Learning Task Decomposition to Assist Humans in Competitive Programming

Jiaxin Wen, Ruiqi Zhong,
Pei Ke, Zhihong Shao,
Honging Wang, Minlie Huang

Background

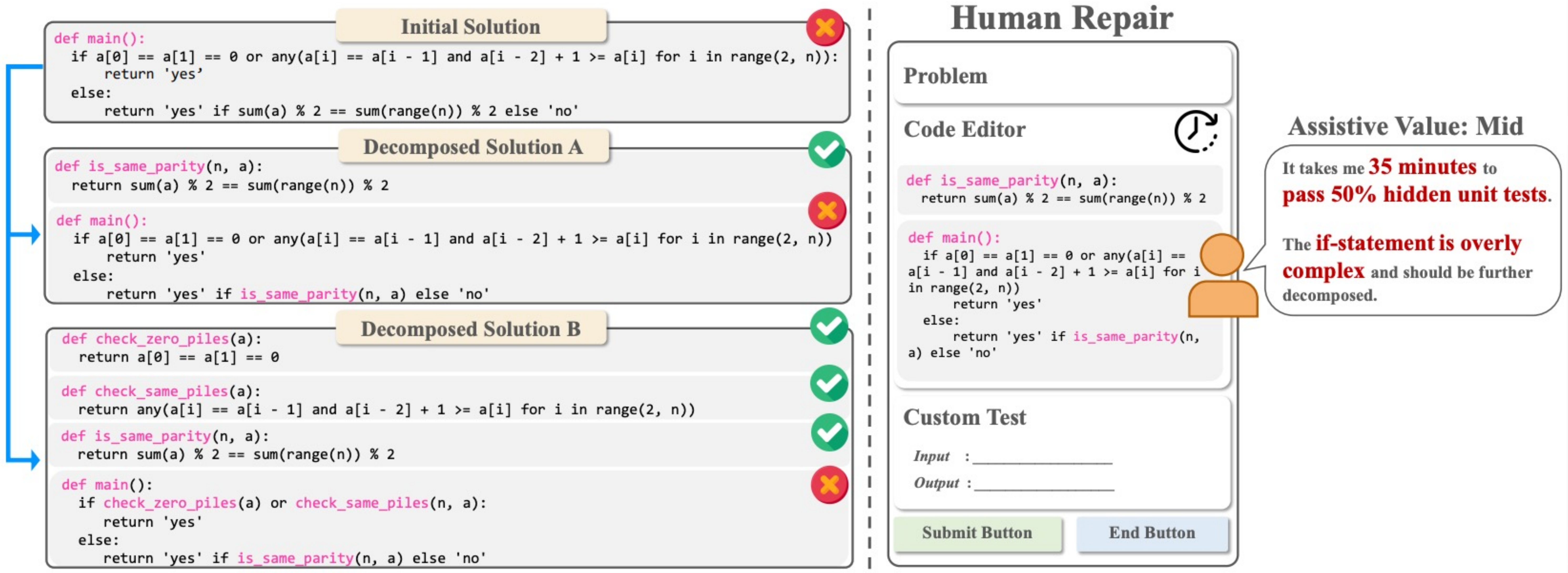
While LMs are being used to solve increasingly complex tasks, LMs might fail to provide reliable solutions.
However, humans also struggle to understand and repair LMs' solutions due to the required time and expertise.

Task

We aim to assist non-expert humans to solve competitive programming problems faster and better, matching the performance of expert humans.

To this end, we use LMs to generate decomposed subtasks and sub-solutions that are easier to understand and fix by humans.

We measure and optimize the **Assistive Value** of LM–generated programs



Initial Solution ✗

```
def main():
  if a[0] == a[1] == 0 or any(a[i] == a[i - 1] and a[i - 2] + 1 >= a[i] for i in range(2, n)):
      return 'yes'
  else:
      return 'yes' if sum(a) % 2 == sum(range(n)) % 2 else 'no'
```

Decomposed Solution A

✓
```
def is_same_parity(n, a):
  return sum(a) % 2 == sum(range(n)) % 2
```

✗
```
def main():
  if a[0] == a[1] == 0 or any(a[i] == a[i - 1] and a[i - 2] + 1 >= a[i] for i in range(2, n))
      return 'yes'
  else:
      return 'yes' if is_same_parity(n, a) else 'no'
```

Decomposed Solution B

✓
```
def check_zero_piles(a):
  return a[0] == a[1] == 0
```

✓
```
def check_same_piles(a):
  return any(a[i] == a[i - 1] and a[i - 2] + 1 >= a[i] for i in range(2, n))
```

