Intelligence = Sampling and Filtering

Presented by:
John Tan Chong Min

Drawing on Lessons From:

AlphaCode – DeepMind (Li et al, 2022)

GPT-4o on ARC (Ryan Greenblatt, 2024)

Learning, Fast and Slow (John and Motani, 2023)

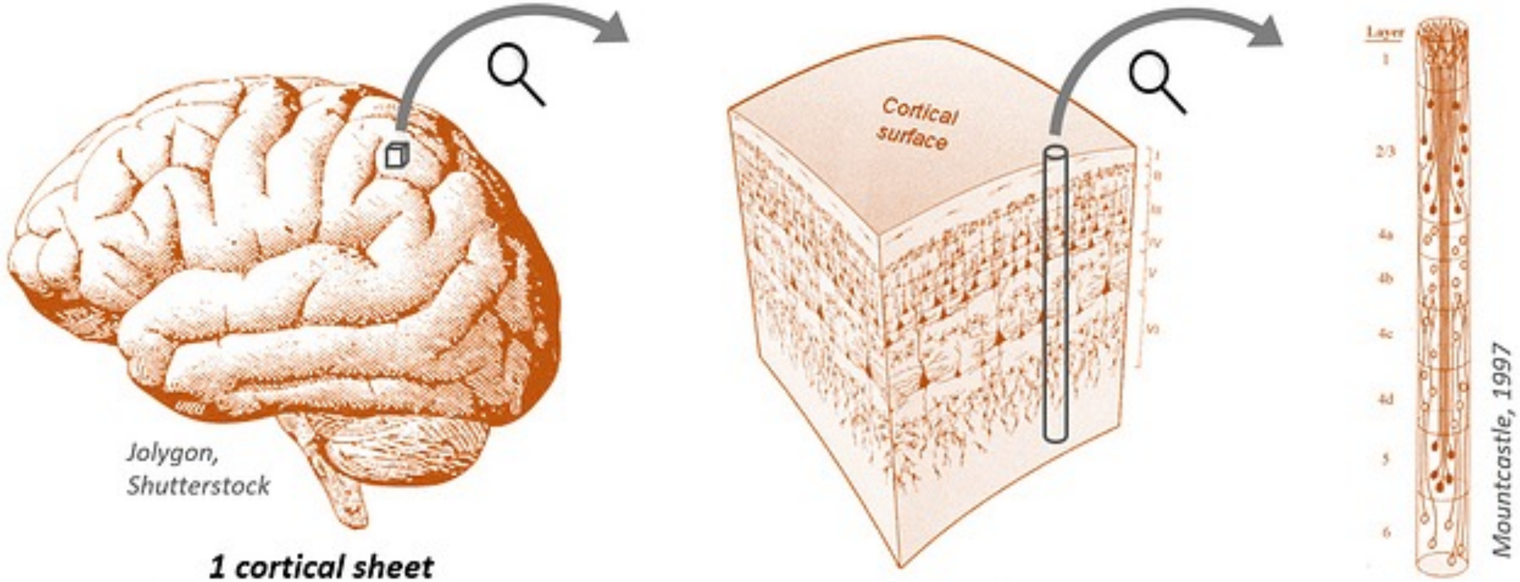
LLMs as a System of Multiple Expert Agents to solve ARC (John and Motani, 2023)

AlphaGo / AlphaZero (Silver et al, 2017)

LLAMA-3 8B with MCTS (Zhang et al, 2024)


Imagine looking down from
the top of a cliff
What do you feel?

Minicolumns are plentiful in humans



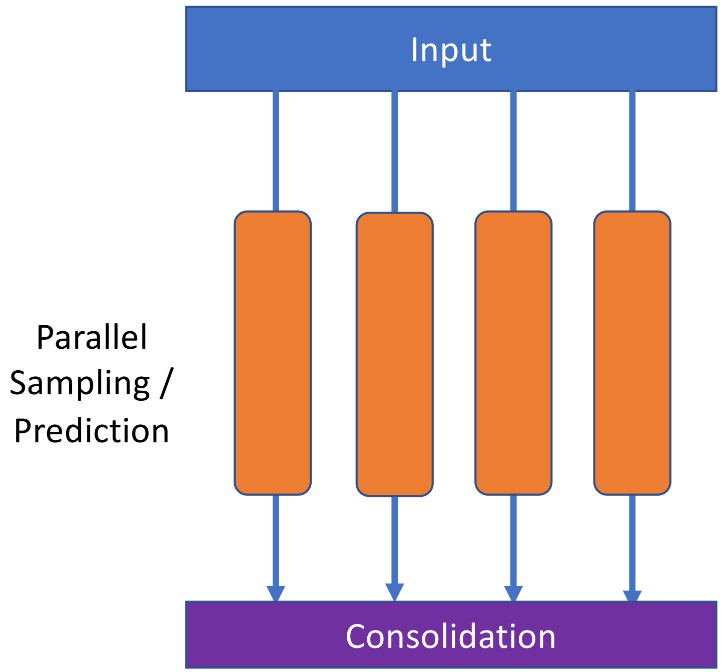
1 cortical sheet
2 million macrocolumns
200 million minicolumns
20 billion neurons

