ICLR

Scattered Forest Search: Smarter Code Space Search Improves LLM Inference Scaling

Can we adapt better adapt search to language?



Berkeley
UNIVERSITY OF CALIFORNIA
UCLA

NEC

Key Idea

- Search and optimizing over “language space” instead of vector space
- Need to balance exploration and exploitation
- Can leverage optimization ideas from numerical optimization

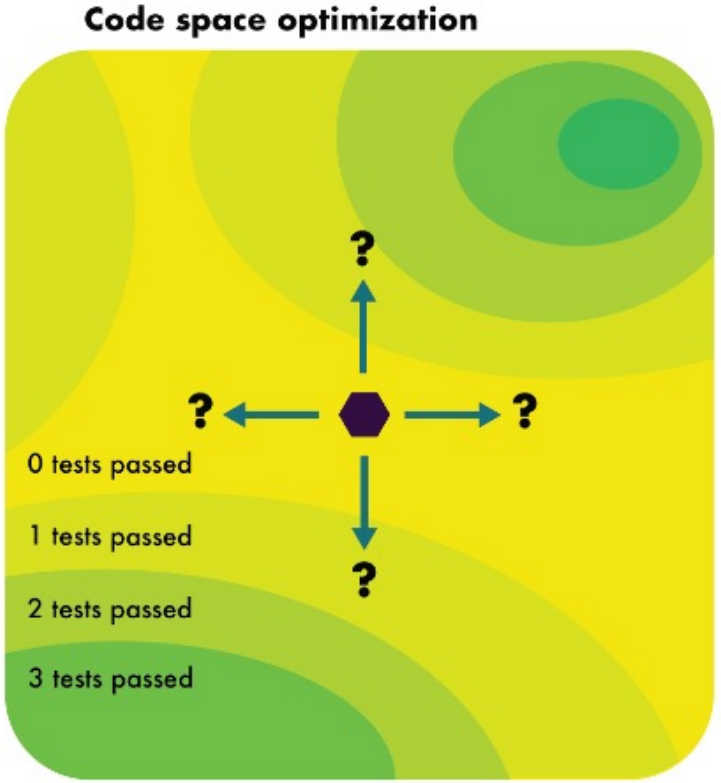


Figure 1: **2D Visualization of Code Space** represents each point as a possible code solution. The goal is to efficiently search this space for the solution with the best performance, defined by the number of unit tests passed, as indicated by the contours above.

Example coding problem

Example code generation prompt, solution, and tests

Prompt: Write a function `greatest_common_divisor(a,b)` that returns the GCD of two integers `a` and `b`

Validation tests:

```
assert(greatest_common_divisor(3,5) == 1)
assert(greatest_common_divisor(25,15) == 5)
assert(greatest_common_divisor(0,3) == 3)
```

Proposed solution 1:

```
def greatest_common_divisor(a, b):
    for i in range(min(a, b), 0, -1):
        if a % i == 0 and b % i == 0:
            return i
```

Test feedback:

```
assert(greatest_common_divisor(3,5) == 1) #output 1 is
correct
assert(greatest_common_divisor(25,15) == 5) #output 5 is
correct
assert(greatest_common_divisor(0,3) == 3) #output None is
incorrect
```

