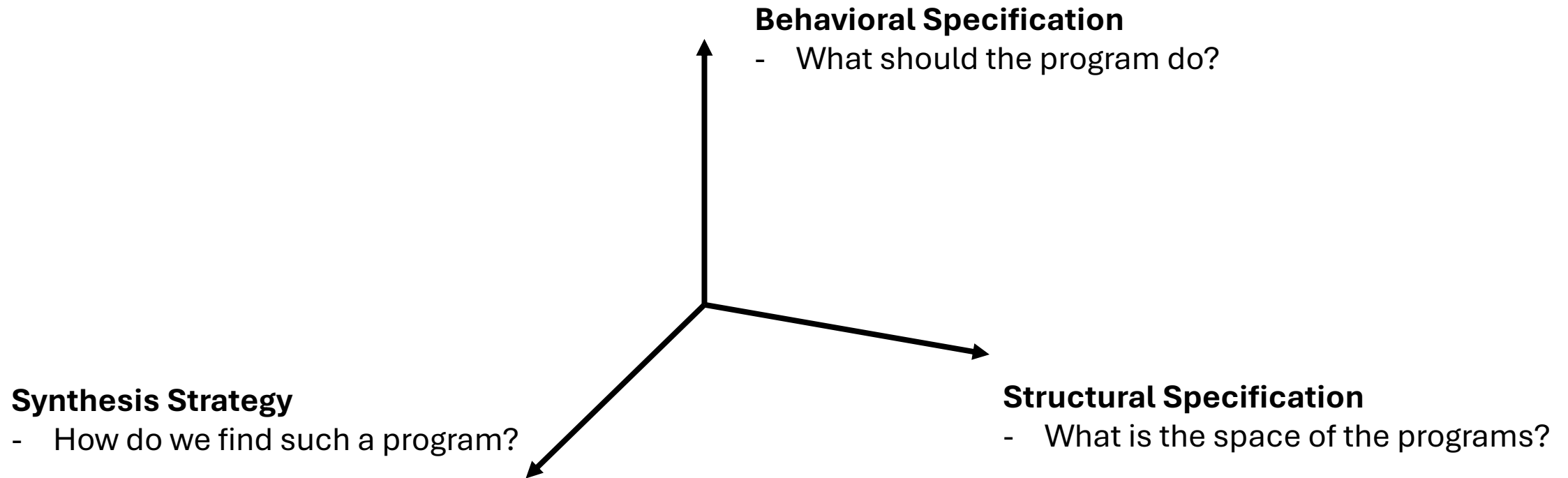


Machine Programming

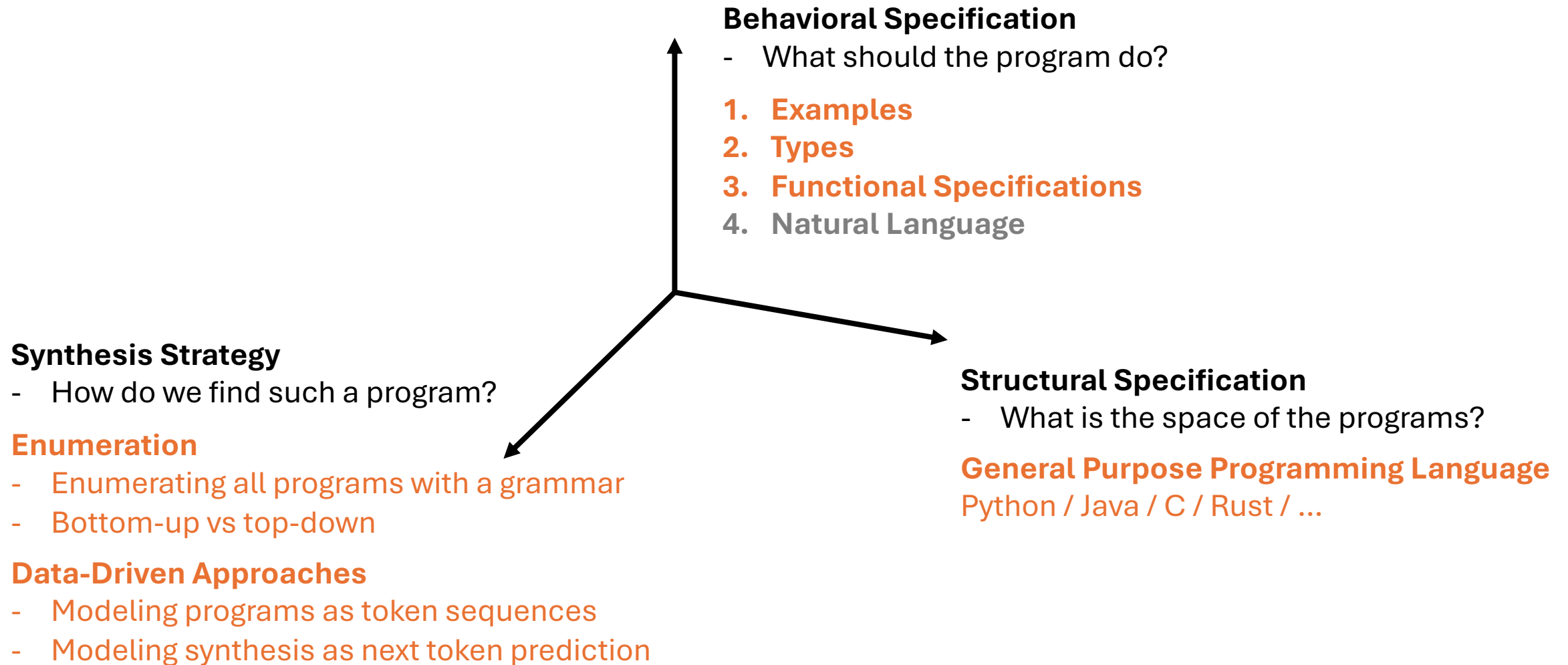
Lecture 6 – Decoding and Prompting for Program Synthesis