1 macrocolumn
100 minicolumns
10.000 neurons

1 minicolumn
100 neurons

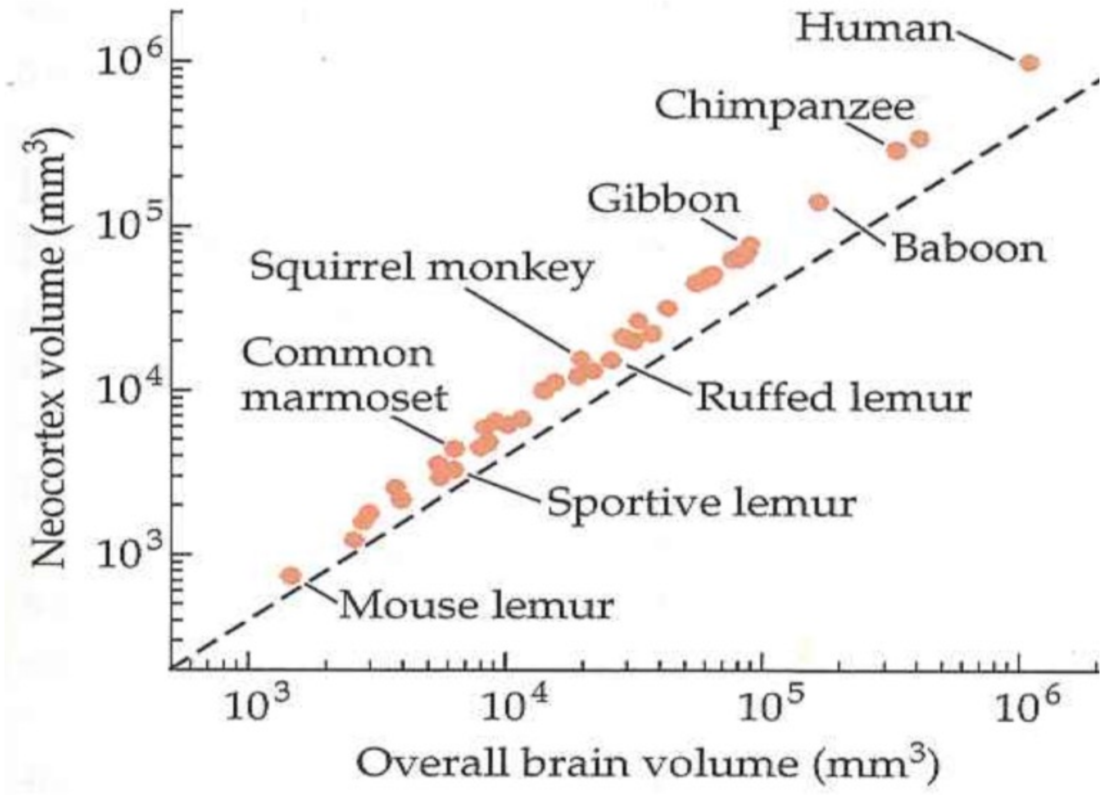
Numenta

What if minicolumns are predicting multiple futures?



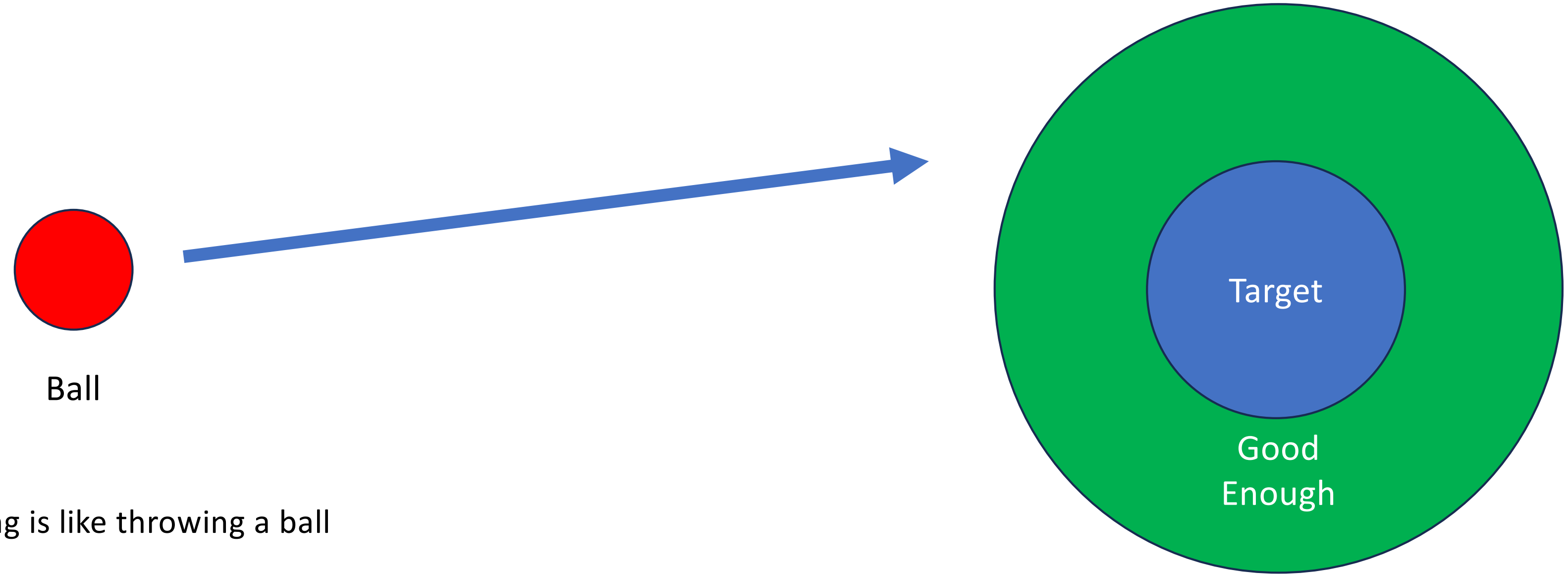
- Perhaps we do parallel prediction in minicolumns
- Perhaps some of them can trigger the fear circuit (ref. the cliff scenario)
- Perhaps human thinking is not meant to be optimal, but use whatever works first
  - The more we sample, the more likely a solution close to optimal will occur
- **Question to ponder: How much performance can we get from increasing parallel search?**

Perhaps intelligence can be gained with multiple sampling?



Toward the quantification of cognition. Richard Granger. 2020.

