# Problem Decomposition for Program Synthesis with Large Language Models

SURF
Summer Undergraduate Research Fellowship

SURF PROJECT CODE: 20230040

Supervisor: Chunchuan Lyu     Authors: Xu Chen, Zichen Qiu, Yanzhuo Jin, Jieran He, Zhaohui Yang

## Abstract

*Program synthesis generates computer programs based on given problem specifications. Large Language Models have been proven to be effective. However, models are still struggling to handle complicated tasks involving multiple steps. Recently, a multi-turn code generation benchmark has been proposed. They provide pre-defined individual steps in natural language. However, this setting is not very useful in practice. Instead of starting from a pre-decomposed steps, we aim to automatically decompose the a short problem specification into individual steps. We reverse-engineered the multi-turn benchmark to include a short summary specification. We empirically examined the ability of current models to automatically decompose the summary specification and complete the task given the decomposed steps.*

## Introduction

***Our contributions :***

- *We construct a dataset based on MTPB by adding a summary instruction that summarized the multi-step instructions in MTPB*
- *We use prompt engineering to automatically decompose our summary prompt and enhance our result with the chain-of-thought prompting.*

***Findings:***

- *LLMs can automatically decompose a short instruction into individual steps*
- *LLMs can complete the code better with both generated and gold individual steps.*
- *LLMs may prefer copying example answers, but this can be alleviated via corrective prompting*

## Related Work

### A. Large Language Models

*Recently, autoregressive Transformer models[2] with billions of parameters pretrained on terabytes of data have been used to model the conditional distribution of natural language. By further training on a wide range of tasks, they can complete new tasks without training[3]. We will use GPT3.5-turbo in our study.*

### B. Multi-Turn Program Synthesis

*Program synthesis is the generation of computer programs based on problem specifications. A program specification could involve multi-turn steps. To investigate the multi-step paradigm for program synthesis, researchers introduced Multi-Turn Programming Benchmark (MTPB) which consists of 115 diverse problem sets that are factorized into multi-turn prompts[1]. While MTPB offers a manual breakdown of each step for a given problem, rarely users will specify coding steps in such detail. We aim to provide a more realistic setting where LLMs start with a short but sufficient problem specification.*

### C. Chain of Thought prompting

*Chain of Thought (CoT) prompting provides demonstrations as exemplars in prompting. CoT teaches LLMs to solve problems by leveraging LLMs' ability to in-context learn new patterns. We use CoT examplars to teach LLMs to decompose the summary specification and provide well-formatted code. [4]*

```
INPUTS:
Let's think step by step\
As the examples belowed\
Example
   Q: Enter an integer array (list), and the function outputs the closest integer to the average of the array (rounded down if equidistant)\
   For example,inputs [1,2,3,4.1] outputs is 3\
   To help generate the code, the steps are given\
   1. The function is called "find_closest_integer".\
   2. It takes one parameter, "array", which is an integer array (list)\.
   3. The purpose of the function is to find the closest integer to the average of the array.\
   4. To find the average of the array, we sum up all the elements and divide it by the length of the array.\
   5. We then find the closest integer to the average by comparing the absolute difference between each element and the average.\
   6. If there are multiple integers equidistant from the average, we round down to the lower integer.\
   7. The function should return the closest integer to the average.\
   8. If the array is empty, the function should return None.\

   A: 'Firstly,we can know that the value is a list with variables. \
   As we need to find the averages, we need to know the function average=sum substracts the number of variables\
   we can use len(inputs) to get the number of variables.\
   to get the sum ,we can use (for i in inputs)to get variables and save it with s,like s=s+i\
   After we get the average we can use \\ to get the range\
   compare the distance between average\\1 and average\\1+1\
   Finally , we can get the answer\
   'def closest_integer(value):\
     s=0\
     for i in value:\
       s=s+i\
     average=s/len(value)\
     if (average\\1-average)^2 > (average\\1+1-average)^2:\
        interger=average\\1+1\
     else:\
        interger=average\\1\
     return interger
 I want you to write a  a function or class in python code   \
the task is {summary_prompt}\'
```

*Auto-D+CoT:To help you generate the code, i will give you each step\The steps are :{multi_step}*