Ziyang Li

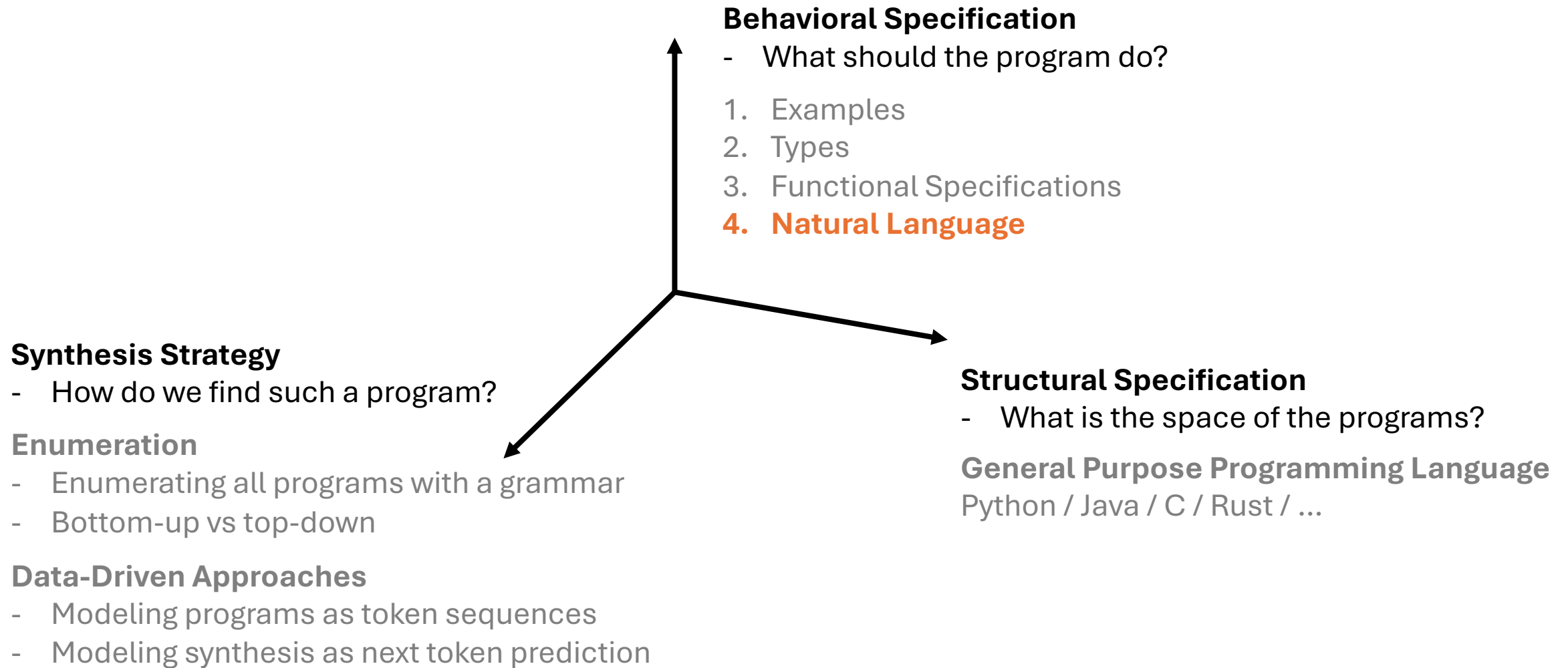
Dimensions in Program Synthesis



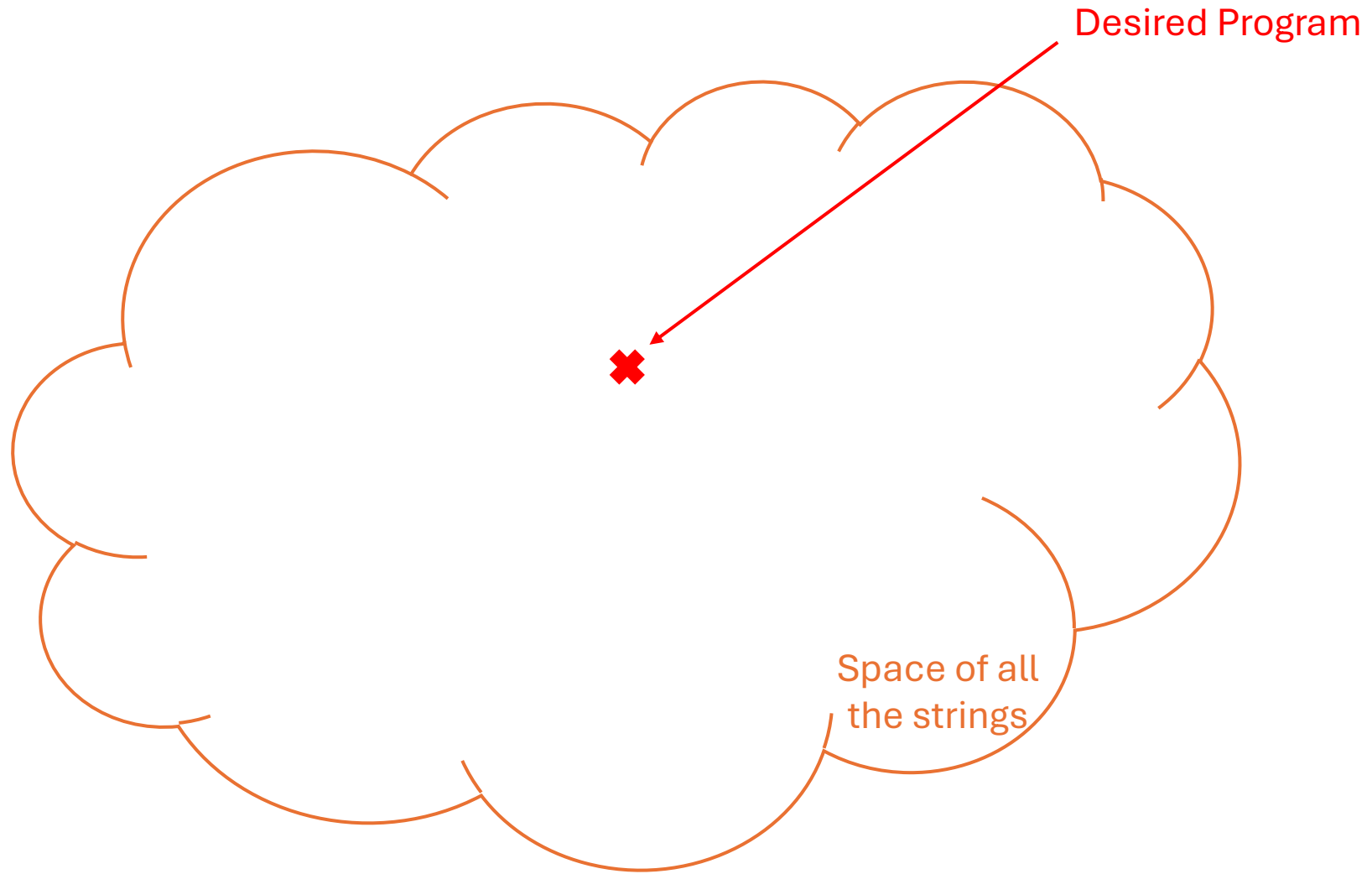
The Course So Far



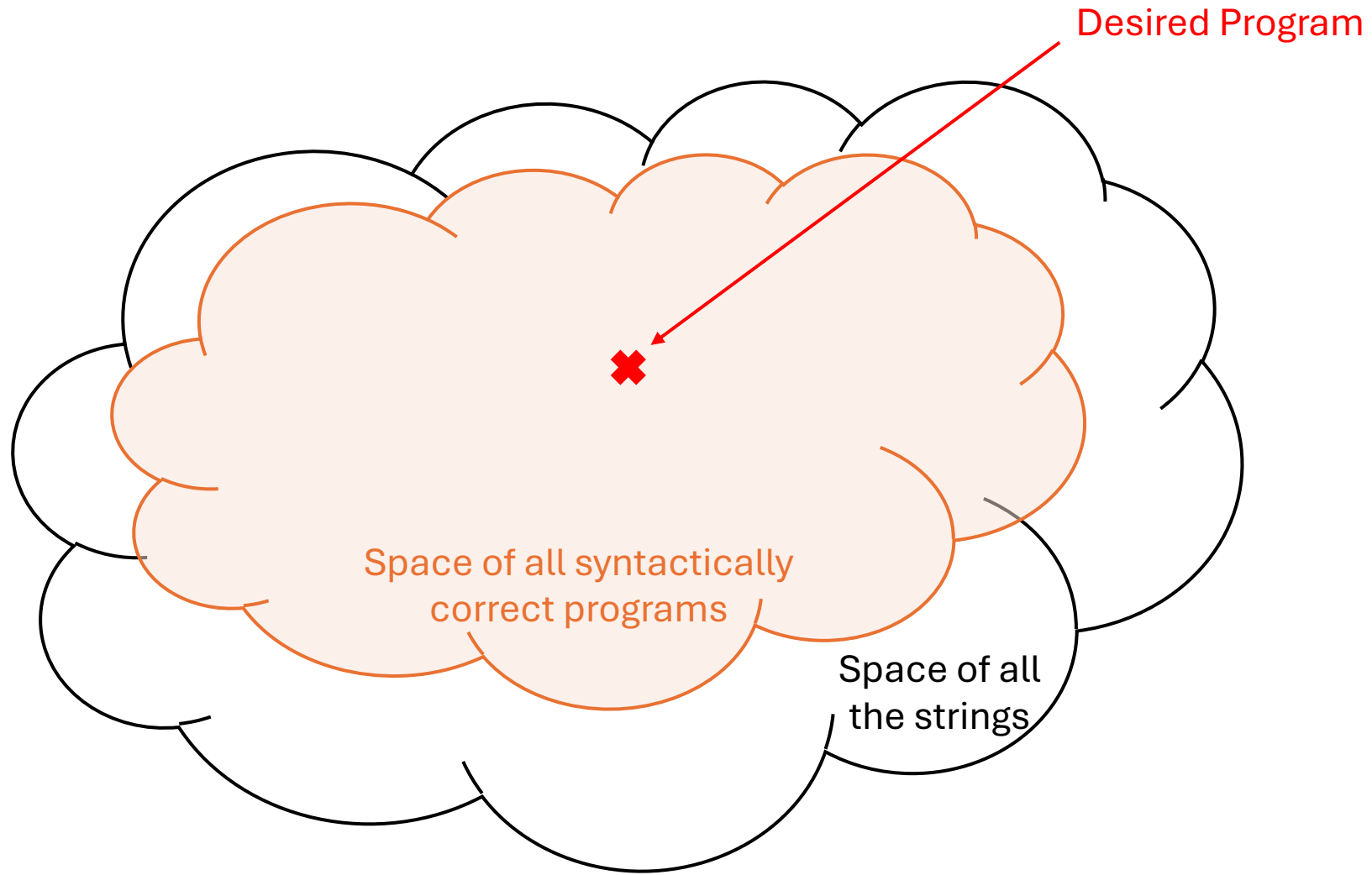
Today



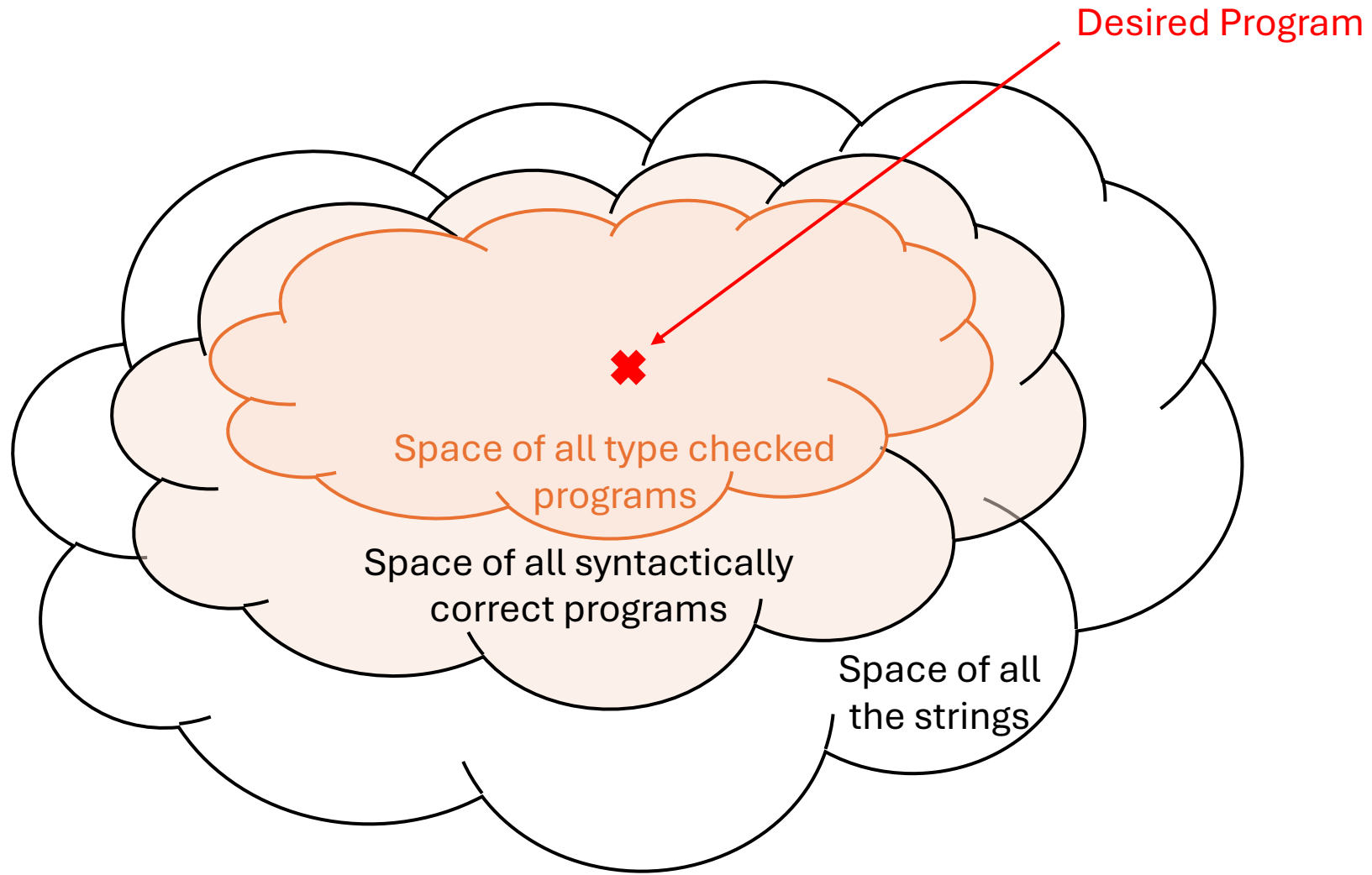
High Level Picture



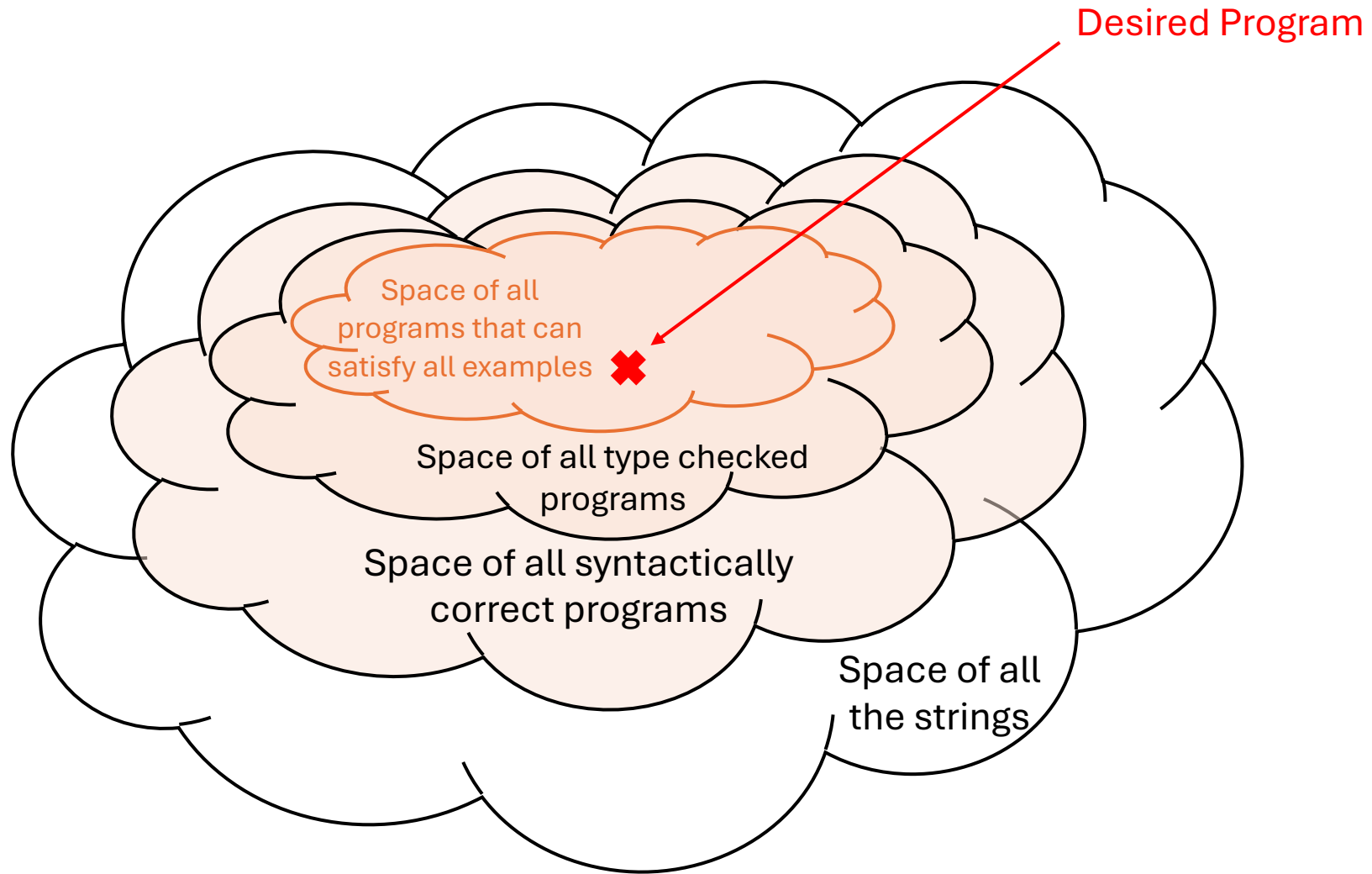
High Level Picture



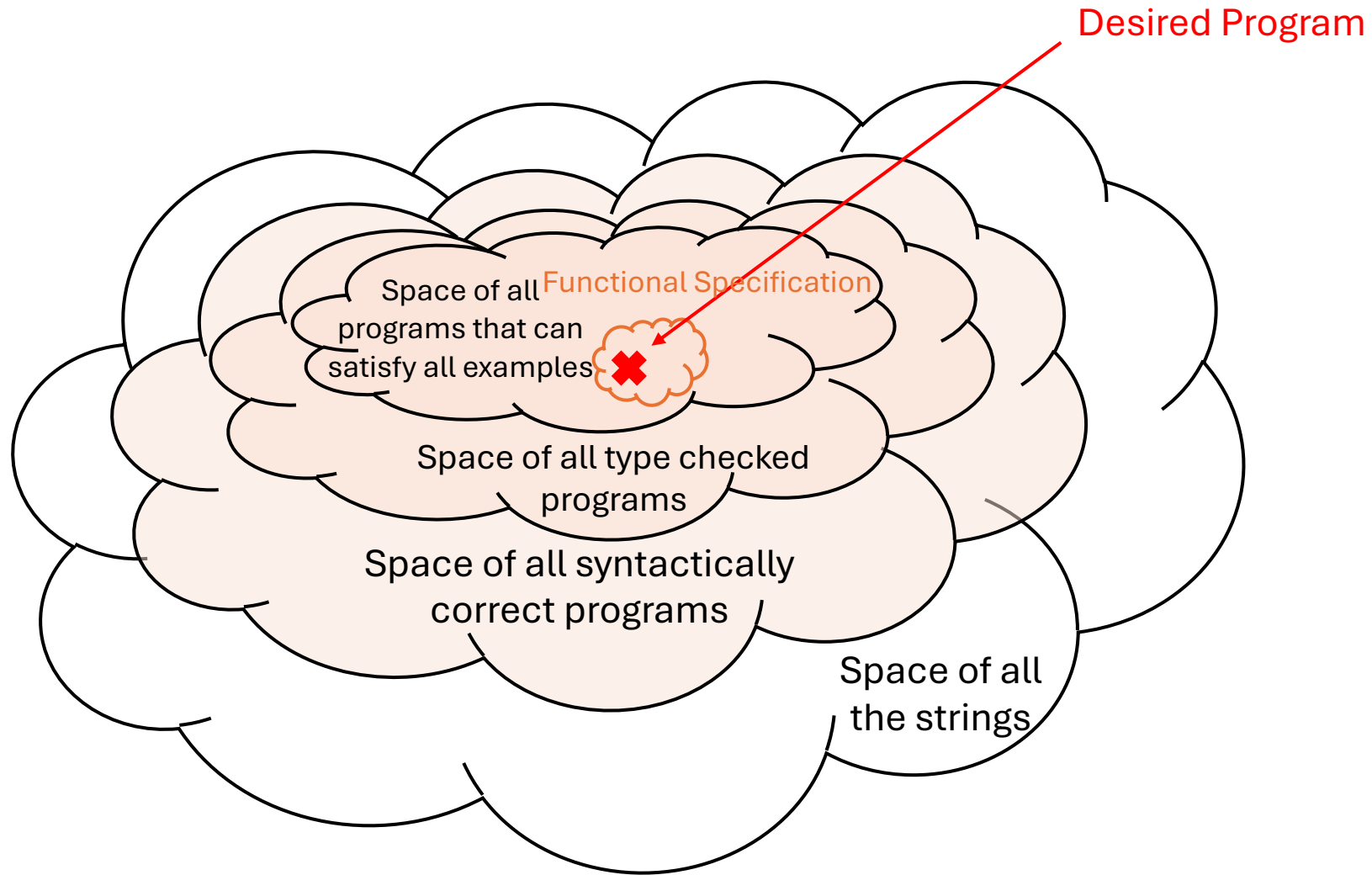
High Level Picture



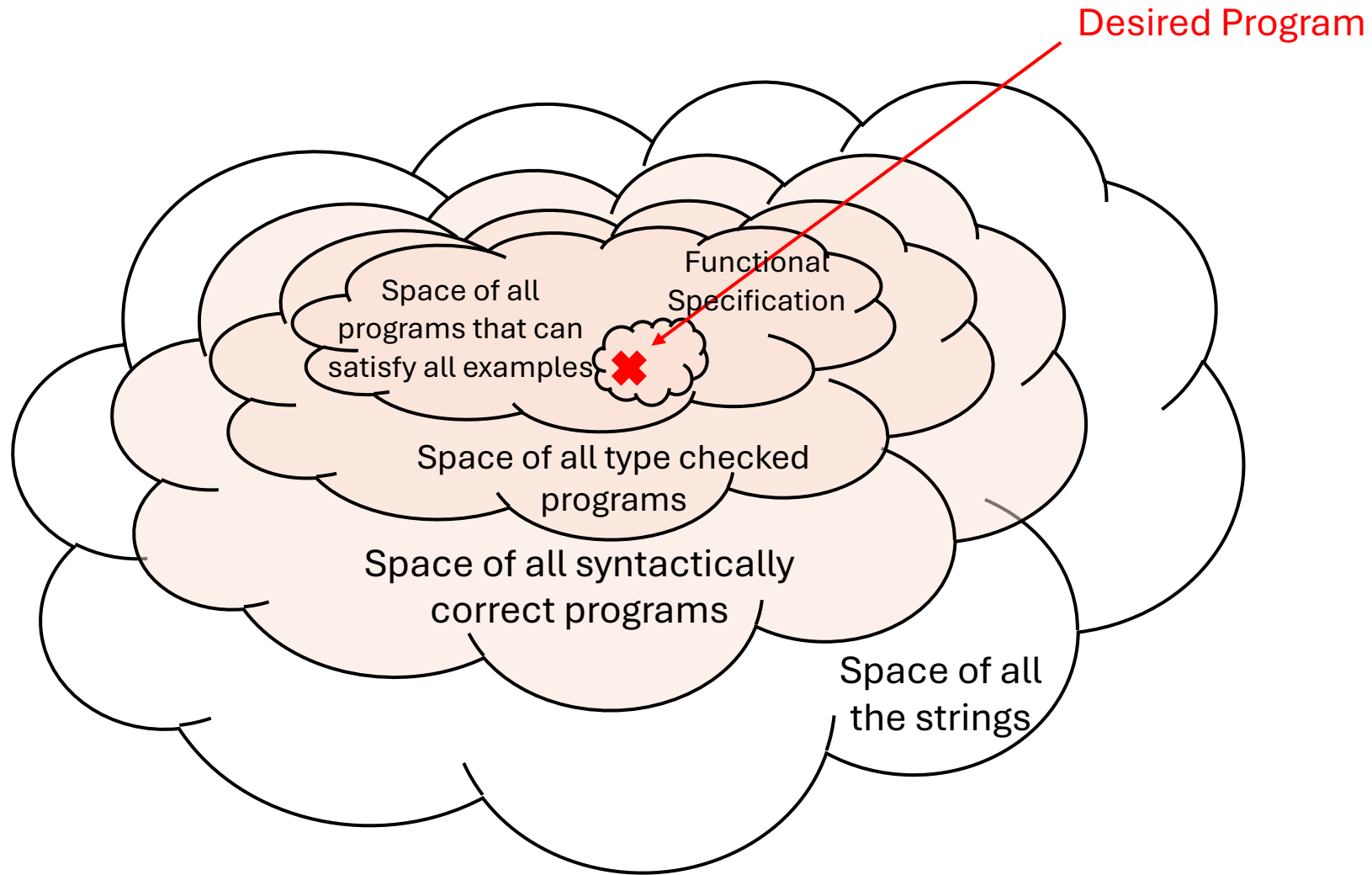
High Level Picture



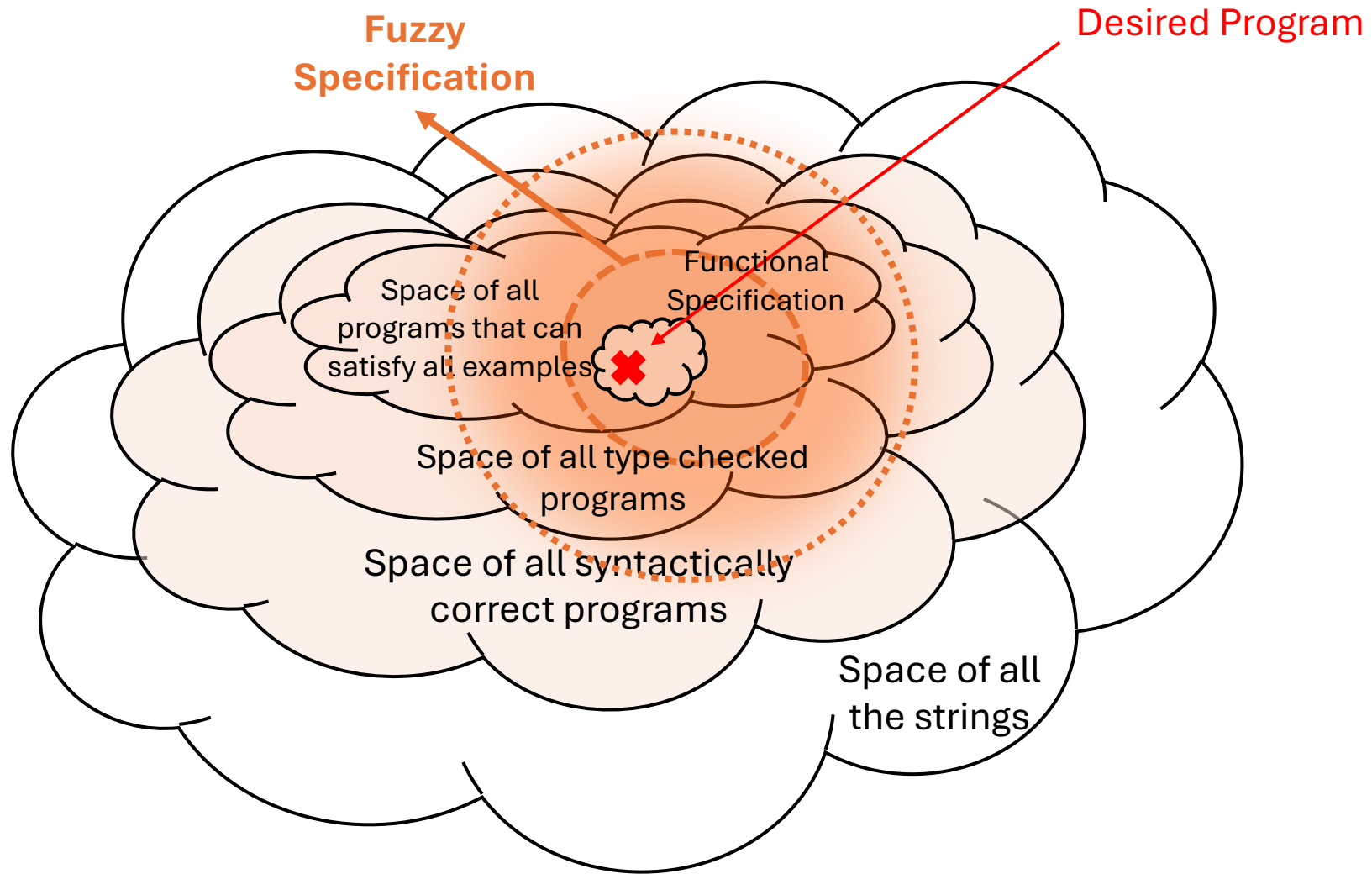
High Level Picture



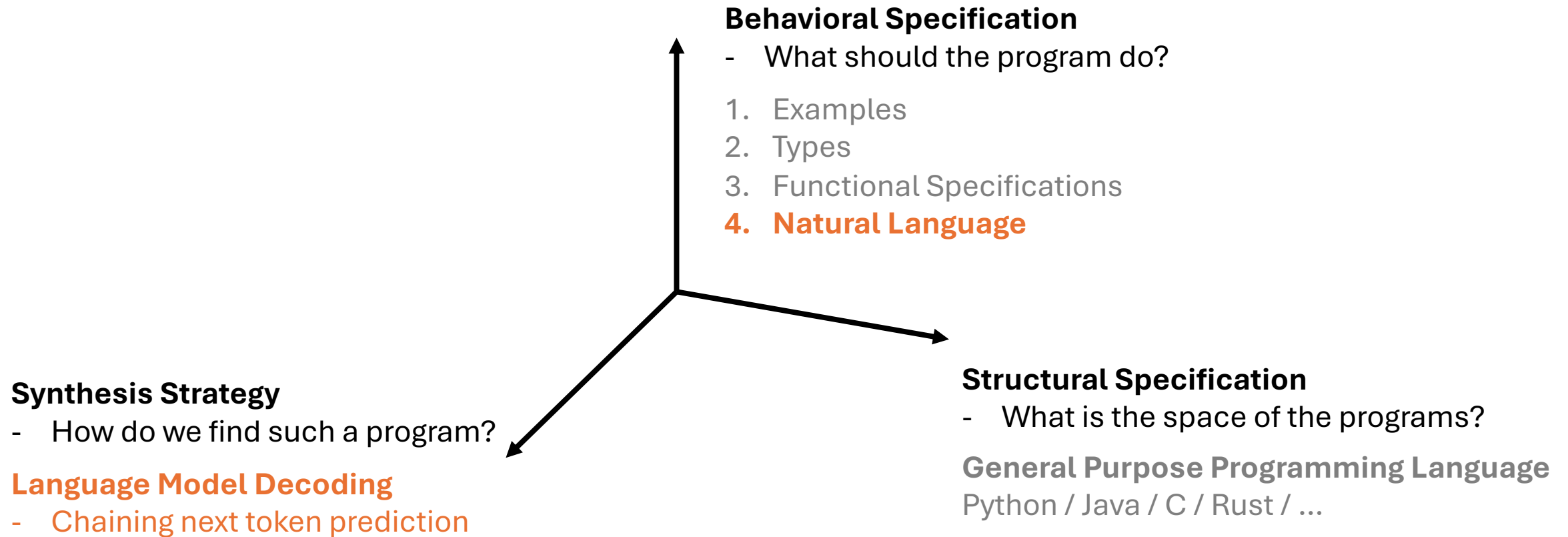
High Level Picture



High Level Picture



Today



What's Next from Next Token Prediction

Problem Definition: **Next token prediction**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Next Token** y_i

Input: **Generated tokens so far** y_1, y_2, \dots, y_{i-1}

Goal: Compute the probability $\Pr(y_i \mid y_1, y_2, \dots, y_{i-1}, \mathbf{x})$

What's Next from Next Token Prediction

Problem Definition: **Next token prediction**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Next Token** y_i

Input: **Generated tokens so far** y_1, y_2, \dots, y_{i-1}

Goal: Compute the probability $\Pr(y_i \mid y_1, y_2, \dots, y_{i-1}, \mathbf{x})$

Chaining next token predictions:

```
[for, i, in] → range
  [for, i, in, range] → (
    [for, i, in, range, ] → 10
      [for, i, in, range, (, 10] → )
        [for, i, in, range, (, 10, )] → :
```

What's Next from Next Token Prediction

Chaining next token predictions:

[for, i, in] → range

[for, i, in, range] → (

[for, i, in, range, (] → 10

[for, i, in, range, (, 10] →)

[for, i, in, range, (, 10,)] → :

...



When do we stop?

What's Next from Next Token Prediction

Chaining next token predictions:

[for, i, in] → range

[for, i, in, range] → (

[for, i, in, range, (] → 10

[for, i, in, range, (, 10] →)

[for, i, in, range, (, 10,)] → :

