
    mk_uppercase(D,B).
```

```
build_95(A,B):-
    skip_to_uppercase_and_copy(A,C),
    skip1(A,C),
    mk_uppercase(C,D),
    mk_uppercase(D,B).
```

Conclusions

Playing allows an ILP system to self-discover reusable programs

Limitations and future work

Need to define instance space

When does it work?

Better curious sampling

Forgetting methods (lots of BK)