*God-D+CoT:To help you generate the code, i will give you each step\The steps are :{prompts}*

```
Auto-D+CoT:

code_json= {
  "code": "def find_k_nearest_points(points, k):\n    distances = []\n    for point in points:\n        distance = (point[0]**2 + point[1]**2)**0.5\n        distances.append(distance)\n    sorted_distances = sorted(distances)\n    nearest_points = []\n    for i in range(k):\n        index = distances.index(sorted_distances[i])\n        nearest_points.append(points[index])\n    return nearest_points",
  "explanation": "The code defines a function called 'find_k_nearest_points' that takes two parameters: 'points' and 'k'. It calculates the Euclidean distance for each point in the 'points' array and stores them in the 'distances' array. Then, it sorts the 'distances' array in ascending order. Next, it iterates through the first 'k' elements of the sorted 'distances' array and finds the corresponding index in the 'points' array. Finally, it appends the points with the corresponding indices to the 'nearest_points' array and returns it.
```

```
God-D+CoT:

code_json ={
"code": "def distance(point):\n    return (point[0]**2 + point[1]**2)**0.5\n\nmy_points = [[1, 3], [2, 2]]\nk = 1\n\nk_closest_points = sorted(my_points, key=distance)[:k]\nprint(k_closest_points)", "explanation": "Define a function 'distance' that calculates the distance between a point and the origin. Assign the list of points to 'my_points' and the integer 'K' to 'k'. Sort 'my_points' based on the distance from the origin and store the k closest points in 'k_closest_points'. Finally, print 'k_closest_points'."}
```