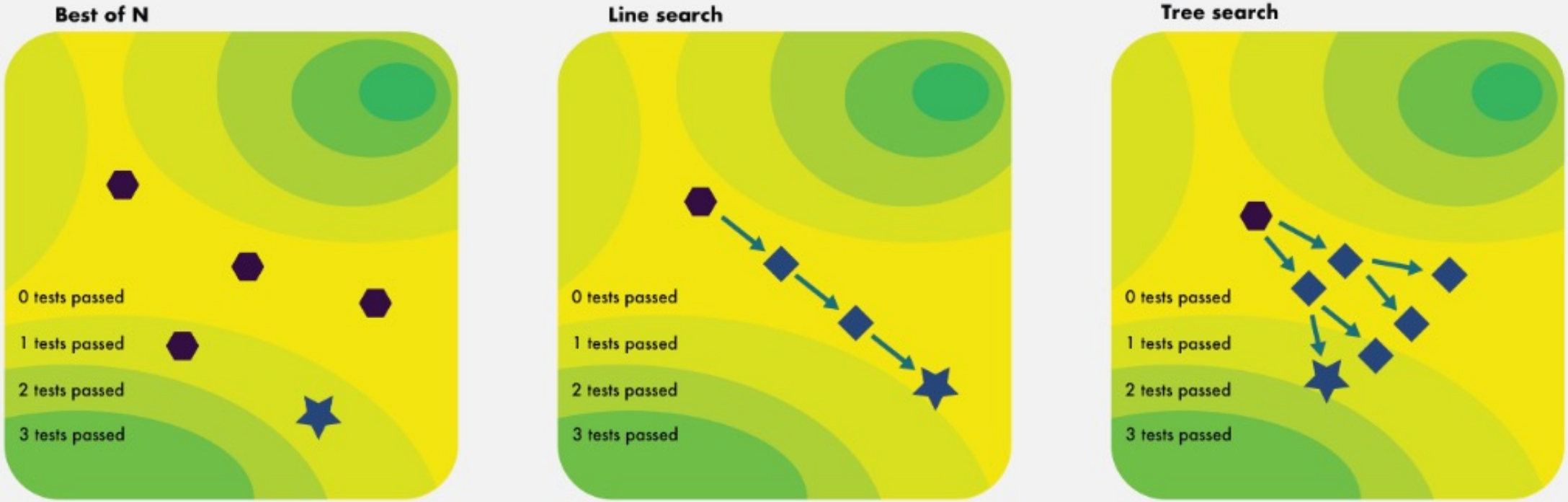
Common prior LLM search methods

Search method

- Sequential revision
- Best of N
- Tree search
- Evolutionary search
- Beam search

We can reinterpret LLM search as optimizing over language space!

Figure 3: **Overview of prior methods** used for code generation with LLMs. Points represent solutions. Hexagons represent initial solutions. Star represents the final selected solution.

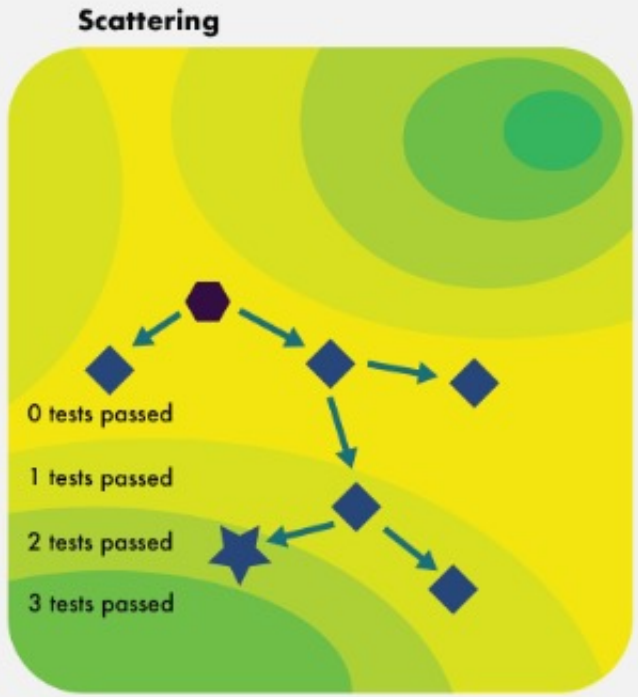


Repeated sampling generates multiple solutions using the LLM without leveraging feedback from previous iterations.

