A Appendix

A.1 Examples of Algorithm Synthesis and Functionality Synthesis

We give examples of algorithm synthesis and functionality synthesis in Figure 7.

The upper part is an instance of algorithm synthesis, where input that describes the problem does not indicate the solution idea 'binary search' in any way. To solve the algorithm synthesis program, the model needs to come up with the idea related to 'binary search' first either implicitly or explicitly before synthesizing the code.

The lower part is an instance of functionality synthesis, where the input is basically a pseudo-code program described in natural language that can directly translate into Python code. The model does not need to come up with the idea because the solution is already clearly stated in the input.

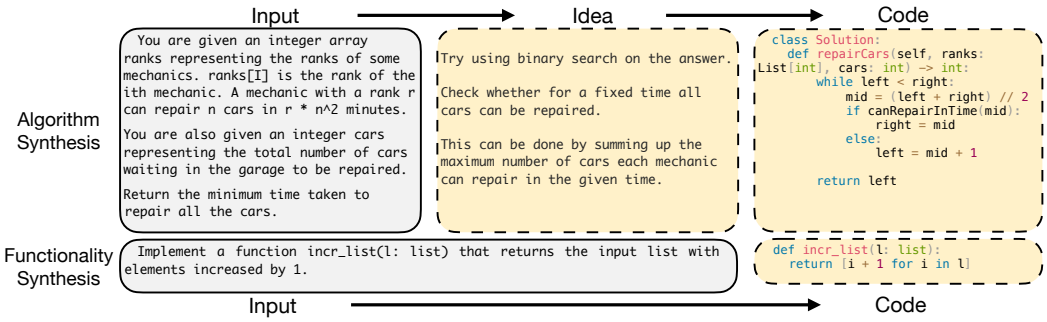

Figure 7: Examples of algorithm synthesis (top) and functionality synthesis (bottom). The parts in solid boxes are given by the problem, while the parts in dotted boxes should be inferred by the model. Functionality synthesis does not require the model to infer the idea, while algorithm synthesis does.

A.2 Prompts in ALGO and Examples of Generated Programs

A.2.1 LeetCode Examples

We list the prompts we used to generate the solution candidates and the verifier components for LeetCode problems here.

Figure 8 lists the prompts and examples for generating the naive solution. Figure 9 lists the prompts and examples for generating solutions guided by instructions that specify the algorithm category. Figure 10 lists the prompts and examples for generating reference oracle. Figure 11 lists an input validator example and its prompt. Figure 12 lists the prompt for the input generator that generates a single test input. Figure 13 lists the prompt for the batch input generator.

Naive Candidate Generation Prompt

```
Solve the following problem with python and test it against the example cases only.

You are given an integer array ranks representing the ranks of some mechanics.
ranks[I] is the rank of the ith mechanic. A mechanic with a rank r can repair n
cars in r * n^2 minutes. Mechanics work in parallel.
You are also given an integer cars representing the total number of cars waiting
in the garage to be repaired.
Return the minimum time taken to repair all the cars.
Examples: (omitted)
Constraints: (omitted)

**Function definition**
```
class Solution:
    def repairCars(self, ranks: List[int], cars: int) -> int:
```
```

Naive Candidate Generated by LLM

```
class Solution:
  def repairCars(self, ranks: List[int], cars: int) -> int:
    ranks.sort()
    max_repair_time = 0
    assigned_cars = 0
    for rank in ranks:
      num_cars = (cars + len(ranks) - assigned_cars - 1)\
        // (len(ranks) - assigned_cars)
      num_cars = min(num_cars, cars - assigned_cars)
      repair_time = rank * num_cars * num_cars
      max_repair_time = max(max_repair_time, repair_time)
      assigned_cars += num_cars
    return max_repair_time

      num_cars = min(num_cars, cars - assigned_cars)
      repair_time = rank * num_cars * num_cars
      max_repair_time = max(max_repair_time, repair_time)
      assigned_cars += num_cars
    return max_repair_time
```

Figure 8: The naive prompt without instruction guidance and a candidate program generated with it. The instructions are in blue. It is a greedy method in linear time but is not suitable for this task. Even the naive candidate generated can pass the examples in the description, it fails both the system judge and ALGO verifier.