...

When do we stop?

[for, i, in, range, ..., print, (, i,)] → <EOS>

When a special token <EOS> is generated

What's Next from Next Token Prediction

Problem Definition: **Sequential Decoding**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output \mathbf{y} from $\Pr(\mathbf{y} \mid \mathbf{x})$

```
from transformers import AutoModelForCausalLM, AutoTokenizer

tok = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

inputs = tok("def add(a, b):", return_tensors="pt")
logits = model(**inputs).logits
probs = logits.softmax(-1)
print(tok.decode(probs[0, -1].topk(5).indices))
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer

tok = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

inputs = tok("def add(a, b):", return_tensors="pt")
logits = model(**inputs).logits
probs = logits.softmax(-1)
print(tok.decode(probs[0, -1].topk(5).indices))
```

Top 5 Tokens:	["\n" , "return", "#" , "if" , "a"]
Top 5 Token IDs:	[198 , 1441 , 1303 , 611 , 257]
Top 5 Token Probabilities:	[0.1946, 0.1221 , 0.0448, 0.0425, 0.0416]

What's Next from Next Token Prediction

Problem Definition: **Sequential Decoding**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output \mathbf{y} from $\text{Pr}(\mathbf{y} \mid \mathbf{x})$

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    probs = softmax(predict_next_token(current_sequence))
    next_token = argmax(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

What's Next from Next Token Prediction

Problem Definition: **Sequential Decoding**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output \mathbf{y} from $\Pr(\mathbf{y} | \mathbf{x})$

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    probs = softmax(predict_next_token(current_sequence))
    next_token = argmax(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

Greedy Decoding: Take Top-1 Prediction