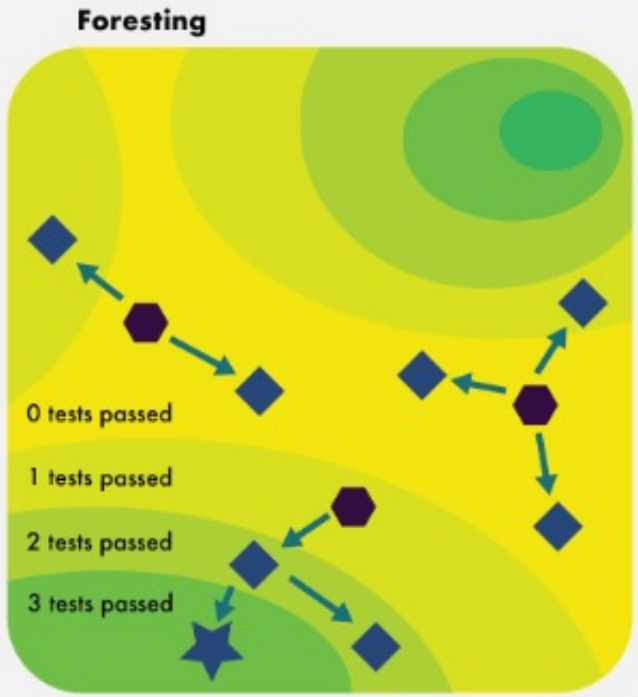
Line search rigidly exploits feedback and cannot revert to a previous solution if a new change worsens the outcome

Tree search is more flexible but still lacks sufficient exploration, as the generated solutions tend to be very similar.

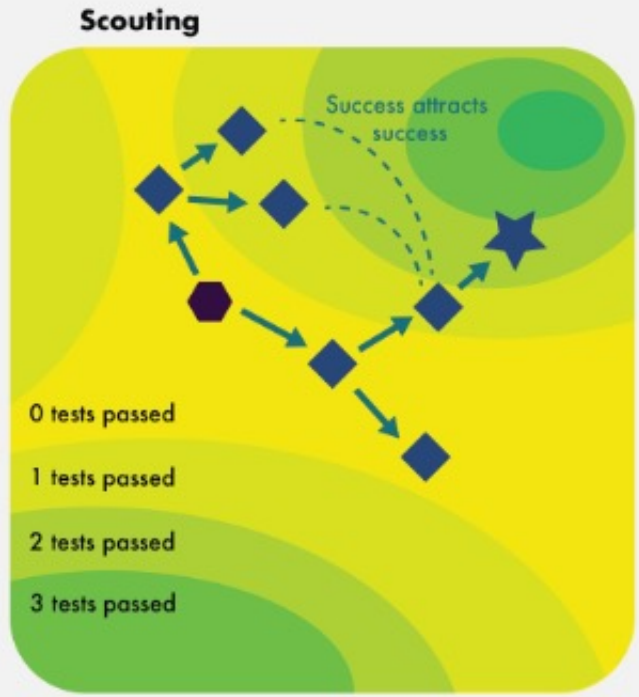
Figure 7: **Core techniques** used by SFS. Points represent solutions. Hexagons represent initial solutions. Star represents the final selected solution.



SCATTERING encourages tree search to **explore** more diverse solutions by using varied directional prompts for each branch or seed solution



FORESTING boosts **exploration** by performing tree search dynamically from multiple random seed solution starting points



SCOUTING shares successful search directions across branches of the search tree, providing general insights to better **exploit** feedback

Example of textual “directions”

feedback from validation tests → propose new textual directions using LLM → choose 1 direction to implement when branching

Example SCATTERING Directions

Thoughts: The feedback suggests that the main problem is that the function is returning the first element of `min-k` and `max-k` instead of the entire lists.

Direction 1: Modify the return statement to return `min-k` and `max-k` instead of (`min-k[0]`, `max-k[0]`, `min-k[-1]`, `max-k[-1]`). This will ensure that the function returns the entire lists of minimum and maximum k elements.

Direction 2: Update the function to handle the case when K is greater than the length of the tuple. In this case, return the entire sorted tuple as both the minimum and maximum k elements.

Improves diversity of search directions

Multi-start initialization (foresting) seed prompts

Example Forest Seed Instructions

Seed instruction 1: Write the code in a modular and extensible manner, ensuring each function or class has a single responsibility and can be easily extended or modified without impacting other parts of the system. Prioritize clear interfaces and loose coupling between components.

Seed instruction 2: Focus on writing highly efficient code with minimal memory usage and fast execution times. Use data structures and algorithms optimized for performance, and consider edge cases that could lead to bottlenecks. Prioritize speed and resource efficiency over readability.