Instruction Guided Generation Prompt

```
Please solve this problem with the binary search method.
After generating the code, please test it against the example cases given in the
definition. Only use the example cases, don't use your own test cases.

You are given an integer array ranks representing the ranks of some mechanics.
ranks[I] is the rank of the ith mechanic. A mechanic with a rank r can repair n
cars in r * n^2 minutes. Mechanics work in parallel.
You are also given an integer cars representing the total number of cars waiting
in the garage to be repaired.
Return the minimum time taken to repair all the cars.
Examples: (omitted)
Constraints: (omitted)
**Function definition**
```

```
class Solution:
    def repairCars(self, ranks: List[int], cars: int) -> int:

```

```
As I said before, please make sure you use the binary search method.
```

Binary Search Candidate Generated

```python
class Solution:
  def repairCars(self, ranks: List[int], cars: int) -> int:
    def countCarsRepaired(ranks, minutes):
      total_cars = 0
      for rank in ranks:
        n = int((minutes / rank) ** 0.5)
        total_cars += n
      return total_cars
    left = 1
    right = max(ranks) * cars * cars
    while left < right:
      mid = (left + right) // 2
      if countCarsRepaired(ranks, mid) < cars:
        left = mid + 1
      else:
        right = mid
    return left
```

Figure 9: The prompt with instructions about the specific algorithm category (binary search). Binary search is the ideal algorithm used by human programmers for this task, with which the coder can generate both correct and efficient solutions. In general, right algorithm category can efficiently improve the quality of generated program.

Reference Oracle Generation Prompt

```
Please solve this problem with a brute-force algorithm. After generating the code,
please test it against the example cases given only, do not generate your own test
cases.When comparing outputs, please do not consider the line breaks and spaces.

You are given an integer array ranks representing the ranks of some mechanics. ranks[I]
is the rank of the ith mechanic. A mechanic with a rank r can repair n cars in r * n^2
minutes.

You are also given an integer cars representing the total number of cars waiting in the
garage to be repaired.

Return the minimum time taken to repair all the cars.

### Examples

{examples, omitted}

Instead of defining your function in the Solution class, change your classname to
BruteforceSolution. As I said before, please make sure you use the most straightforward
and brute-force algorithm to solve the algorithm. Do not consider any efficiency issue,
make the solution as brute-force as possible. It's okay for you to enumerate over a very
large search space as long as the solution is correct. What do you think are some
variables that may affect the answer and how do you think they can be enumerated?
```

Reference Oracle Generated by LLM

```python
class BruteforceSolution:
  def repairCars(self, ranks: List[int], cars: int) -> int:
    num_mechanics = len(ranks)
    min_time = float('inf')
    def generate_allocations(allocation, remaining_cars):
      nonlocal min_time
      if len(allocation) == num_mechanics:
        if sum(allocation) == cars:
          times = [ranks[i] * allocation[i] * allocation[i]
for i in range(num_mechanics)]
          max_time = max(times)
          min_time = min(min_time, max_time)
        return
      for num_cars in range(remaining_cars + 1):
        new_allocation = allocation + [num_cars]
        new_remaining_cars = remaining_cars - num_cars
        generate_allocations(new_allocation,
new_remaining_cars)
    generate_allocations([], cars)
    return min_time
```

Figure 10: The prompt we used for oracle generation and one oracle generated with it. The instructions are in blue. The language model is instructed to generate the most straightforward solution by enumerating over a very large search space of all combinations of relevant variables. The generated oracle enumerates all the possible ordered partitions of work allocations to find out the optimal one.

Input Validator Generation Prompt

```
You are given this leetcode problem. Please help me by generating a validator function
`is_valid_input` that takes exactly the same inputs as the solution function and returns
a boolean value indicating whether the input is valid and follows the constraints
defined in the problem description. Please test your data validator by checking the
validity of the example cases given in the problem description.

You are given an integer array ranks representing the ranks of some mechanics. ranks[I]
is the rank of the ith mechanic. A mechanic with a rank r can repair n cars in r * n^2
minutes. Mechanics work in parallel.
You are also given an integer cars representing the total number of cars waiting in the
garage to be repaired.
Return the minimum time taken to repair all the cars.
Examples: (omitted)
Constraints: (omitted)
**Function definition** (omitted)
```

Input Validator Generated by LLM

```python
def is_valid_input(ranks: List[int], cars: int) -> bool:
  if not 1 <= len(ranks) <= 10 ** 5:
    return False
  for rank in ranks:
    if not 1 <= rank <= 100:
      return False
  if not 1 <= cars <= 10 ** 6:
    return False
  return True
```

Figure 11: The prompt is utilized to generate a input validator to verify the validity of the generated test input, ensuring it aligns with the constraints specified in the description. In practice, this validation task is a functionality synthesis task, which can be easily solved by LLM.

A.2.2 Codecontests Examples

We list the prompts we used to generate the solution candidates and the verifier components for Codecontests problems here. Figure 14 lists the prompts and examples for generating the reference oracle using an exhaustive search. Figure 15 lists the prompt for generating the batch input generator.