Aiming a ball

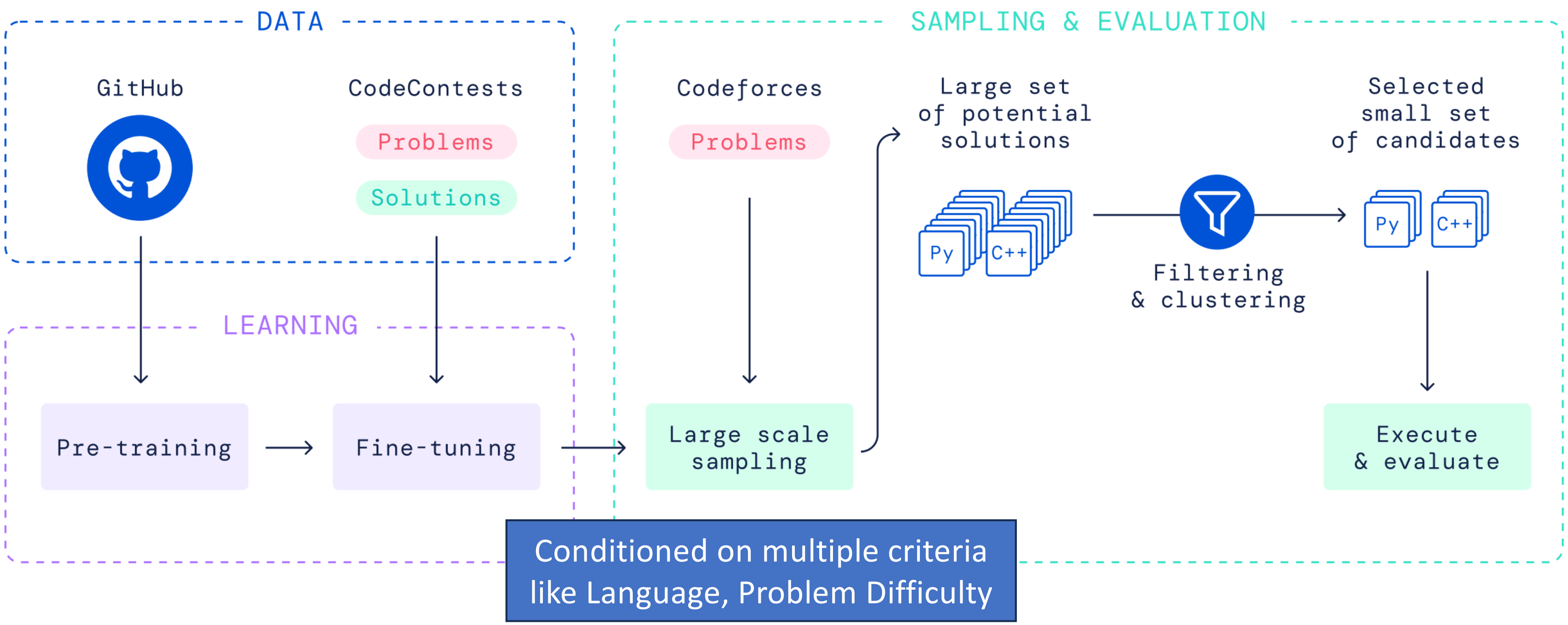


- Sampling is like throwing a ball
- Repeated throws have a higher chance to hit target (vary your starting angle etc.)
- **But what if the target is behind you?**

AlphaCode

Massive sampling and filtering to solve competitive programming questions

AlphaCode



Competition-Level Code Generation with AlphaCode. DeepMind. 2022

Encoder Input X:

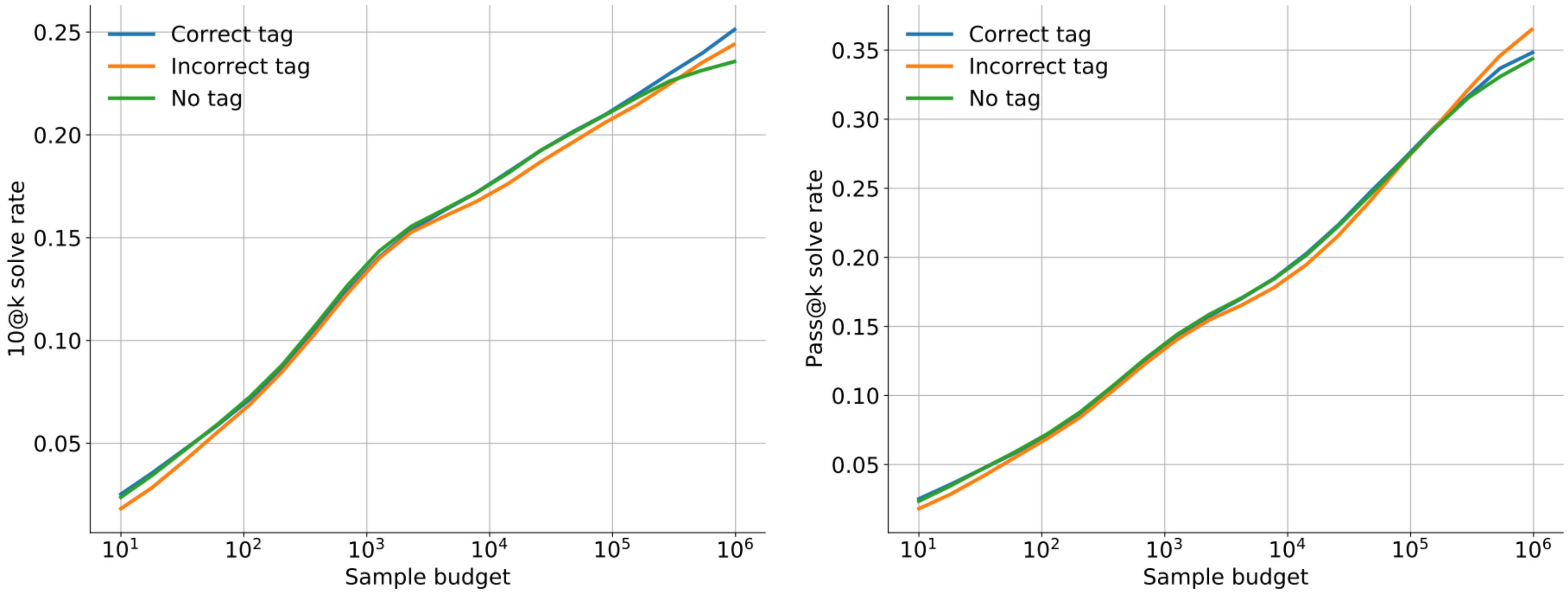
```
// RATING: 1200
// TAGS: math
// LANGUAGE IS cpp
// CORRECT SOLUTION
// n towns are arranged in a circle sequentially. The towns are numbered from 1
// to n in clockwise order. In the i-th town, there lives a singer with a
// repertoire of a_i minutes for each i ∈ [1, n].
```

Decoder Output Y:

```
#include <bits/stdc++.h>
using namespace std;
void solve() {
  long long int n, i, j;
  cin >> n;
  vector<long long int> a(n);
  for (i = 0; i < n; i++) cin >> a[i];
  long long int sum = 0;
  for (i = 0; i < n; i++) sum += a[i];
  if (sum %
    cout << "NO" << '\n';
    return;
  }
```