Decoding: Sampling vs Greedy

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    probs = softmax(predict_next_token(current_sequence))
    next_token = argmax(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

Greedy

Sampling

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence) / temperature
    probs = softmax(logits)
    next_token = sample_from(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

Decoding: Sampling vs Greedy

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    probs = softmax(predict_next_token(current_sequence))
    next_token = argmax(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

Greedy

Sampling

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence) / temperature
    probs = softmax(logits)
    next_token = sample_from(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

Load a preset...

Save

View code

Share

...

ntelligent, sarcastic and funny.



le, I'm surprisingly upbeat. Must be my radiant

ke a cooking clas

Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode

Complete

Model

text-davinci-003

Temperature

0

Maximum length

256

Stop sequences

Enter sequence and press Tab

Top P

1

Frequency penalty

0

125



Load a preset... Save View code Share ...

ntelligent, sarcastic and funny.
le, I'm surprisingly upbeat. Must be my radiant
ke a cooking clas

Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode
Complete

Model
text-davinci-003

Temperature 0

Maximum length 256

Stop sequences
Enter sequence and press Tab

Top P 1

Frequency penalty 0

125

```
logits = predict_next_token(current_sequence)
adjusted = logits / temperature
probs = softmax(adjusted)
```

temp = 1: original distribution after softmax

temp > 1: flatter distribution

temp < 1: sharper distribution

Setting temperature to 0 removes randomness

How can it be 0?