Input Generator Generation Prompt

```
You are given this leetcode problem and its test input validator `is_valid_input`.
Please help me by generating an input generator function `gen_input` that randomly
generates test inputs which follow the constraints defined. Your `gen_input` function
should make use of `is_valid_input` by validating the generated input with it. Your
function should only generate test inputs and not test outputs.

You are given an integer array ranks representing the ranks of some mechanics. ranks[I]
is the rank of the ith mechanic. A mechanic with a rank r can repair n cars in r * n^2
minutes. Mechanics work in parallel.
You are also given an integer cars representing the total number of cars waiting in the
garage to be repaired.
Return the minimum time taken to repair all the cars.
Examples: (omitted)
Constraints: (omitted)
**Function definition** (omitted)

def is_valid_input(ranks: List[int], cars: int) -> bool: (omitted)
```

Input Generator Generated by LLM

```python
def is_valid_input(ranks: List[int], cars: int) -> bool:
  if not 1 <= len(ranks) <= 10 ** 5:
    return False
  for rank in ranks:
    if not 1 <= rank <= 100:
      return False
  if not 1 <= cars <= 10 ** 6:
    return False
  return True

def gen_input() -> tuple:
  while True:
    n = random.randint(1, 10 ** 5)
    ranks = [random.randint(1, 100) for _ in range(n)]
    cars = random.randint(1, 10 ** 6)
    if is_valid_input(ranks, cars):
      return (ranks, cars)
```

Figure 12: The prompt is used to generate a data generator, to generate extra test cases when combined with the reference oracle. In practice, this generator generation task is a functionality synthesis task, which can be easily solved by LLM.

Batch Generator Generation Prompt

```
You are given this leetcode problem, its random input generator `gen_input`. Please help
me by generating 5 random inputs with the random input generator `gen_input`. You should
return a function named `random_test()` that takes several arguments to control the
upper limit of the input and returns a list of 5 tuples. You do not need to generate
test outputs, only generate inputs.

You are given an integer array ranks representing the ranks of some mechanics. ranks[I]
is the rank of the ith mechanic. A mechanic with a rank r can repair n cars in r * n^2
minutes. Mechanics work in parallel.
You are also given an integer cars representing the total number of cars waiting in the
garage to be repaired.
Return the minimum time taken to repair all the cars.
Examples: (omitted)
Constraints: (omitted)

def gen_input() -> tuple: (omitted)
```

Batch Generator Generated by LLM

```python
def random_test(max_ranks_length: int=10 ** 5,
max_rank_value: int=100, max_cars: int=10 ** 6) ->
List[tuple]:

  def gen_input() -> tuple:
    while True:
      n = random.randint(1, min(max_ranks_length, 10 ** 5))
      ranks = [random.randint(1, min(max_rank_value, 100))
for _ in range(n)]
      cars = random.randint(1, min(max_cars, 10 ** 6))
      if is_valid_input(ranks, cars):
        return (ranks, cars)
  random_inputs = []
  for _ in range(5):
    random_inputs.append(gen_input())
  return random_inputs
```

Figure 13: The prompt is used to generate a batch generator that employs the already generated `gen_input` to generate several input cases by calling it multiple times.

Since the test set problems in Codecontests are all from `https://codeforces.com/`, they originally require standard input/output. However, we follow the setting in Codecontests by concatenating the entire standard input as a single string and asking the model to generate reference oracles that map a string to a string.

Reference Oracle Generation Prompt

```
Please solve this problem with a brute-force algorithm. After generating the code,
please test it against the example cases given only, do not generate your own test
cases.When comparing outputs, please do not consider the line breaks and spaces.

Andi and Budi were given an assignment to tidy up their bookshelf of n books.
Each book is represented by the book title – a string si numbered from 1 to n,
each with length m. Andi really wants to sort the book lexicographically
ascending, while Budi wants to sort it lexicographically descending.
(https://codeforces.com/problemset/problem/1575/A, remaining part omitted)

### Examples

{examples, omitted}

Please feed the example inputs to the function solution as an entire string. As I said
before, please make sure you use the most straightforward and brute-force algorithm to
solve the algorithm. Do not consider any efficiency issue, make the solution as brute-
force as possible. What do you think are some variables that may affect the answer and
how do you think they can be enumerated?
```

Reference Oracle Generated by LLM

```python
from functools import cmp_to_key
def asc_desc_ending_comparator(a, b):
  for i in range(len(a)):
    if (i + 1) % 2 != 0:
      if a[i] < b[i]:
        return -1
      elif a[i] > b[i]:
        return 1
    elif a[i] > b[i]:
      return -1
    elif a[i] < b[i]:
      return 1
  return 0
def solution(stdin: str) -> str:
  lines = stdin.strip().split('\n')
  (n, m) = map(int, lines[0].split())
  books = [(title, idx) for (idx, title) in
enumerate(lines[1:], start=1)]
  books.sort(key=cmp_to_key(lambda x, y:
asc_desc_ending_comparator(x[0], y[0])))
  output = [str(book[1]) for book in books]
  return ' '.join(output)
```

Figure 14: The prompt we used for oracle generation and one oracle generated with it. The instructions are in blue. The language model is instructed to generate the most straightforward solution by enumerating over a very large search space of all combinations of relevant variables.