***Figure4: The Programmer Prompts and code results of Auto-D+CoT& Gold +CoT of id66***

## Dataset and Methods

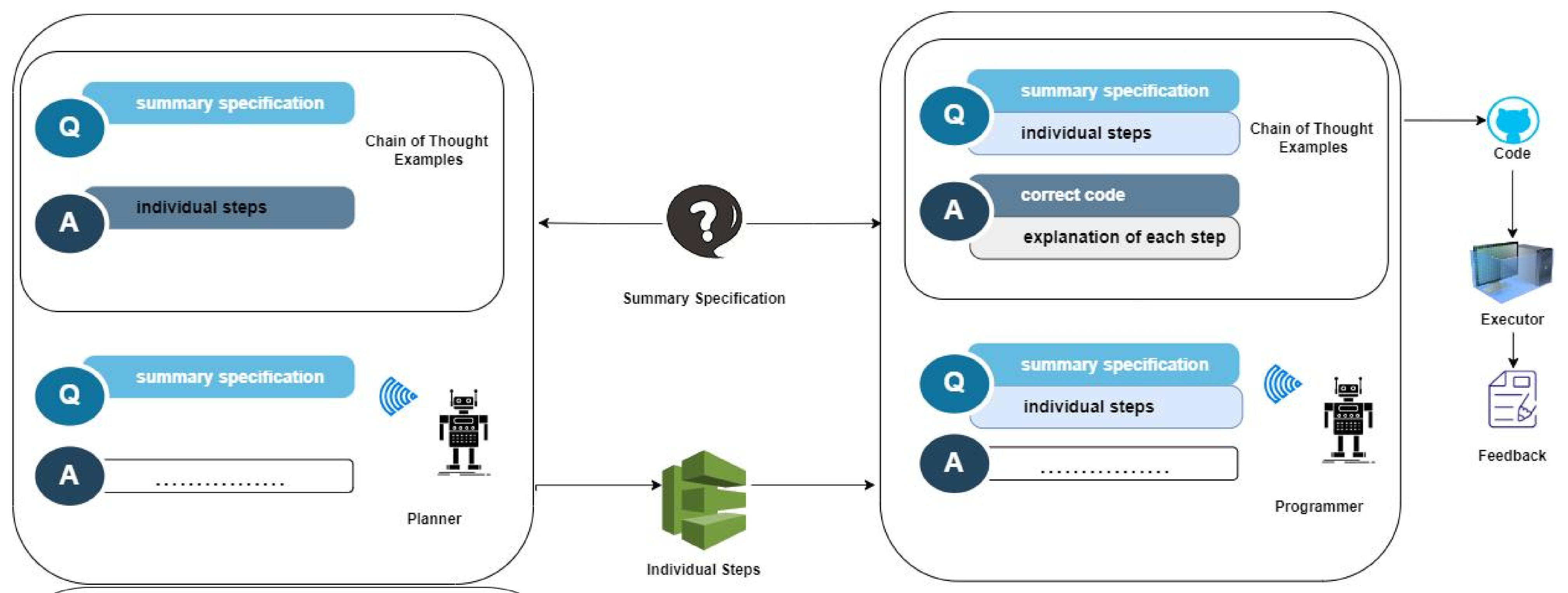


***Figure1: The flow path of Auto-D with CoT***

```
EMTPB/id1:

"summary_prompt":"Define a function to return a sorted list of distinct characters obtained from a given string, after converting it to lowercase."

"test": "def check(candidate):\n assert candidate(\"abcdecadeCADE\")==[\"a\", \"b\", \"c\", \"d\", \"e\"] \n assert candidate(\"Jerry jERRY JeRRRY\")==[\" \",\"e\",\"j\", \"r\", \"y\" ]\n",

"prompts":["Assign the string \"{A}\" to a variable named \"my_string\".", "Lowercase the given string \"my_string\".", "Assign the distinct characters of the string to a variable named \"chars\".", "Sort these characters in alphabetical order.", "Return the resulting list of characters."],

"compressible":"True"

"example":"input \"abcdj\" \n output: \"a\", \"b\", \"c\", \"d\", \"j\""
```

***Figure2: Instance of Data***

```
INPUTS:

Let's think step by step\
For example \
the question:"Given the lengths of the four sides of the quadrilateral in an array, judge whether the quadrilateral is Cyclic"\
the answer will be :\
"1.Assume the function is called "is_cyclic_quadrilateral"\
2.It takes one parameter, "sides", which is an array containing the lengths of the four sides of a quadrilateral.\
3.The purpose of the function is to determine whether the quadrilateral is cyclic or not.\
4.A cyclic quadrilateral is a quadrilateral that can be inscribed in a circle, meaning that all four vertices of the quadrilateral lie on the circumference of a circle.\
5.To determine if a quadrilateral is cyclic, we need to check if the sum of opposite angles is equal to 180 degrees.\
6.If the sum of opposite angles is equal to 180 degrees, then the quadrilateral is cyclic.\
7.If the sum of opposite angles is not equal to 180 degrees, then the quadrilateral is not cyclic.\
8.The function should return a boolean value, True if the quadrilateral is cyclic, and False if it is not."\
Now the question is :{summary_prompt}\
please give me the answer in the similar way\

OUTPUTS:

multi_step=['####', '    Here is the answer to the question "Find the k nearest points to the origin":', '', '    1. The function is called "find_k_nearest_points".', '    2. It takes two parameters, "points" and "k". "points" is an array containing the coordinates of multiple points in the form (x, y), and "k" is the number of nearest points to find.', '    3. The purpose of the function is to find the k nearest points to the origin (0, 0).', '    4. To find the distance between each point and the origin, we can use the Euclidean distance formula: distance = sqrt(x^2 + y^2).', '    5. We can calculate the distance for each point and store it in a separate array.', '    6. Then, we can sort the array of distances in ascending order.', '    7. Finally, we can return the first k points from the sorted array as the k nearest points to the origin.', '    8. The function should return an array containing the k nearest points.', '', '    Example usage:', '    points = [(1, 2), (3, 4), (5, 6), (7, 8)]', '    k = 2', '    nearest_points = find_k_nearest_points(points, k)', '    print(nearest_points)  # Output: [(1, 2), (3, 4)]']
```

***Figure3: The Planner Prompt and decomposition result of id66***

### A. The Construction of Summary Specification

*The **summary specification** should be short, but sufficiently descriptive of the tasks defined by multiple steps. Five annotators manually read all the individual steps in all **MTPB** and one lead annotator checked the validity of summary specification with the help of LLMs. The average length of our summary specification is **15** and average length of all individual steps is **55***
*We find that some problems in MTPB are simple concatenation of almost irrelevant processing steps that cannot be summarized. We split the data into two groups: compressible and un-compressible. There are **105** compressible cases and **10** un-compressible cases.*

### B. Auto Decomposition with Chain of Thought

***Baseline:** The GPT3.5 model is given only the summary specification to complete the task, and it should directly produce the code.*
***Automatic decomposition (Auto-D):** The GPT3.5 model is given only the summary specification to complete the task, but it is asked to provide a decomposition of summary specification into each steps then provide the code.*
***Gold decomposition (Gold-D):** Given both the summary specification and the individual steps in original MTBP.*
***Automatic decomposition with Chain of Thought (Auto-D + CoT):** In addition to Auto-D, provided with a example of decomposition process and answer.*
***Gold decomposition with Chain of Thought (Gold-D + CoT):** In addition to Gold-D, provided with a example of decomposition and answer.*