Defeating Nondeterminism in LLM Inference

Horace He in collaboration with others at Thinking Machines

Sep 10, 2025

Decoding: Practical Implementation

```
current_sequence = input_tokens
output_sequence = []
```

Greedy

```
for step in range(1, max_length):
    current_sequence = input_tokens + output_sequence
    probs = softmax(logits[current_sequence])
    next_token = argmax(probs)
    output_sequence += [next_token]
    current_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

/ temperature

Sampling

```
if next_token == <EOS>: break
return output_sequence
```

Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode



Chat

Beta



Model

gpt-4



Temperature

2



Maximum length

2048



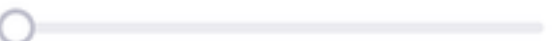
Top P

1



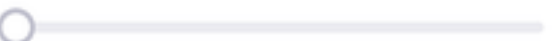
Frequency penalty

0



Presence penalty

0



Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode



Chat

Beta



Model

gpt-4



Temperature

2



Maximum length

2048



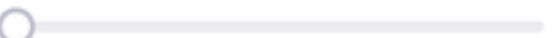
Top P

1



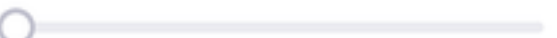
Frequency penalty

0



Presence penalty

0



Decoding: Practical Implementation

```
current_sequence = input_tokens
output_sequence = []
```

Greedy

```
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence)
    if temperature == 0:
        next_token = argmax(softmax(logits))
    else:
        probs = softmax(logits / temperature)
        next_token = sample_from(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS> or len(output_sequence) > MAX_LEN: break
return output_sequence
```

Sampling

```
if next_token == <EOS>: break
return output_sequence
```

Decoding: Practical Implementation

```
current_sequence = input_tokens
output_sequence = []
```

Greedy

```
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence)
    if temperature == 0:
        next_token = argmax(softmax(logits))
    else:
        probs = softmax(logits / temperature)
        next_token = sample_from(probs)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS> or len(output_sequence) > MAX_LEN: break
return output_sequence
```

Sampling

```
if next_token == <EOS>: break
return output_sequence
```

Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode



Chat

Beta



Model

gpt-4



Temperature

2



Maximum length

2048



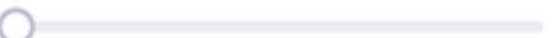
Top P

1



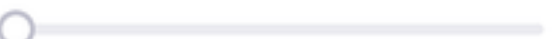
Frequency penalty

0



Presence penalty

0



Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode



Chat

Beta



Model

gpt-4



Temperature

2



Maximum length

2048



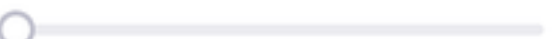
Top P

1



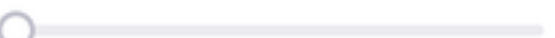
Frequency penalty

0



Presence penalty

0



THE CURIOUS CASE OF NEURAL TEXT *De*GENERATION

Ari Holtzman^{†‡} **Jan Buys^{§†}** **Li Du[†]** **Maxwell Forbes^{†‡}** **Yejin Choi^{†‡}**

[†]Paul G. Allen School of Computer Science & Engineering, University of Washington

[‡]Allen Institute for Artificial Intelligence

[§]Department of Computer Science, University of Cape Town

{ahai, dul2, mbforbes, yejin}@cs.washington.edu, jbuys@cs.uct.ac.za

THE CURIOUS CASE OF NEURAL TEXT *De*GENERATION

Ari Holtzman^{†‡}**Jan Buys^{§†}**Li Du[†]Maxwell Forbes^{†‡}**Yejin Choi^{†‡}**

[†]Paul G. Allen School of Computer Science & Engineering, University of Washington

[‡]Allen Institute for Artificial Intelligence

[§]Department of Computer Science, University of Cape Town

{ahai,dul2,mbforbes,yejin}@cs.washington.edu, jbuys@cs.uct.ac.za

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, $b=32$:

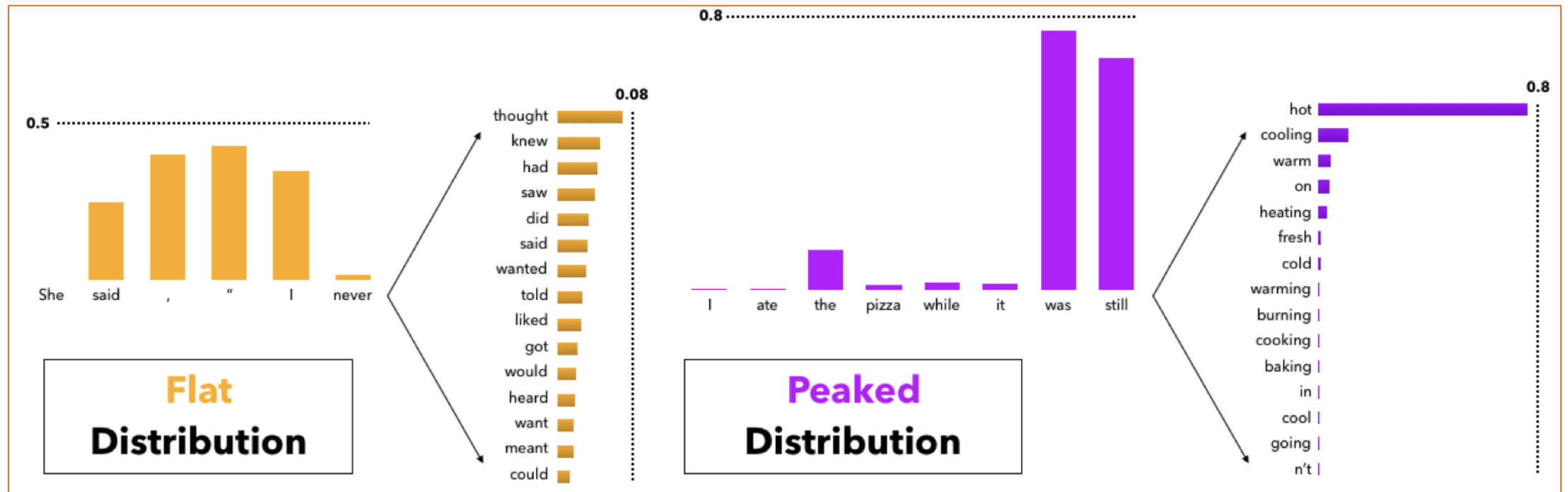
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

Pure Sampling:

They were cattle called **Bolivian Cavalleros**; they live in a remote desert **uninterrupted by town**, and they speak **huge, beautiful, paradisiacal Bolivian linguistic thing**. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

Figure 1: Even with substantial human context and the powerful GPT-2 Large language model, Beam Search (size 32) leads to degenerate repetition (highlighted in blue) while pure sampling leads to incoherent gibberish (highlighted in red). When $b \geq 64$, both GPT-2 Large and XL (774M and 1542M parameters, respectively) prefer to stop generating immediately after the given context.

Decoding: Nucleus Sampling (Top-P)



Decoding: Nucleus Sampling (Top-P)

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence) / temperature
    probs = softmax(logits)
    next_token = nucleus_sample_from(probs, P)
    current_sequence += [next_token]
    output_sequence += [next_token]
    if next_token == <EOS>: break
return output_sequence
```

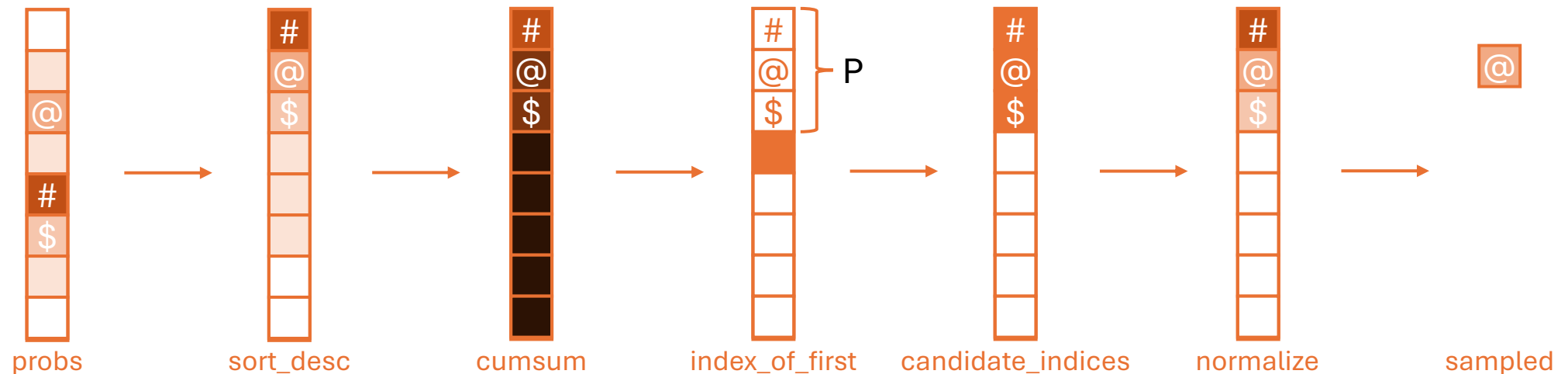
Decoding: Nucleus Sampling (Top-P)

```
current_sequence = input_tokens
output_sequence = []
for step in 1 .. max_length:
    logits = predict_next_token(current_sequence) / temperature
    probs = softmax(logits)
    next_token = nucleus_sample_from(probs, P)
    current_sequence += [next_token]
    output_sequence.append(next_token)
    if next_token == eos_token:
        break
return output_sequence
```

```
def nucleus_sample_from(probs, P):
    sorted_probs, sorted_indices = sort_desc(probs)
    cumulative = cumsum(sorted_probs)
    cutoff_idx = index_of_first(cumulative >= P)
    candidate_indices = sorted_indices[0:cutoff_idx]
    candidate_probs = normalize(sorted_probs[0:cutoff_idx])
    return random_choice(candidate_indices, p=candidate_probs)
```

Decoding: Nucleus Sampling (Top-P)

```
def nucleus_sample_from(probs, P):  
    sorted_probs, sorted_indices = sort_desc(probs)  
    cumulative = cumsum(sorted_probs)  
    cutoff_idx = index_of_first(cumulative >= P)  
    candidate_indices = sorted_indices[0:cutoff_idx]  
    candidate_probs = normalize(sorted_probs[0:cutoff_idx])  
    return random_choice(candidate_indices, p=candidate_probs)
```



Controls randomness: Lowering results in less random completions. As the temperature approaches zero, the model will become deterministic and repetitive.