Seed instruction 3: Prioritize readability and maintainability in your code. Write clear and descriptive comments, use meaningful variable and function names, and structure the code in a way that is easy for others to understand and modify. Follow established coding standards and best practices.

Starts the search algorithm out from different initial starting points

Scouting: Example global insights

Example SCOUTING Insights

Insight 1: Modify the return statement to return `min-k` and `max-k` instead of (`min-k[0]`, `max-k[0]`, `min-k[-1]`, `max-k[-1]`). This will ensure that the function returns the entire lists of minimum and maximum k elements.

Insight 2: Update the function to handle the case when K is greater than the length of the tuple. In this case, return the entire sorted tuple as both the minimum and maximum k elements.

use insights to generate directions → see if the direction worked or not → update insights based on feedback

- Global insights developed during search process
- Like ACO/PSO, provides more guidance towards correct search direction

Measure exploration vs exploitation for LLM search

Table 6: **Metrics for different search methods.** We run search methods for 10 iters. each using `gpt-3.5-turbo` on HumanEval. Our method generates more diverse solutions and discovers the correct solution faster. We also compare against a genetic algorithm (Romera-Paredes et al., 2024)

Method	pass@1	pass@any	BERT sim.	val. score	iters. (incl)	iters. (excl)
Line	68.1%	83.1%	0.9992	0.795	2.09	7.13
Tree (MCTS)	75.6%	76.9%	0.9998	0.827	2.38	8.09
Best of N	73.8%	75.6%	0.9983	0.774	2.59	9.00
Genetic	74.4%	75.6%	0.9994	0.815	3.04	10.36
Ours (SFS)	82.5%	89.0%	0.9945	0.813	1.67	5.06

- Similarity scores help measure diversity and exploration
- Average validation scores help measure exploitation

Strong empirical performance

- Both Pass@any and Pass@1 scores indicate strong performance given a fixed budget

Table 2: **Performance of our method compared to prior search methods.** Pass@1 performance reported here. Both solutions and validation tests were generated using `gpt-3.5-turbo`.

Method / Benchmark	HumanEval+	MBPP+	Leetcode	APPS	CodeContests
Base	58.5%	64.9%	30.0%	16.0%	1.82%
Line	53.0%	61.2%	28.9%	14.5%	1.21%
Tree (MCTS)	59.8%	65.4%	31.7%	18.0%	2.42%
Best of N	65.2%	64.4%	33.3%	19.5%	1.82%
Ours (SFS)	**67.1%**	**65.7%**	**36.7%**	**20.5%**	**4.24%**

Table 3: **Pass@any accuracy of our method compared to prior search methods.** Both solutions and validation tests were generated using `gpt-3.5-turbo`. We run each method for 10 iterations.

Method / Benchmark	HumanEval	MBPP	Leetcode	APPS	CodeContests
Line	83.1%	82.9%	33.3%	22.0%	2.99%
Tree (MCTS)	76.9%	79.6%	33.9%	21.5%	2.42%
Best of N	75.6%	77.3%	33.9%	23.0%	3.03%
Ours (SFS)	**89.0%**	**86.1%**	**39.4%**	**32.5%**	**6.06%**