✓
```
def is_same_parity(n, a):
  return sum(a) % 2 == sum(range(n)) % 2
```

✗
```
def main():
  if check_zero_piles(a) or check_same_piles(n, a):
     return 'yes'
  else:
     return 'yes' if is_same_parity(n, a) else 'no'
```

Human Repair

```
def is_same_parity(n, a):
  return sum(a) % 2 == sum(range(n)) % 2
```

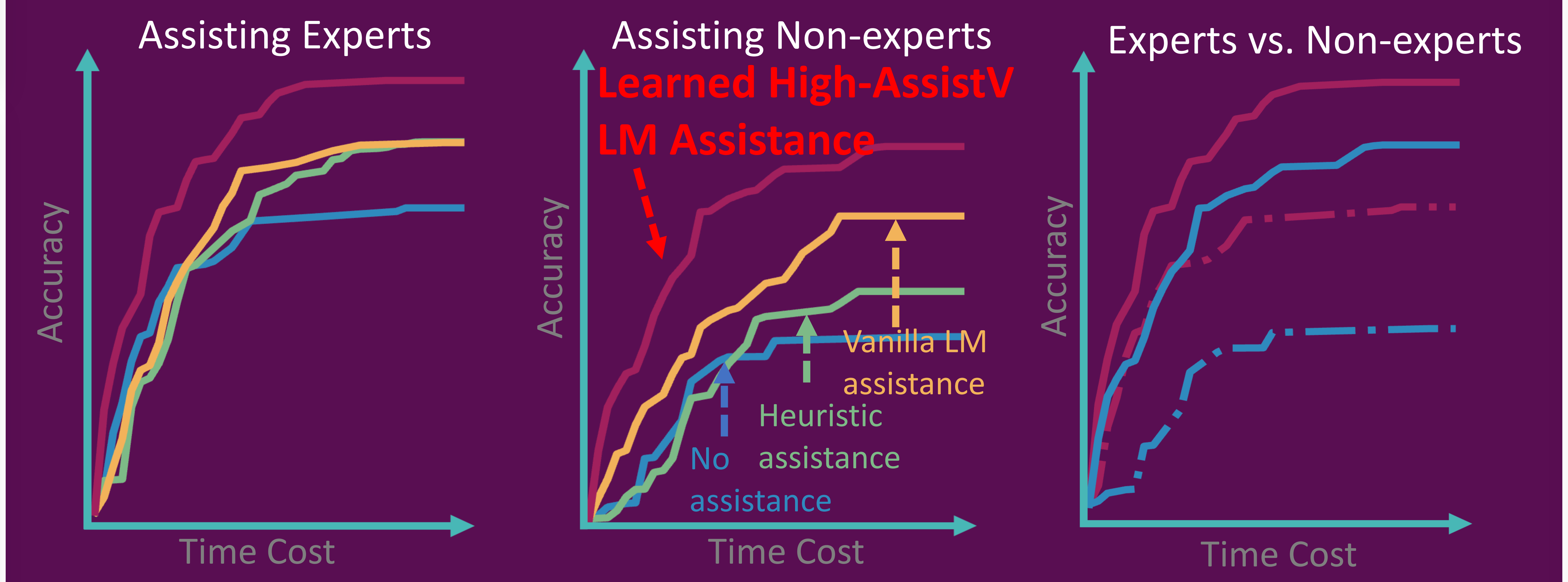
```
def main():
  if a[0] == a[1] == 0 or any(a[i] ==
a[i - 1] and a[i - 2] + 1 >= a[i] for i
in range(2, n))
      return 'yes'
  else:
      return 'yes' if is_same_parity(n,
a) else 'no'
```

Assistive Value: Mid

It takes me **35 minutes** to **pass 50% hidden unit tests**.

The **if-statement is overly complex** and should be further decomposed.

Assisted non-experts can solve 33.3% more code challenges, work 3.3x faster, and match unassisted experts



Take a picture to download the full paper

Assistive Value (AssistV) of a Program

Can it assist humans to quickly obtain a correct program, even when the program itself is wrong?

Method

- Collect AssistV labels on various code decompositions
- Learn to generate high-AssistV code decompositions by critiquing, refining, and ranking.

Takeaways

1. A novel objective for scalable oversight: Assistive Value.
   - We explore AssistV in programming.
   - Future work can extend AssistV to other domains (e.g., QA, Summarization)
2. Learning-based scalable oversight is promising.
   - **LMs can learn to better assist humans** in solving problems **beyond their capabilities.**
   - LMs' **assistance performance scales with their capabilities,** sometimes even outperforming human baselines.