### C. Corrective Prompting to Alleviate Copying from CoT Examples

*We find that when using CoT, GPT3.5 may prefer copying from examples. This cause the model to deviate from solving the given problem. We explicitly add "You must not copy the code about the question example" into the prompts to avoid this issue.*

## Experiments and Results

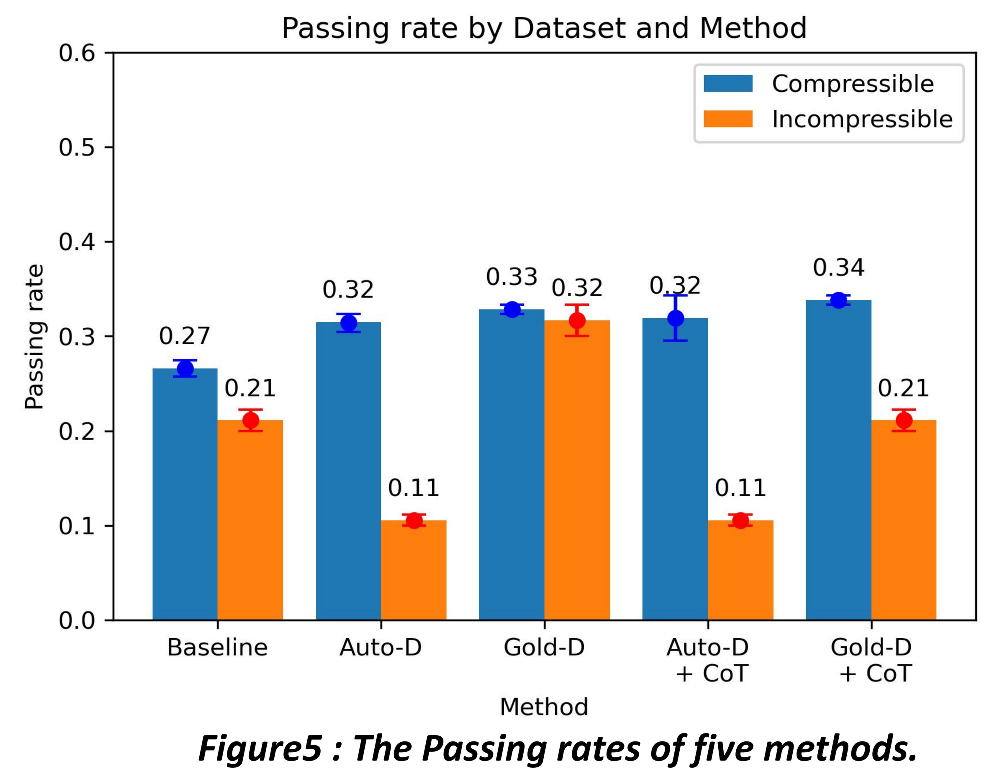


***Figure5 : The Passing rates of five methods.***

We can see that programmer can do better with decomposed steps. So, GPT3.5 can perform reasonable planning. However, while CoT improves Gold-D, the improvement is not significant.

*In ablation study, we have found that the Passing rate of Auto-D+CoT&Gold-D+CoT without " You must not copy the code about the question example" is 32.38% and 34.29% (compared with 33.33% and 34.31%)*

## Conclusion and Discussion

*In this study, we show that LLMs can automatically decompose user intention into individual steps, and its' performance of code completion can benefit from relying on decomposed steps. However, our Chain-of-Thought prompt might need more tuning. Overall, our study is limited in terms of the models we have tested and the size and variety of coding task. It is possible that better models such as GPT4 can decompose the instruction better or it might be more capable or solving the problem without explicit decomposition.*

### Reference


*[1] Nijkamp E, Pang B, Hayashi H, et al. Codegen: An open largelanguage model for code with multi-turn program synthesis[J]. arXiv preprint arXiv:2203.13474, 2022.*
*[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, L ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, volume 30, 2017.*
*[3] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.*
*[4] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in Neural Information Processing Systems, 2022, 35:24824-24837.*


*code&report(vpn only)*