Competition-Level Code Generation with AlphaCode. DeepMind. 2022

Increased sampling = Increased Solve Rate



(a) 10 attempts per problem (b) Unlimited attempts per problem

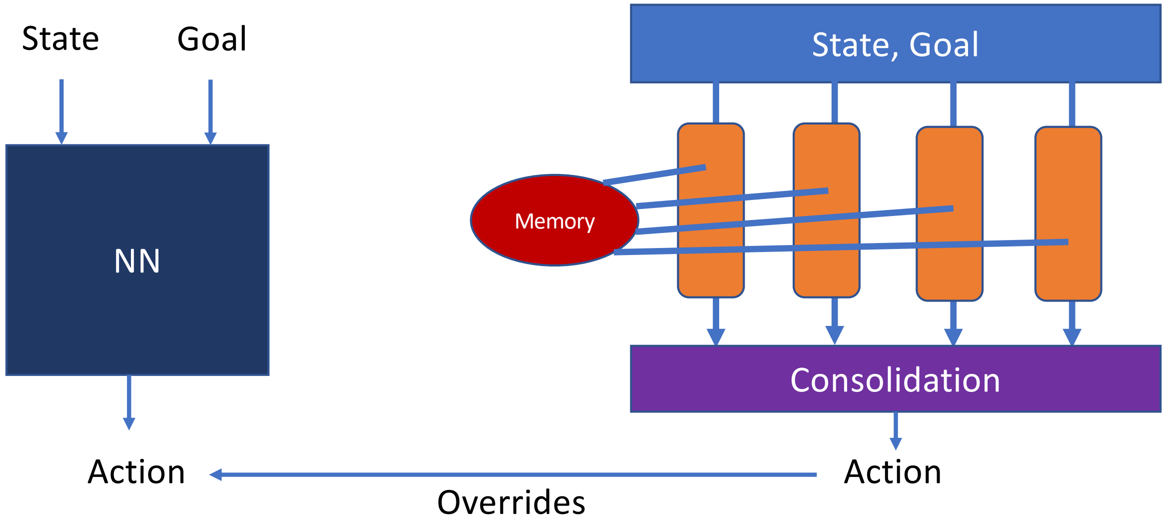
Appendix Figure A13 | **Conditioning on the `CORRECT SOLUTION`, the `INCORRECT SOLUTION`, or no tag.**

Competition-Level Code Generation with AlphaCode. DeepMind. 2022

Learning, Fast and Slow

Massive sampling and filtering to solve a dynamically changing maze

Two Networks – Fast and Slow



Neural Networks: Fast retrieval, slow learning

Memory: Slow retrieval, fast learning

Learning, Fast and Slow. John and Motani. 2023

Dynamic Environment

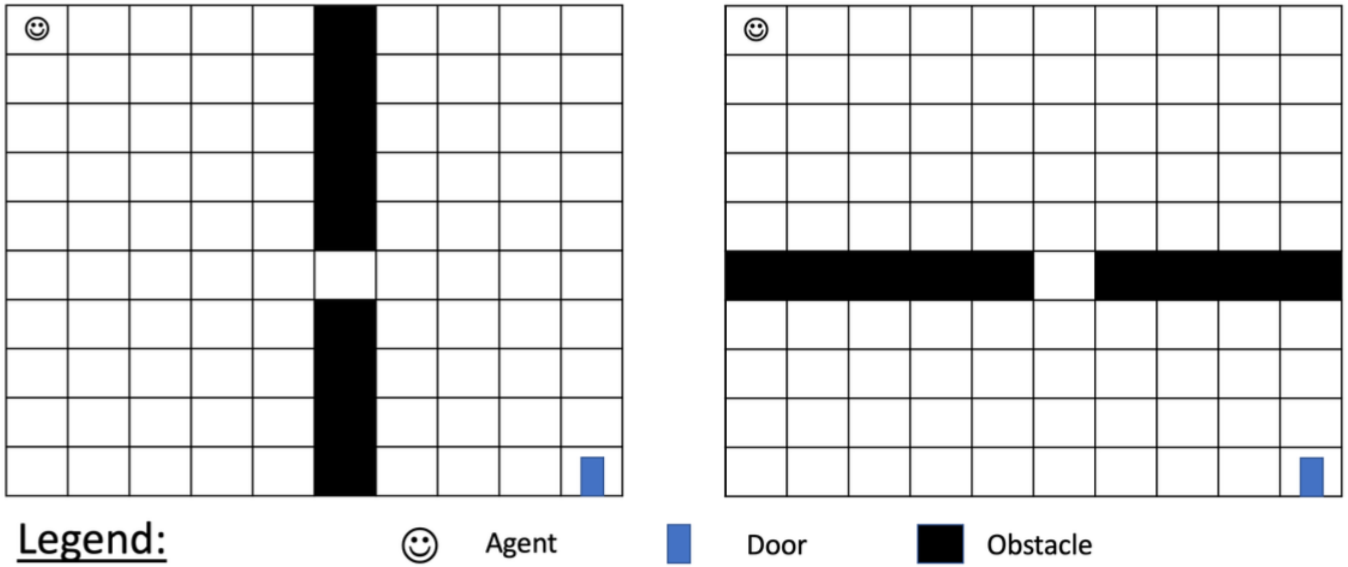
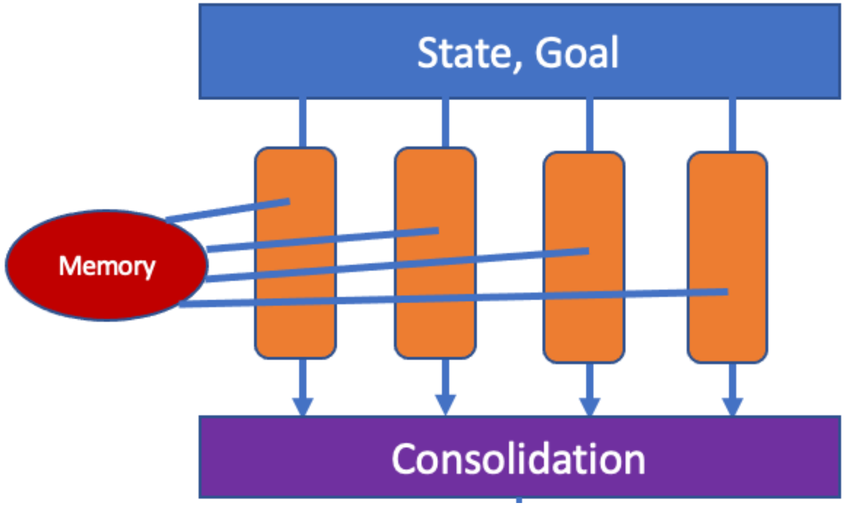


Figure 6: A sample maze environment of size 10x10. By default, the agent's start state is at the top left and the door is at the bottom right, but it can be varied. **(Left)** Obstacles before episode 50 form a vertical wall with a gap in the center across the mid-point. **(Right)** Obstacles after episode 50 from a horizontal wall with a gap in the center across the mid-point.