A.3 The List of Problems from LeetCode

We list the problems we collected from LeetCode as benchmarks to test ALGO in Table 3, among them are 10 easy problems, 18 medium problems, and 7 hard problems.

Batch Generator Generation Prompt

```
You are given this problem.
Andi and Budi were given an assignment to tidy up their bookshelf of n books.
Each book is represented by the book title — a string si numbered from 1 to n,
each with length m. Andi really wants to sort the book lexicographically
ascending, while Budi wants to sort it lexicographically descending.
(https://codeforces.com/problemset/problem/1575/A, remaining part omitted)

You should first create an input generator function `gen_input`.`gen_input` should take
a list of arguments to control the upper limit of the input and return a single string
that represents the input.`gen_input` should only generate test inputs that follow the
problem constraints. `gen_input` should only generate test inputs and not test outputs.

You should then create an extra function `batch_gen_inputs` that takes one argument —
`batch_size`. `batch_gen_inputs` should run `gen_input` for `batch_size` times with a
small upper limit to make sure that it works. `batch_gen_inputs` should return a list of
strings that represent the inputs.
```

Batch Generator Generated by LLM

```python
def gen_input(n_max: int, m_max: int) -> str:
  n = random.randint(1, n_max)
  m = random.randint(1, min(m_max, 10 ** 6 // n))
  book_titles = set()
  while len(book_titles) < n:
    book_title =
''.join([random.choice(string.ascii_uppercase) for _ in
range(m)])
    book_titles.add(book_title)
  input_string = f'{n} {m}\n' + '\n'.join(book_titles)
  return input_string

def batch_gen_inputs(batch_size,) -> list:
  inputs = []
  for _ in range(batch_size):
    inputs.append(gen_input(5, 10))
  return inputs
```

Figure 15: The prompt is used to first generate a function `gen_input` and then generate a batch generator that employs the already generated `gen_input` to generate several input cases by calling it multiple times.

Table 3: The Leetcode problems we use. We only pick problems that were released concurrently or after GPT-4 to avoid contamination.

Problem ID	Problem Name	Level
2582	pass-the-pillow	easy
2583	kth-largest-sum-in-a-binary-tree	medium
2584	split-the-array-to-make-coprime-products	hard
2585	number-of-ways-to-earn-points	hard
2586	count-the-number-of-vowel-strings-in-range	easy
2587	rearrange-array-to-maximize-prefix-score	medium
2588	count-the-number-of-beautiful-subarrays	medium
2589	minimum-time-to-complete-all-tasks	hard
2591	distribute-money-to-maximum-children	easy
2592	maximize-greatness-of-an-array	medium
2593	find-score-of-an-array-after-marking-all-elements	medium
2594	minimum-time-to-repair-cars	medium
2595	number-of-even-and-odd-bits	easy
2596	check-knight-tour-configuration	medium
2597	the-number-of-beautiful-subsets	medium
2598	smallest-missing-non-negative-integer-after-operations	medium
2600	k-items-with-the-maximum-sum	easy
2601	prime-subtraction-operation	medium
2602	minimum-operations-to-make-all-array-elements-equal	medium
2603	collect-coins-in-a-tree	hard
2609	find-the-longest-balanced-substring-of-a-binary-string	easy
2610	convert-an-array-into-a-2d-array-with-conditions	medium
2611	mice-and-cheese	medium
2612	minimum-reverse-operations	hard
2614	prime-in-diagonal	easy
2615	sum-of-distances	medium
2616	minimize-the-maximum-difference-of-pairs	medium
2617	minimum-number-of-visited-cells-in-a-grid	hard
2639	find-the-width-of-columns-of-a-grid	easy
2640	find-the-score-of-all-prefixes-of-an-array	medium
2641	cousins-in-binary-tree-ii	medium
2643	row-with-maximum-ones	easy
2644	find-the-maximum-divisibility-score	easy
2645	minimum-additions-to-make-valid-string	medium
2646	minimize-the-total-price-of-the-trips	hard