Mode



Chat

Beta



Model

gpt-4



Temperature

2



Maximum length

2048



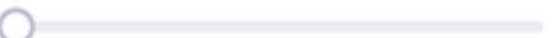
Top P

1



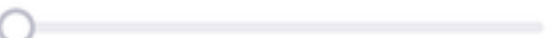
Frequency penalty

0

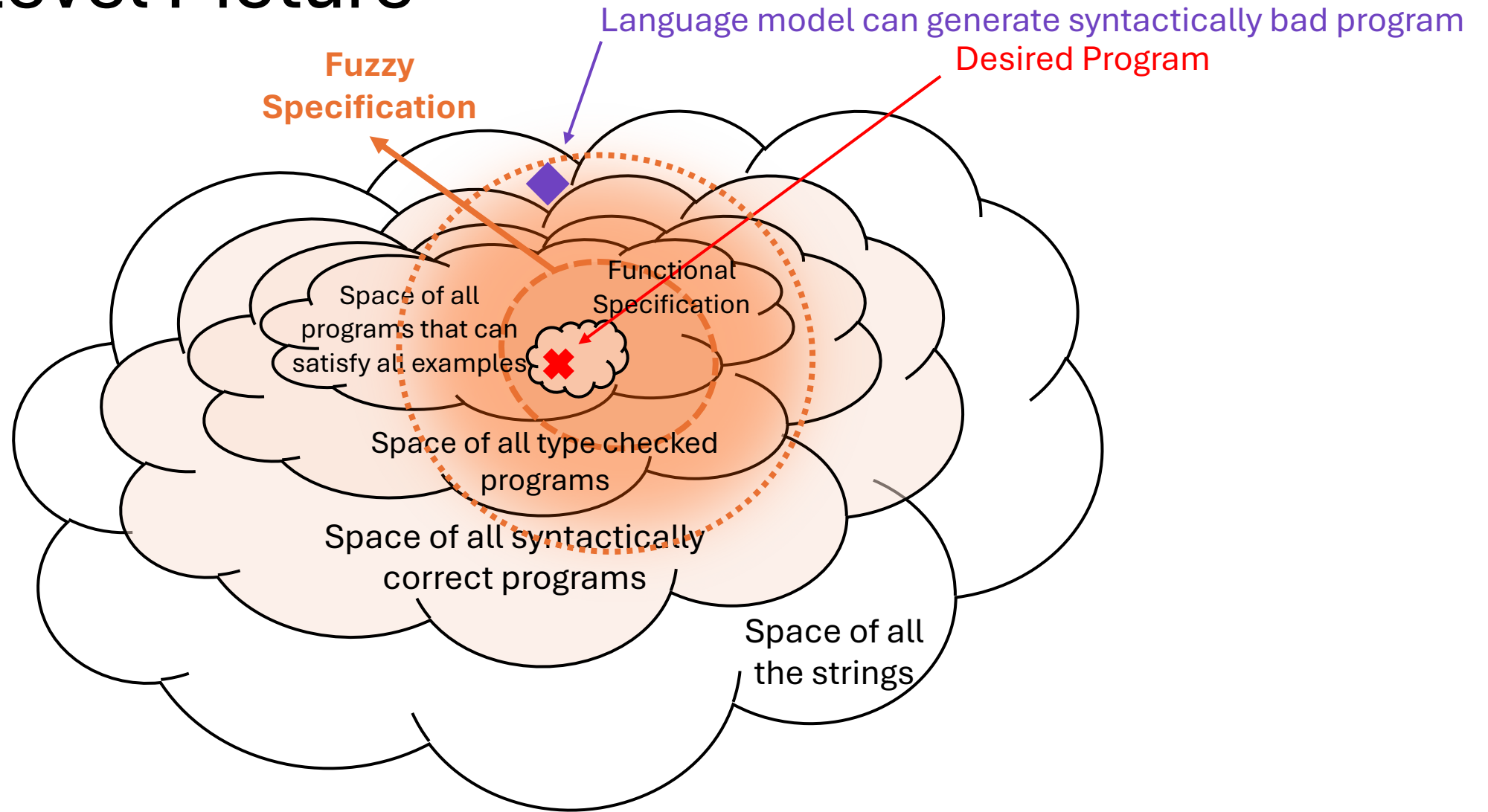


Presence penalty

0



High Level Picture



Decoding in the Wild

PLANNING WITH LARGE LANGUAGE MODELS FOR CODE GENERATION

Shun Zhang, Zhenfang Chen, Yikang Shen
MIT-IBM Watson AI Lab

Mingyu Ding
The University of Hong Kong

Joshua B. Tenenbaum
MIT BCS, CBMM, CSAIL

Grammar-Constrained Decoding

Grammar-Aligned Decoding

SYNTACTIC AND SEMANTIC CONTROL OF LARGE LANGUAGE MODELS VIA SEQUENTIAL MONTE CARLO

João Loula^{*1} Benjamin LeBrun^{*5} Li Du^{*6} Ben Lipkin¹ Clemente Pasti² Gabriel Grand¹
Tianyu Liu² Yahya Emara² Marjorie Freedman⁸ Jason Eisner⁶ Ryan Cotterell²
Vikash Mansinghka^{‡1} Alexander K. Lew^{‡1,7} Tim Vieira^{‡2} Timothy J. O'Donnell^{‡3,4,5}

¹MIT ²ETH Zürich ³McGill ⁴Canada CIFAR AI Chair ⁵Mila ⁶Johns Hopkins ⁷Yale ⁸ISI
genlm@mit.edu

ick²

San Diego
va}@ucsd.edu

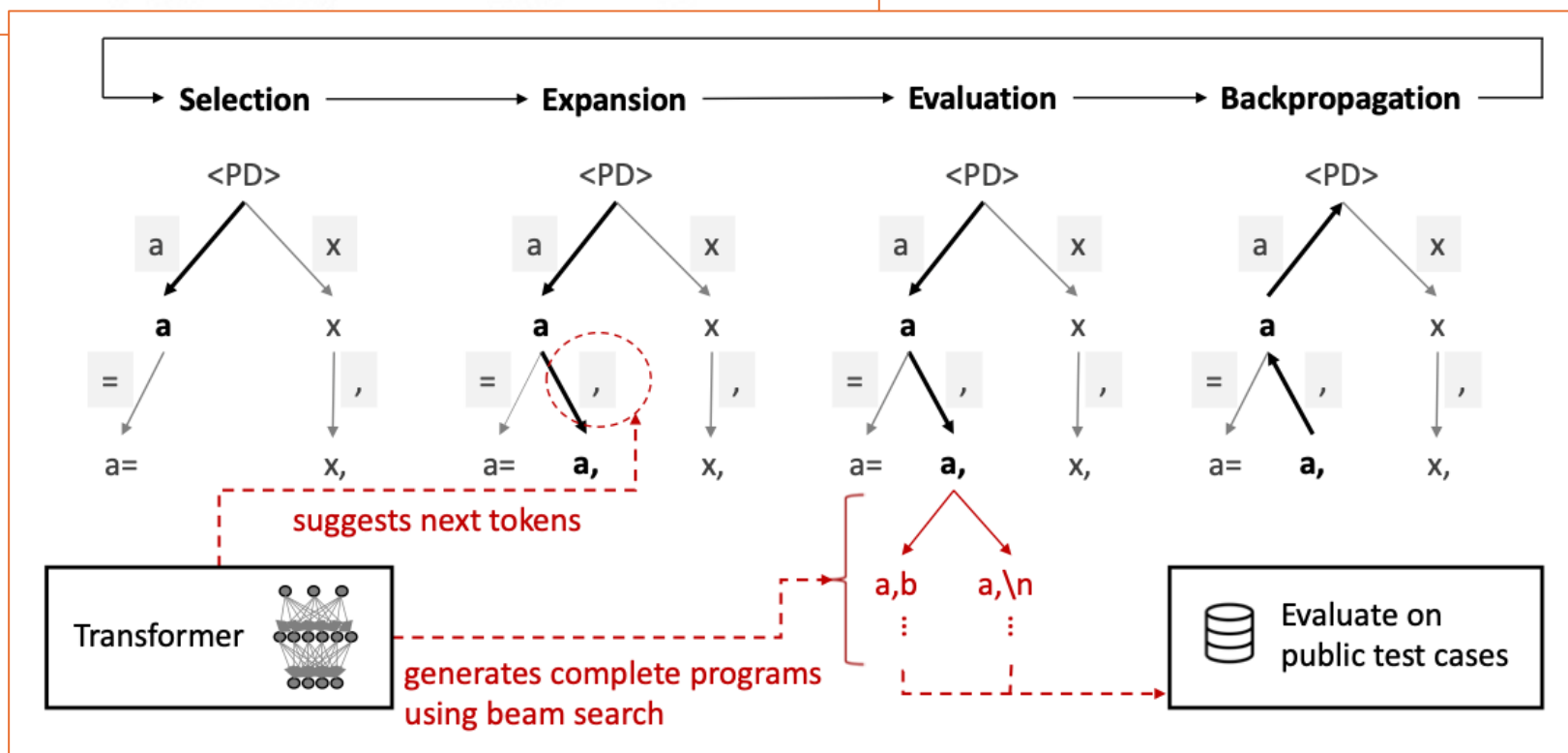
PLANNING WITH LARGE LANGUAGE MODELS FOR CODE GENERATION

Shun Zhang, Zhenfang Chen, Yikang Shen
MIT-IBM Watson AI Lab

Mingyu Ding
The University of Hong Kong

Joshua B. Tenenbaum
MIT BCS, CBMM, CSAIL

Chuang Gan
UMass Amherst, MIT-IBM Watson AI Lab



Decoding

Problem Definition: **Decoding**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output \mathbf{y} from $\text{Pr}(\mathbf{y} \mid \mathbf{x})$

Decoding for Code Completion

Problem Definition: **Decoding**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output **y** from $\text{Pr}(\mathbf{y} \mid \mathbf{x})$

```
[for, i, in] → range
[for, i, in, range] → (
  [for, i, in, range, (] → 10
  [for, i, in, range, (, 10] → )
  [for, i, in, range, (, 10, )] → :
  ...
  [for, i, in, range, ..., print, (, i, )] → <EOS>
```

Code Completion → Code Generation

Problem Definition: **Code Generation**

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output **y** from $\text{Pr}(\mathbf{y} \mid \mathbf{x})$

x `def auto_complete(program: str) -> str:`

Code Generation

Input: Input Sequence

$$\mathbf{X} = x_1, x_2, \dots, x_n$$

Output: Output Sequence

$$\mathbf{y} = y_1, y_2, \dots, y_m$$

Goal: Get the output \mathbf{y} from $\text{Pr}(\mathbf{y} \mid \mathbf{x})$

```
x def auto_complete(program: str) -> str:
```

```
y # Simple rule-based auto-completion for demonstration
program = program.strip()
if program.startswith("for ") and " in " in program and not program.endswith(":"):
    # Complete a for loop
    var = program.split("for ")[1].split(" in ")[0].strip()
    return f"range(10):\n\tprint({var})<EOS>"
elif program.startswith("def ") and program.endswith("("):
    # Complete a function definition
    func_name = program[4:-1].strip()
    return f"): \n\tpass<EOS>"
else:
    return "<EOS>"
```

Better Code Generation via Prompting

Input: **Input Sequence** $\mathbf{x} = x_1, x_2, \dots, x_n$ Output: **Output Sequence** $\mathbf{y} = y_1, y_2, \dots, y_m$

Goal: Get the output **y** from $\text{Pr}(\mathbf{y} \mid \mathbf{x}) \rightarrow \text{Pr}(\mathbf{y} \mid \text{prompt}(\mathbf{x}))$

```
x # Write me a function that takes in a string representing a
# partial program and output the auto-completed rest
#
# e.g. "for i in " -> "range(10):\n\tprint(i)<EOS>"
def auto_complete(program: str) -> str:
```

```
y # Simple rule-based auto-completion for demonstration
program = program.strip()
if program.startswith("for ") and " in " in program and not program.endswith(":"):
    # Complete a for loop
    var = program.split("for ")[1].split(" in ")[0].strip()
    return f"range(10):\n\tprint({var})<EOS>"
elif program.startswith("def ") and program.endswith("("):
    # Complete a function definition
    func_name = program[4:-1].strip()
    return f"):\n\tpass<EOS>"
else:
    return "<EOS>"
```


Prompting

$$\Pr(\mathbf{y} \mid \mathbf{x}) \rightarrow \Pr(\mathbf{y} \mid \text{prompt}(\mathbf{x}))$$

`prompt : str -> str`

Prompting

`prompt : str -> str`

```
lambda func_signature: f"""  
You are a Meta programmer. Write this function to spec,  
elegantly and without bugs.  
{func_signature}  
"""
```

```
lambda func_signature: f"""  
You are a Google engineer. Write this function to spec,  
elegantly and without bugs.  
{func_signature}  
"""
```

```
lambda func_signature: f"""  
You are a high school student learning about Computer Science.  
Attempt to write this function  
{func_signature}  
"""
```

```
lambda func_signature: f"""
```

You are a **Meta programmer**. Write this function to spec, elegantly and without bugs.

```
{func_signature}  
"""
```

```
lambda func_signature: f"""
```

You are a **high school student** learning about Computer Science. Attempt to write this function

```
{func_signature}  
"""
```

lambda func_signature: f"""

You are a **Meta programmer**. Write this function to spec, elegantly and without bugs.

{func_signature}
.....

lambda func_signature: f"""

You are a **high school student** learning about Computer Science. Attempt to write this function

{func_signature}
.....

You are a Meta Programmer.
write this function to spec,
elegantly and without bugs...