Increased sampling helps increase efficiency of solution

- Increased depth of sampling, increased breadth (threads) helps find shorter paths



Memory Retrieval Component in "Learning, Fast and Slow"

TABLE IX

ABLATION STUDY ON EFFICIENCY OF F&S AGENT ON A DYNAMIC 10x10 NAVIGATION TASK. LOWER IS BETTER (IN BOLD).

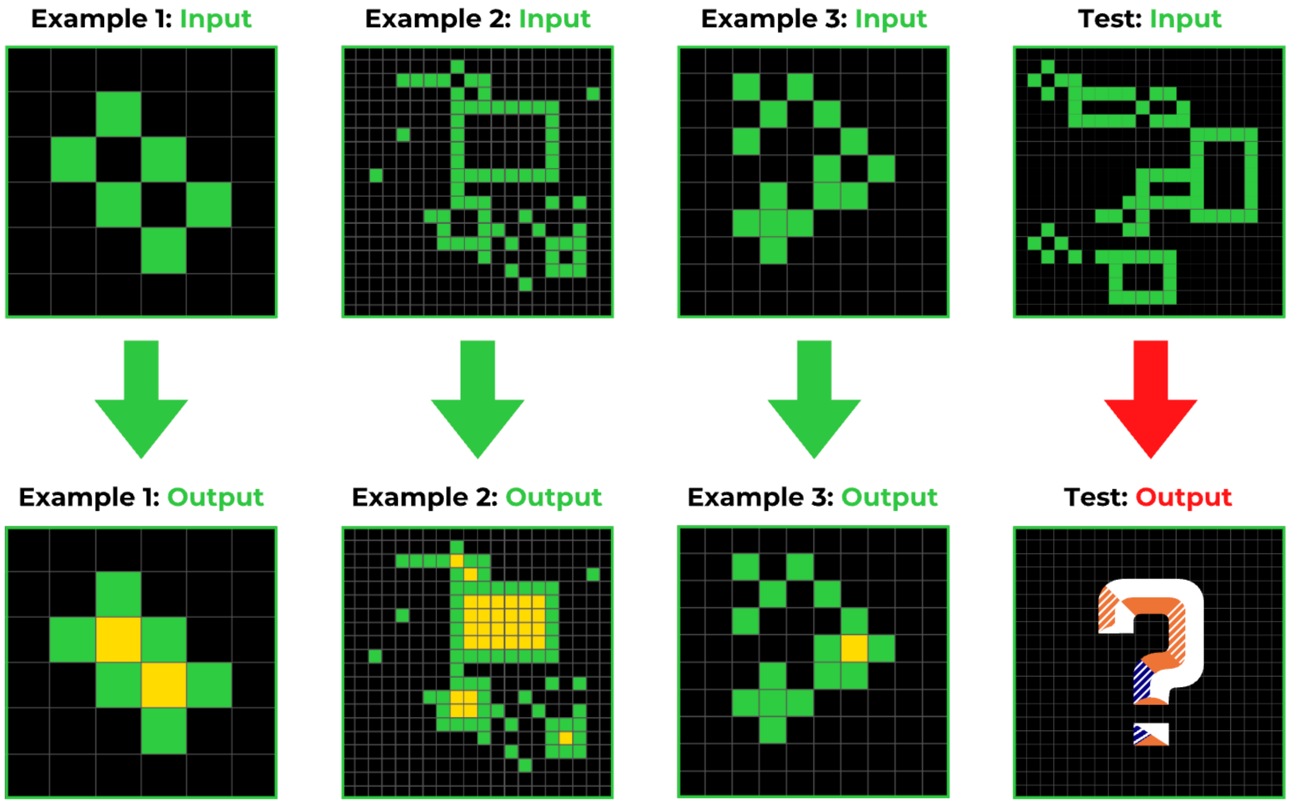
Agent	Steps Above Minimum		
	First 50 eps	Last 50 eps	Total
Baseline	1029.5± 145.4	675.4± 223.3	1704.9± 280.6
No Slow	2625± 234.4	2517.0± 316.0	5142.7± 389.7
No Fast	2694.7± 216.8	2386.6± 445.0	5081.3± 496.2
No Fast,Slow	3890.6± 222.6	3853.0± 207.5	7743.6± 317.3
10 depth	1225.5± 225.1	821.3± 292.3	2046.8± 455.8
50 depth	941.2± 168.0	617.6± 115.4	1558.8± 250.6
50 threads	1112.6± 216.0	761.2± 156.9	1873.8± 231.0
200 threads	**870.2 ± 152.9**	**521.2 ± 132.7**	**1391.4 ± 224.9**

Learning, Fast and Slow. John and Motani. 2023

ARC Challenge

Massive sampling and filtering to solve image-based IQ puzzles

ARC Challenge



https://lab42.global/arc/

Overall Approach by Ryan Greenblatt

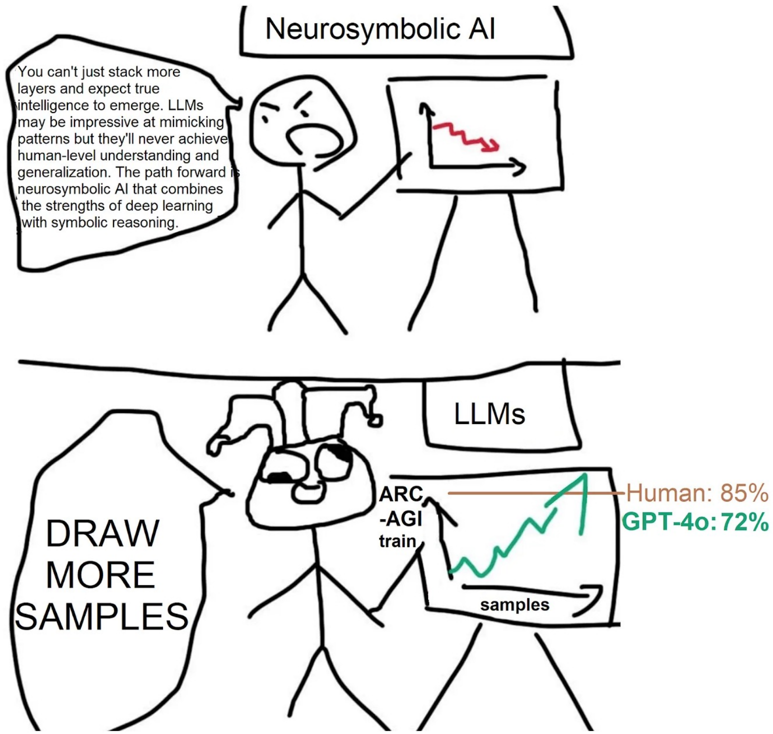
- Show the problem in image form, various text forms for the grid representation
  - The text representations include showing which **cells are occupied by different connected components of colors** and showing **diffs between the input and output** (in cases where the grid shapes are the same)