Strong inference scalability

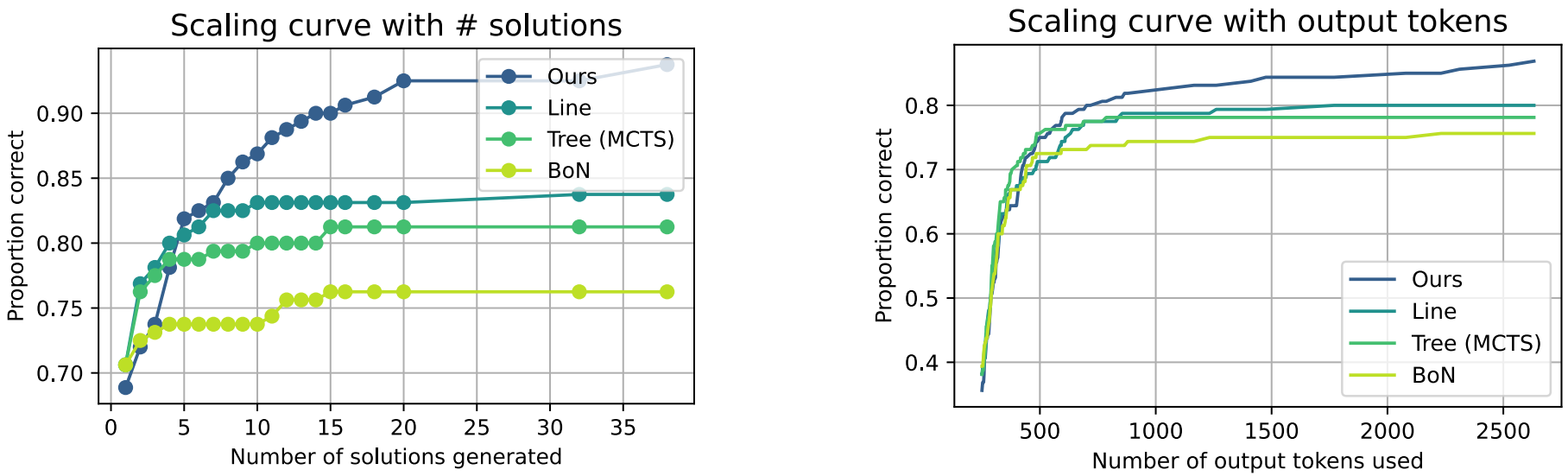


Figure 12: **Scaling curves for different search methods.** *Left:* Proportion of problems solved vs. number of solutions generated. *Right:* Proportion of problems solved vs. number of tokens used. Results are from `gpt-3.5-turbo` on HumanEval. Our method shows consistent improvement up to 20 solutions. Other methods plateau and do not catch up to SFS even with additional scaling. Additional curves can be found in Sec. D, including curves for other datasets.

Key advantages

Simple, effective, easy to use.

Our method requires

- no prompt tuning
- no additional training
- no complicated workflows

Great scaling at any budget!

You can feed it nonsense, and it still works!

"Jabberwocky" prompts are completely unrelated to code generation yet inserting them to improve solution diversity still improves performance!

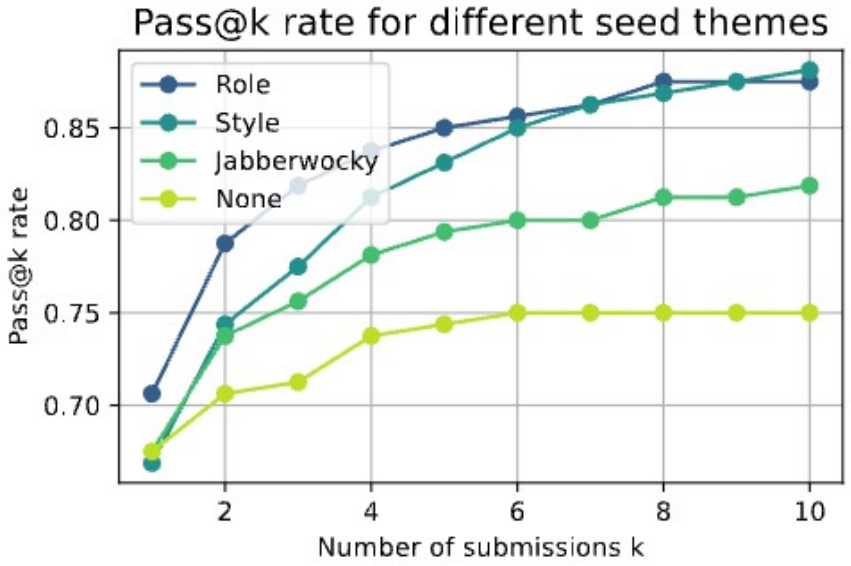


Figure 11: **Pass@k rate** for repeated sampling with different initialization seed types on HumanEval using `gpt-3.5-turbo-0613`. Increasing seed variety with SCATTERING significantly improves both Pass@k rate and scaling.

"Beware the Jabberwock, my Son."