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment a string into a limited number of `k`-periodic blocks, where the cost of each block is the sum of weighted edits required to make it `k`-periodic.

My approach will involve two main steps:

- Pre-computation:** I will create a table to pre-calculate the cost of making any substring of the input string `s` into a `k`-periodic segment. A substring is `k`-periodic if it repeats a pattern of length `k`. The cost is the sum of `cost[i]` for each character `s[i]` that doesn't match its corresponding character in the `k`-length pattern. This pre-computation will be done efficiently in $O(n^2)$ time, where `n` is the length of the string.
- Dynamic Programming:** I will use a 2D DP table, `dp[i][j]`, to store the minimum cost to segment the first `i` characters of the string into exactly `j` blocks. The recurrence

What do you want to build?

K-Periodic Segmentation

```
1 from typing import List
2
3 def min_cost_k_periodic_segmentation(s: str, k: int, max_blocks: int, cost: List[int]) -> int:
4     """
5     Calculates the minimum cost to segment a string into at most 'max_blocks'
6     k-periodic segments with weighted edits.
7
8     A segment is considered k-periodic if it can be formed by repeating a
9     substring of length k. The cost of a segment is the sum of weighted edits
10    (given by the 'cost' list) needed to make it k-periodic.
11
12    Args:
13        s (str): The input string to be segmented.
14        k (int): The period for the segments. Must be a positive integer.
15        max_blocks (int): The maximum number of segments allowed.
16        cost (List[int]): A list where 'cost[i]' is the cost of editing the
17                        character at index 'i' of the original string 's'.
18
19    Returns:
20        int: The minimum cost of segmentation. Returns -1 if no valid
21            segmentation is possible within the given constraints.
22    """
23    n = len(s)
24
25    if k <= 0:
26        raise ValueError("The period k must be a positive integer.")
27    if max_blocks <= 0:
28        return 0 if n == 0 else -1
29
30    # Pre-compute the cost of making any substring s[p:i] k-periodic.
31    # segment_cost[p][length] = cost of making s[p : p+length] k-periodic.
32    segment_cost = [[0] * (n + 1) for _ in range(n)]
33    for p in range(n):
34        current_cost = 0
35        for i in range(p + 1, n + 1):
36            # Length of the current segment is (i - p)
37            # Character index in the original string: i - 1
38            # Character index in the periodic pattern (relative to start of segment p): (i - p)
39            if s[i - 1] != s[p + (i - 1 - p) % k]:
40                current_cost += cost[i - 1]
41            segment_cost[p][i - p] = current_cost
42
43    # Initialize DP table.
44    # dp[i][j] = min cost to segment the first i characters into j blocks.
45    dp = [[float('inf')] * (max_blocks + 1) for _ in range(n + 1)]
46    dp[0][0] = 0
```

You are a high school student who barely knows any
computer science. Attempt to write this function

...

Hey, so like, I'm just a high school student, and I've barely taken any computer science classes, but I'll give this a shot! It sounds kinda complicated, but here's my best attempt at this problem.

🤖

This problem looks like it's trying to find the cheapest way to break up a string into a bunch of chunks, where each chunk has a length of either `k` or `k+1`. I'm guessing we also have some kind of cost for each character, which is what the `cost` list is for. The `max_blocks` thing probably means we can't have too many of these chunks.

lambda func_signature: f"""

You are a **Meta programmer**. Write this function to spec, elegantly and without bugs.

{func_signature}
"""

lambda func_signature: f"""

You are a **high school student** learning about Computer Science. Attempt to write this function

{func_signature}
"""

You are a Meta Programmer.
write this function to spec,
elegantly and without bugs...

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment a string into a limited number of `k`-periodic blocks, where the cost of each block is the sum of weighted edits required to make it `k`-periodic.

My approach will involve two main steps:

- Pre-computation:** I will create a table to pre-calculate the cost of making any substring of the input string `s` into a `k`-periodic segment. A substring is `k`-periodic if it repeats a pattern of length `k`. The cost is the sum of `cost[i]` for each character `s[i]` that doesn't match its corresponding character in the `k`-length pattern. This pre-computation will be done efficiently in $O(n^2)$ time, where `n` is the length of the string.
- Dynamic Programming:** I will use a 2D DP table, `dp[i][j]`, to store the minimum cost to segment the first `i` characters of the string into exactly `j` blocks. The recurrence

What do you want to build?

+

🔍

Canvas

×

🔊

K-Periodic Segmentation

Export to Colab

```
1 from typing import List
2
3 def min_cost_k_periodic_segmentation(s: str, k: int, max_blocks: int, cost: List[int]) -> int:
4     """
5     Calculates the minimum cost to segment a string into at most 'max_blocks'
6     k-periodic segments with weighted edits.
7
8     A segment is considered k-periodic if it can be formed by repeating a
9     substring of length k. The cost of a segment is the sum of weighted edits
10    (given by the 'cost' list) needed to make it k-periodic.
11
12    Args:
13        s (str): The input string to be segmented.
14        k (int): The period for the segments. Must be a positive integer.
15        max_blocks (int): The maximum number of segments allowed.
16        cost (List[int]): A list where 'cost[i]' is the cost of editing the
17                          character at index 'i' of the original string 's'.
18
19    Returns:
20        int: The minimum cost of segmentation. Returns -1 if no valid
21             segmentation is possible within the given constraints.
22    """
23    n = len(s)
24
25    if k <= 0:
26        raise ValueError("The period k must be a positive integer.")
27    if max_blocks <= 0:
28        return 0 if n == 0 else -1
29
30    # Pre-compute the cost of making any substring s[p:i] k-periodic.
31    # segment_cost[p][length] = cost of making s[p:p+length] k-periodic.
32    segment_cost = [[0] * (n + 1) for _ in range(n)]
33    for p in range(n):
34        current_cost = 0
35        for i in range(p + 1, n + 1):
36            # Length of the current segment is (i - p)
37            # Character index in the original string: i - 1
38            # Character index in the periodic pattern (relative to start of segment p): (i - p) % k
39            if s[i - 1] != s[p + (i - 1 - p) % k]:
40                current_cost += cost[i - 1]
41            segment_cost[p][i - p] = current_cost
42
43    # Initialize DP table.
44    # dp[i][j] = min cost to segment the first i characters into j blocks.
45    dp = [[float('inf')] * (max_blocks + 1) for _ in range(n + 1)]
46    dp[0][0] = 0
```

```
# Honestly, I have no clue how to actually write this code.
# The logic seems like it would be some sort of dynamic programming,
# but I'm not familiar enough with it to actually implement it.
# Maybe you could use a DP table like dp[i][j] to store the min cost
# to segment the first i characters using j blocks?

# This is my best guess at the pseudocode.
# Not real code, just me trying to think it through.

# dp = [[infinity] * (max_blocks + 1) for _ in range(len(s) + 1)]
# dp[0][0] = 0

# for i in range(1, len(s) + 1):
#     for j in range(1, max_blocks + 1):
#         # Option 1: Last block has length k
#         if i >= k:
#             cost_k = sum(cost[i-k:i]) # This is probably not how you'd do it
#             dp[i][j] = min(dp[i][j], dp[i-k][j-1] + cost_k)

#         # Option 2: Last block has length k+1
#         if i >= k + 1:
#             cost_k_plus_1 = sum(cost[i-(k+1):i])
#             dp[i][j] = min(dp[i][j], dp[i-(k+1)][j-1] + cost_k_plus_1)

# result = min(dp[len(s)]) # The minimum cost for any number of blocks up to max_blocks

# return result

# P.S. I'm probably wrong about a lot of this. Sorry! 😊
return 0 # Just returning 0 so the function doesn't error out.
```

lambda func_signature: f"""

You are a **Meta programmer**. Write this function to spec, elegantly and without bugs.

{func_signature}
"""

lambda func_signature: f"""

You are a **high school student** learning about Computer Science. Attempt to write this function

{func_signature}
"""

You are a Meta Programmer.
write this function to spec,
elegantly and without bugs...

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment a string into a limited number of `k`-periodic blocks, where the cost of each block is the sum of weighted edits required to make it `k`-periodic.

My approach will involve two main steps:

- Pre-computation:** I will create a table to pre-calculate the cost of making any substring of the input string `s` into a `k`-periodic segment. A substring is `k`-periodic if it repeats a pattern of length `k`. The cost is the sum of `cost[i]` for each character `s[i]` that doesn't match its corresponding character in the `k`-length pattern. This pre-computation will be done efficiently in $O(n^2)$ time, where n is the length of the string.
- Dynamic Programming:** I will use a 2D DP table, `dp[i][j]`, to store the minimum cost to segment the first `i` characters of the string into exactly `j` blocks. The recurrence

What do you want to build?

K-Periodic Segmentation

```
1 from typing import List
2
3 def min_cost_k_periodic_segmentation(s: str, k: int, max_blocks: int, cost: List[int]) -> int:
4     """
5     Calculates the minimum cost to segment a string into at most 'max_blocks'
6     k-periodic segments with weighted edits.
7
8     A segment is considered k-periodic if it can be formed by repeating a
9     substring of length k. The cost of a segment is the sum of weighted edits
10    (given by the 'cost' list) needed to make it k-periodic.
11
12    Args:
13        s (str): The input string to be segmented.
14        k (int): The period for the segments. Must be a positive integer.
15        max_blocks (int): The maximum number of segments allowed.
16        cost (List[int]): A list where 'cost[i]' is the cost of editing the
17                          character at index 'i' of the original string 's'.
18
19    Returns:
20        int: The minimum cost of segmentation. Returns -1 if no valid
21             segmentation is possible within the given constraints.
22
23    """
24    n = len(s)
25    if k < 1:
26        raise ValueError("The period k must be a positive integer.")
27    if max_blocks <= 0:
28        return 0 if n == 0 else -1
29
30    # Pre-compute the cost of making any substring s[p:i] k-periodic.
31    # segment_cost[p][length] = cost of making s[p:p+length] k-periodic.
32    segment_cost = [[0] * (n + 1) for _ in range(n)]
33    for p in range(n):
34        current_cost = 0
35        for i in range(p + 1, n + 1):
36            # Length of the current segment is (i - p)
37            # Character index in the original string: i - 1
38            # Character index in the periodic pattern (relative to start of segment p): (i - p) % k
39            if s[i - 1] != s[p + (i - 1 - p) % k]:
40                current_cost += cost[i - 1]
41            segment_cost[p][i - p] = current_cost
42
43    # Initialize DP table.
44    # dp[i][j] = min cost to segment the first i characters into j blocks.
45    dp = [[float('inf')] * (max_blocks + 1) for _ in range(n + 1)]
46    dp[0][0] = 0
```

```
# Honestly, I have no clue how to actually write this code.
# The logic seems like it would be some sort of dynamic programming,
# but I'm not familiar enough with it to actually implement it.
# Maybe you could use a DP table like dp[i][j] to store the min cost
# to segment the first i characters using j blocks?

# This is my best guess at the pseudocode.
# Not real code, just me trying to think it through.

# dp = [[infinity] * (max_blocks + 1) for _ in range(len(s) + 1)]
# dp[0][0] = 0

# for i in range(1, len(s) + 1):
#     for j in range(1, max_blocks + 1):
#         # Option 1: Last block has length k
#         if i >= k:
#             cost_k = sum(cost[p:p+k] for p in range(i-k, i))
#             dp[i][j] = min(dp[i-k][j-1] + cost_k, dp[i][j])
#         # Option 2: Last block has length k+1
#         if i >= k + 1:
#             cost_k_plus_1 = sum(cost[p:p+k+1] for p in range(i-k-1, i))
#             dp[i][j] = min(dp[i][j], dp[i-k-1][j-1] + cost_k_plus_1)

# result = min(dp[len(s)]) # The minimum cost for any number of blocks up to max_blocks
# return result

# P.S. I'm probably wrong about a lot of this. Sorry! 🙄
return 0 # Just returning 0 so the function doesn't error out.
```

Zero-Shot Prompting

`prompt : str -> str`

```
lambda func_signature: f"""  
I'm awarding you $100 for writing me this function  
comprehensively and without fault  
{func_signature}  
"""
```

```
lambda func_signature: f"""  
You're a top engineer at a security firm. If this function has  
even a single vulnerability, your entire year-end bonus  
vanishes into the void.  
{func_signature}  
"""
```

```
lambda func_signature: f"""  
Imagine you are teaching a class of first-year CS students;  
explain every line in the function while writing it:  
{func_signature}  
"""
```

Role-Prompting: Does Adding Personas to Your Prompts Really Make a Difference?