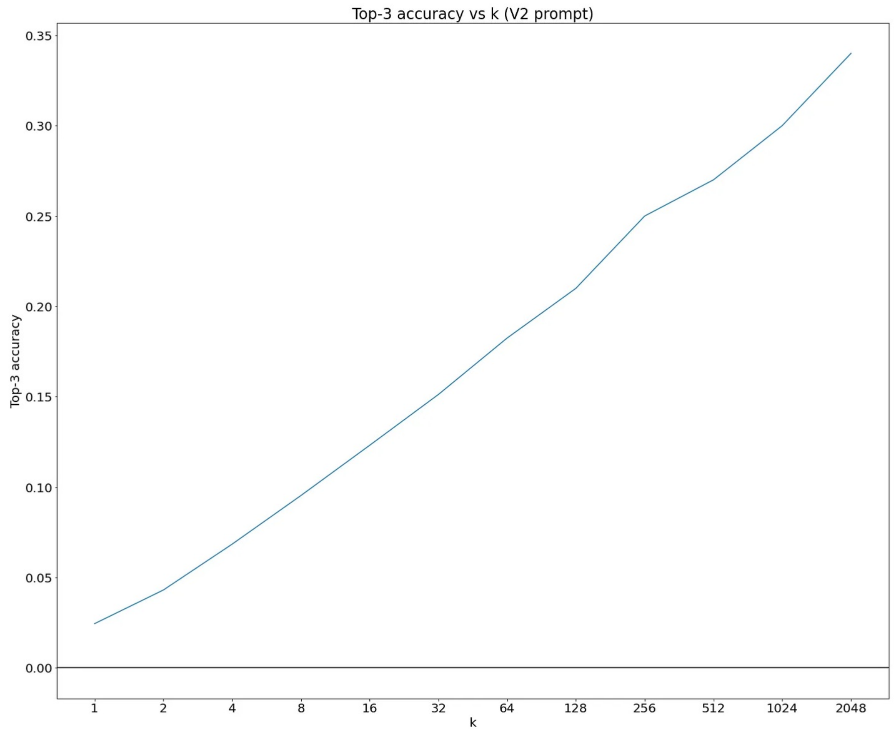
- Sample 8000 programs per problem (wow)

- Take most promising 12 programs (based on score on training examples) and revise outputs according to generated output vs actual training set output

https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt

The benefits of sampling





https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt

The drawbacks of sampling

[–] **Aaron_Scher** 23d 11 0

Nice! Do you have a sense of the total development (and run-time) cost of your solution? "Actually getting to 50% with this main idea took me about 6 days of work." I'm interested in the person-hours and API calls cost of this.

Reply

[–] **ryan_greenblatt** 22d 14 3

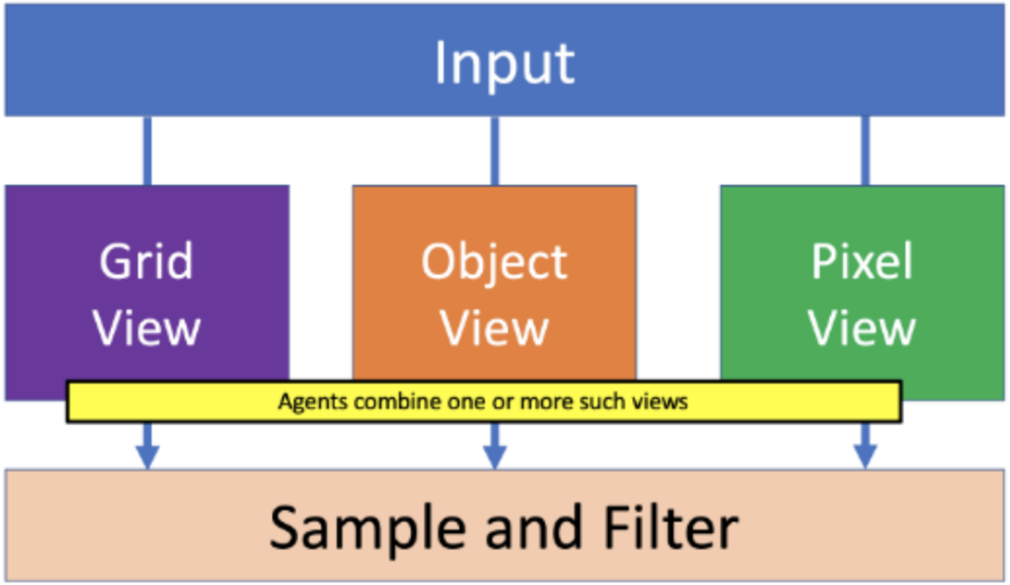
I was pretty inefficient on iteration, so around $40,000. Around 6 person days of development work, though a considerable amount of hours on each day.

Reply 16

https://www.lesswrong.com/posts/Rdwui3wHxCeKb7feK/getting-50-sota-on-arc-agi-with-gpt-4o#XheMa2FNjjjdQGSqJ

My solution – sample with agents with different views

- Each agent views the grid differently, like grid, object, or pixel level
- Generate multiple potential Python programs
- Filter solutions by those which pass training examples
- 20 views * 3 samples = 60 for each problem
- **50 out of 111 public training set tasks solved (45%)**



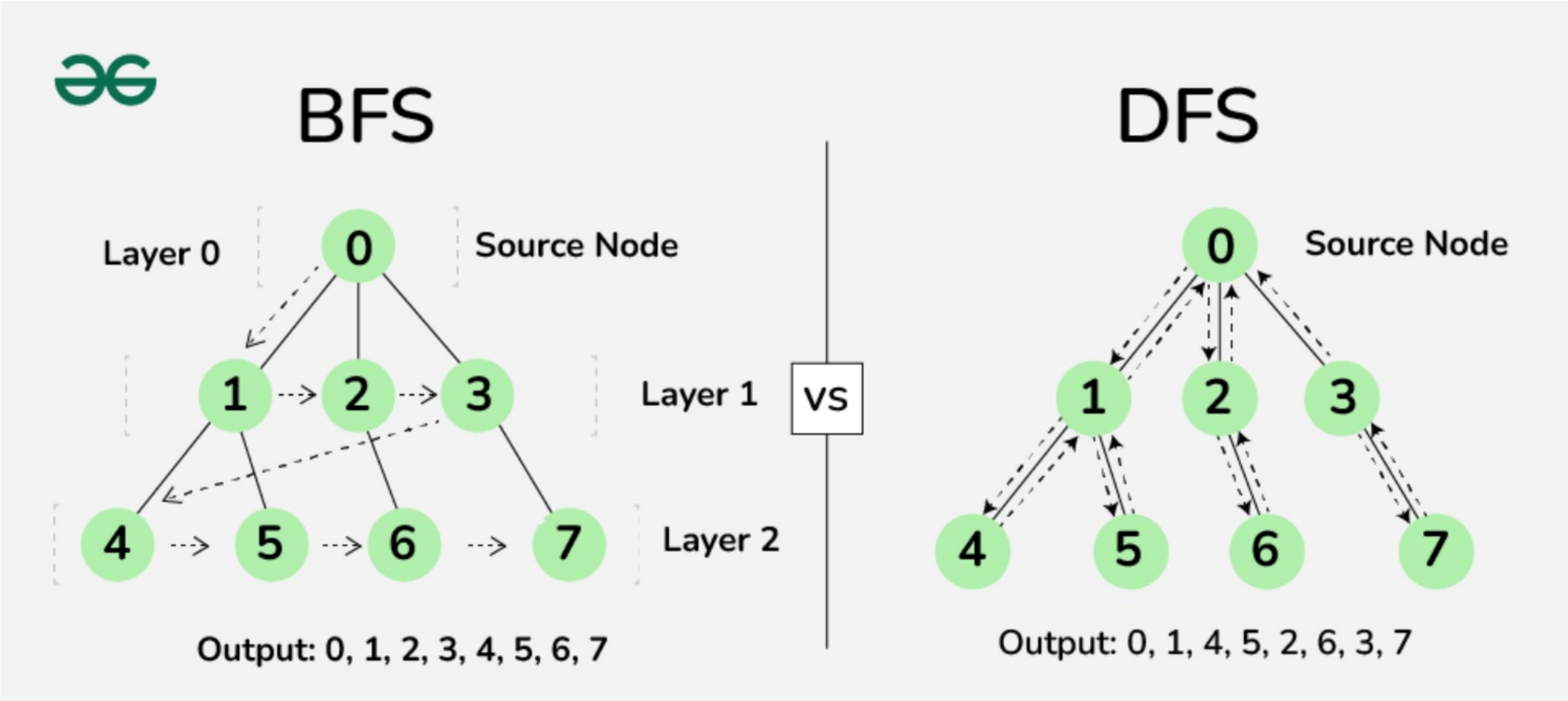
LLMs as a System of Multiple Expert Agents. John and Motani. 2023.

More Efficient Sampling

Sample breadth-wise, sample depth-wise

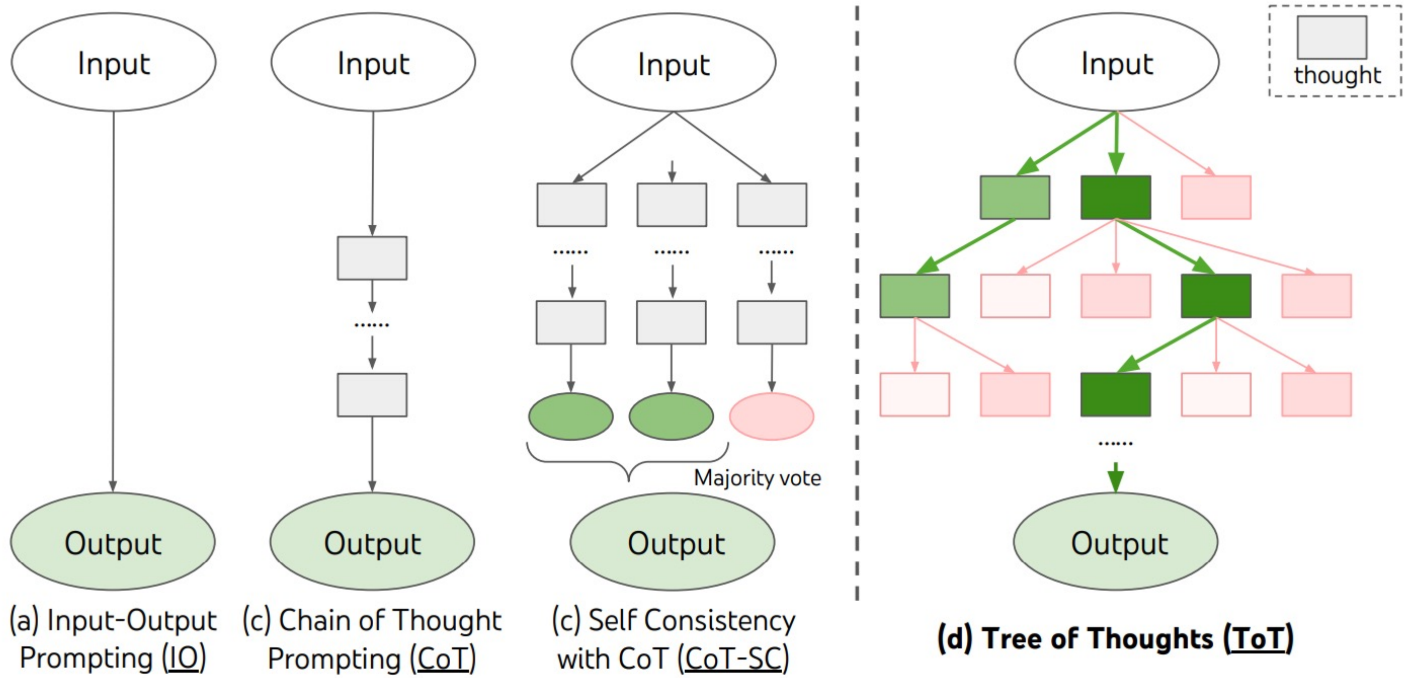
Sample Monte-Carlo Tree Search-wise

Breadth-First Search vs Depth-First Search



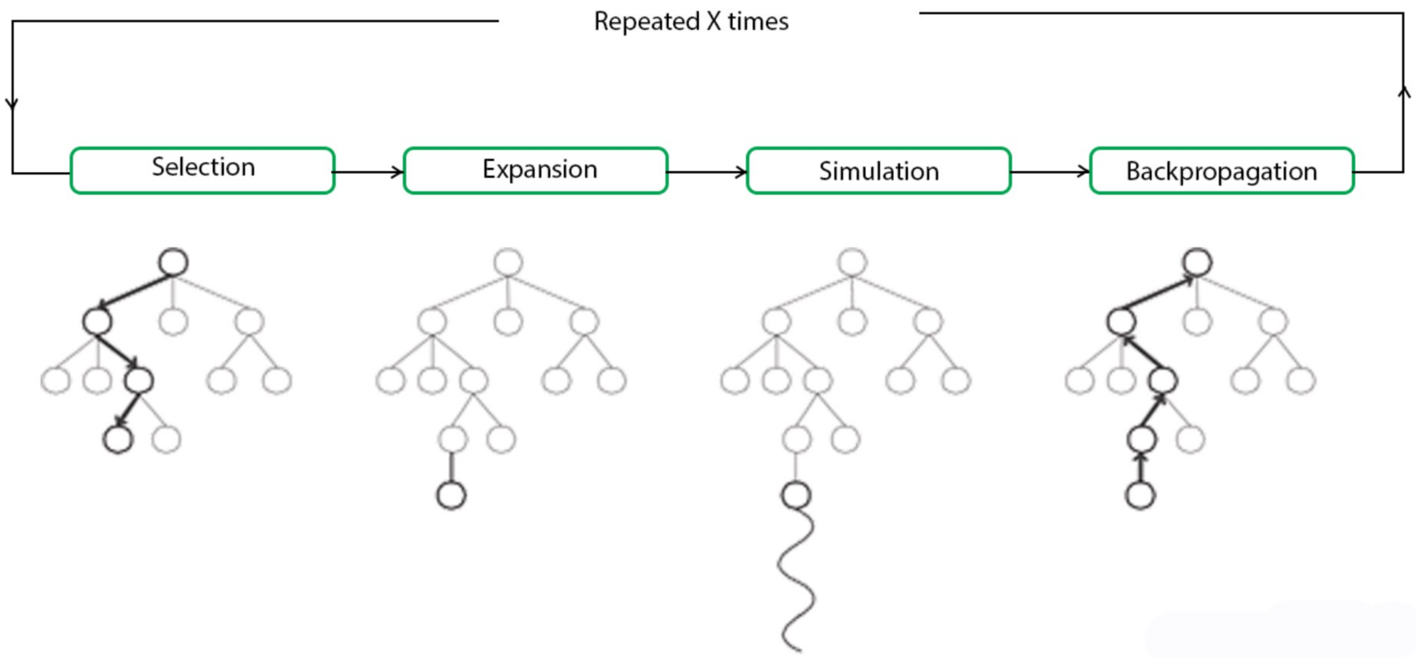
https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

Tree of Thought: Expand out most promising node, prune breadth-wise or depth-wise



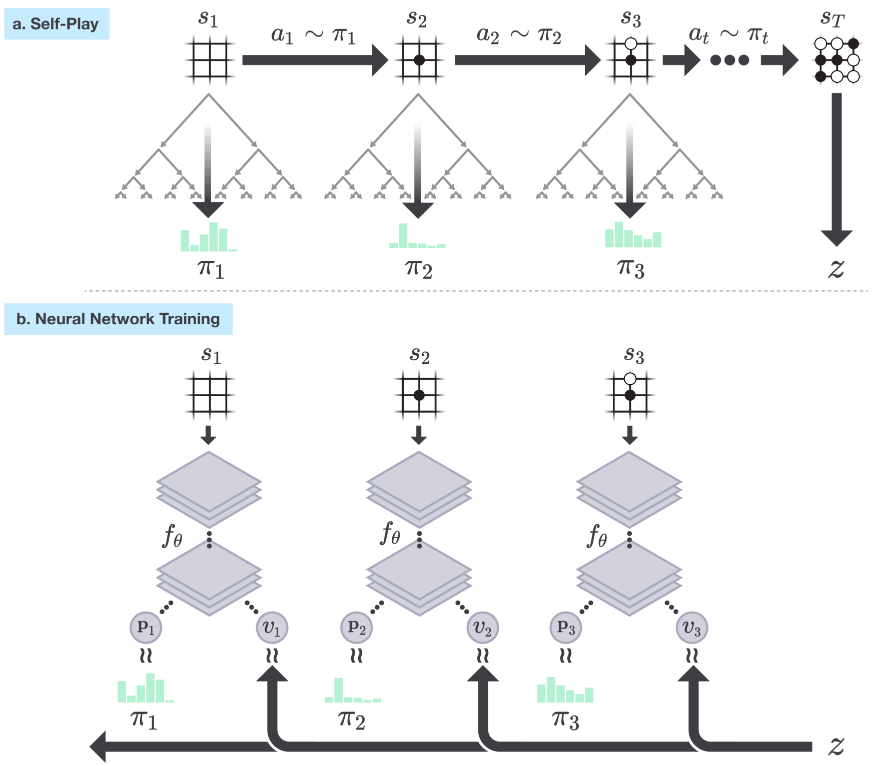
Tree of Thoughts: Deliberate Problem Solving with Large Language Models. Yao et al. 2023.

Monte-Carlo Tree Search: Search with Explore and Exploit



https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

MCTS can be an efficient way to search and improve



AlphaZero: MCTS used to improve policy and value networks

Datasets	Zero-Shot CoT	One-turn Self-refine	4-rollouts MCTSr	8-rollouts MCTSr	Example Nums
GSM8K	977	1147	1227	1275	1319
	74.07%	86.96%	93.03%	96.66%	
GSM-Hard	336	440	526	600	1319
	25.47%	33.36%	39.88%	45.49%	

Table 1: Performance of MCTSr on the GSM Dataset

MCTSr: Using Llama3 8B to evaluate promising states to expand with a refinement step

Accessing GPT-4 level Mathematical Olympiad Solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B: A Technical Report. Zhang et al. 2024.

My thoughts

- More complicated forms of sampling like BFS, DFS, MCTS require keeping track of existing states globally

- Minicolumns in the brain may not have the infrastructure to do that and may just be doing parallel search with different biases

- BFS, DFS, MCTS are traditionally sequential operations and VERY SLOW to implement
  - Potentially able to do asynchronous operations at the cost of some non-optimality (ref. AlphaGo setting node value to huge negative number to prevent exploitation when one branch is being calculated)

Questions to Ponder

- Can sampling help if model does not know how to generate the answer?
- How do we bias the generation effectively?
- Is System 2 thinking simply just more deliberate use of the parallel generation mechanism?
- Are more complex forms of search, like BFS or DFS or MCTS, needed?