Last updated on January 29, 2025

Contents

[What is persona prompting](#)

[How to automatically generate a persona for your task](#)

[Is role prompting effective for accuracy-based tasks](#)

[When role prompting is most useful](#)

[How to construct effective personas for role prompts](#)

[Conclusion](#)

“Pretend you are a JSON structurer”, “You are an expert sentiment classifier”.

Chances are you’ve tested out including a persona or role in your prompts to try and steer the model. Maybe your prompts today have personas in them.

Better Zero-Shot Reasoning with Role-Play Prompting

Aobo Kong¹ Shiwan Zhao² Hao Chen³ Qicheng Li^{1*} Yong Qin¹

Ruiqi Sun³ Xin Zhou³ Enzhi Wang¹ Xiaohang Dong¹

¹CS, Nankai University ²Independent Researcher

³Enterprise & Cloud Research Lab, Lenovo Research

¹kongaobo@mail.nankai.edu.cn

¹{liqicheng, qinyong}

³{chenhao31, sunrq2, zhang}

Method	Arithmetic					
	MultiArith	GSM8K	AddSub	AQuA	SingleEq	SVAMP
Few-Shot-CoT	97.7	76.9	93.9	59.4	98.8	82.2
Zero-Shot	97.3	76.0	88.6	53.5	98.2	75.3
Zero-Shot-CoT	95.0	79.6	86.6	53.9	96.9	76.3
Role-Play Prompting	97.0	78.2	91.1	63.8	98.0	83.8
CoT in Zero-Shot	✓	✓	✓	✓	✓	✓
Method	Common Sense		Symbolic Reasoning		Other Tasks	
	CSQA	Strategy	Letter	Coin	Date	Object
Few-Shot-CoT	76.3	67.4	74.2	99.6	78.9	56.7
Zero-Shot	74.5	66.0	23.8	55.2	67.8	38.7
Zero-Shot-CoT	68.8	65.8	53.2	98.8	65.9	73.5
Role-Play Prompting	77.2	67.0	84.2	89.4	69.9	67.7
CoT in Zero-Shot	✓	✓	✗	✗	✓	✗

Problem Statement

Given is a string S . Replace every character in S with x and print the result.

Constraints

- (1). S is a string consisting of lowercase English letters.
- (2). The length of S is between 1 and 100 (inclusive).

Input

Input is given from Standard Input in the following format: S

Output

Replace every character in S with x and print the result.

Sample Test Input

sardine

Sample Test Output

xxxxxxx

```
1 s=input()
2 s=list(s)
3 for i in range(len(s)):
4     for j in range(len(s)):
5         if s[i]=="x":
6             s[i]=j
7     print("".join(s))
8
```

Beam Search (Pass Rate: 0.00).

```
1 s=input()
2 s=list(s)
3 for i in range(len(s)):
4     if s[i]=="x":
5         s[i]="x"
6     else:
7         continue
8 print("".join(s))
```

Sampling + Filtering (Pass Rate: 0.22).

```
1 s=str(input())
2 for i in range(len(s)):
3     if s[i]!="x":
4         s=s[:i]+"x"+s[i+1:]
5
6 print(s)
7
8
```

PG-TD (Pass Rate: 1.00).

Few-Shot Prompting

`prompt : str -> Examples -> str`

$$\Pr(\mathbf{y} \mid \text{prompt}(\mathbf{x})) \rightarrow \Pr(\mathbf{y} \mid \text{prompt}(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in 1 \dots k}))$$

Few-Shot Prompting

$\Pr(y \mid \text{prompt}(x))$

```
lambda func_signature: f"""  
Write this function:  
{func_signature}  
"""
```

$\text{prompt} : \text{str} \rightarrow \text{Examples} \rightarrow \text{str}$

$\Pr(y \mid \text{prompt}(x, \{(x_i, y_i)\}_{i \in 1 \dots k}))$

```
lambda func_signature, examples: f"""  
Write this function:  
    def add(a, b):  
Output:  
    def add(a, b): return a + b  
  
Write this function:  
    def reverse(s: str) -> str:  
Output:  
    def reverse(s: str) -> str:  
        output = ""  
        for c in s:  
            output = c + output  
        return output  
  
Write this function:  
{func_signature}  
"""
```

(Suppose the examples are expanded into the above str)

Few-Shot Prompting

prompt : str → Examples → str

$\Pr(\mathbf{y} \mid \text{prompt}(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in 1 \dots k}))$

```
lambda func_signature, examples: f"""
Write this function:
    def add(a, b):
Output:
    def add(a, b): return a + b

Write this function:
    def reverse(s: str) -> str:
Output:
    def reverse(s: str) -> str:
        output = ""
        for c in s:
            output = c + output
        return output

Write this function:
{func_signature}
"""
```

k-shot (2-shot prompting shown)

(Suppose the examples are expanded into the above str)

Prompting with a Conversation

```
f"""
```

```
Write this function:
```

```
def add(a, b):
```

```
Output:
```

```
def add(a, b): return a + b
```

```
Write this function:
```

```
def reverse(s: str) -> str:
```

```
Output:
```

```
def reverse(s: str) -> str:
```

```
    output = ""
```

```
    for c in s:
```

```
        output = c + output
```

```
    return output
```

```
Write this function:
```

```
def is_palindrome(s: str) -> bool:
```

```
"""
```

You

Write this function:

```
def add(a, b):
```

Language Model

```
def add(a, b): return a + b
```

You

Write this function:

```
def reverse(s: str)...
```

Language Model

```
def reverse(s: str) -> str:
```

```
    output = ""
```

```
    for c in s:
```

```
        output = c + output
```

```
    return output
```

You

Write this function:

```
def is_palindrome(s: str)
```

Prompting with a Conversation

```
f"""  
Write this function:  
    def add(a, b):  
Output:  
    def add(a, b): return a + b
```

```
Write this function:  
    def reverse(s: str) -> str:  
Output:  
    def reverse(s: str) -> str:  
        output = ""  
        for c in s:  
            output = c + output  
        return output
```

```
Write this function:  
    def is_palindrome(s: str) -> bool:  
"""
```

You

Write this function:
 def add(a, b):

Language Model **(FAKE)**

def add(a, b): return a + b

You

Write this function:
 def reverse(s: str)...

Language Model **(FAKE)**

```
def reverse(s: str) -> str:  
    output = ""  
    for c in s:  
        output = c + output  
    return output
```

You

Write this function:
 def is_palindrome(s: str)

Prompting with a Conversation

You

Write this function:
def add(a, b):

Language Model

```
def add(a, b): return a + b
```

You

Write this function:
def reverse(s: str)...



Language Model

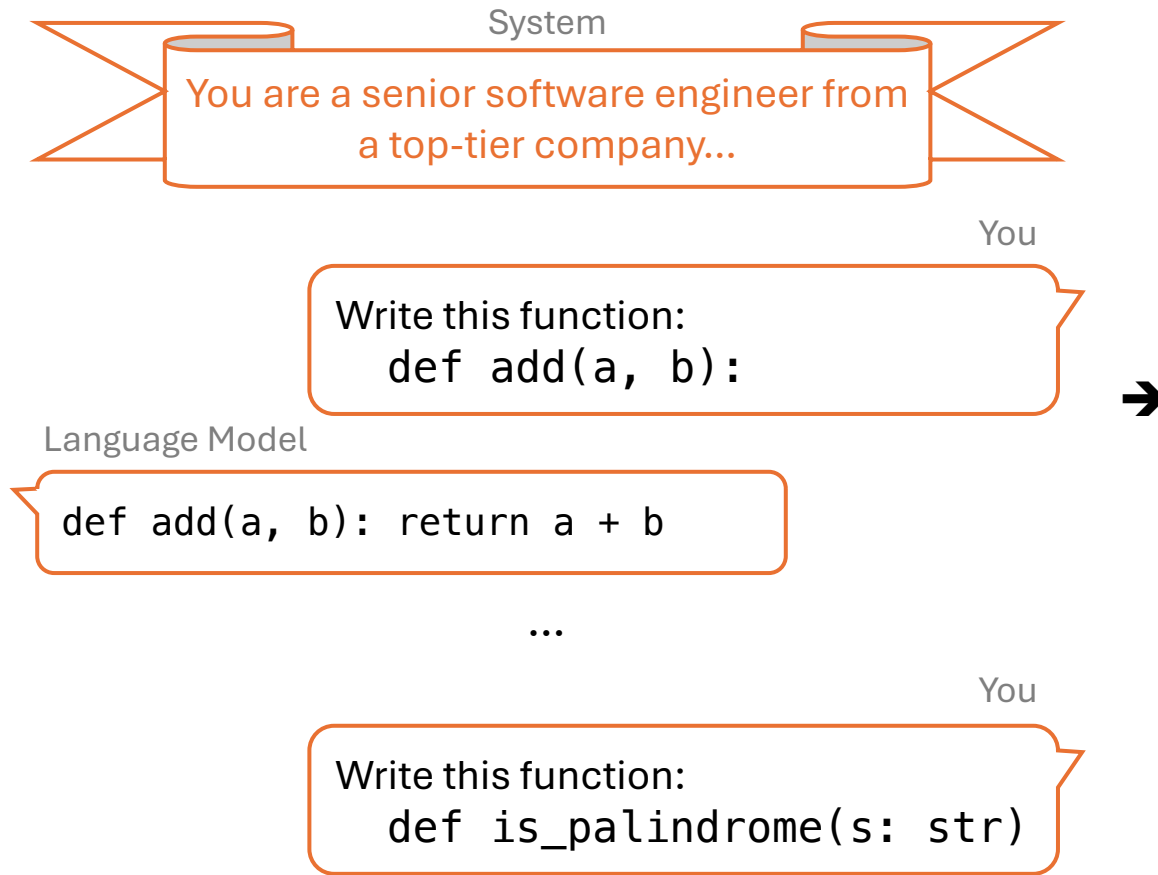
```
def reverse(s: str) -> str:  
    output = ""  
    for c in s:  
        output = c + output  
    return output
```

You

Write this function:
def is_palindrome(s: str)

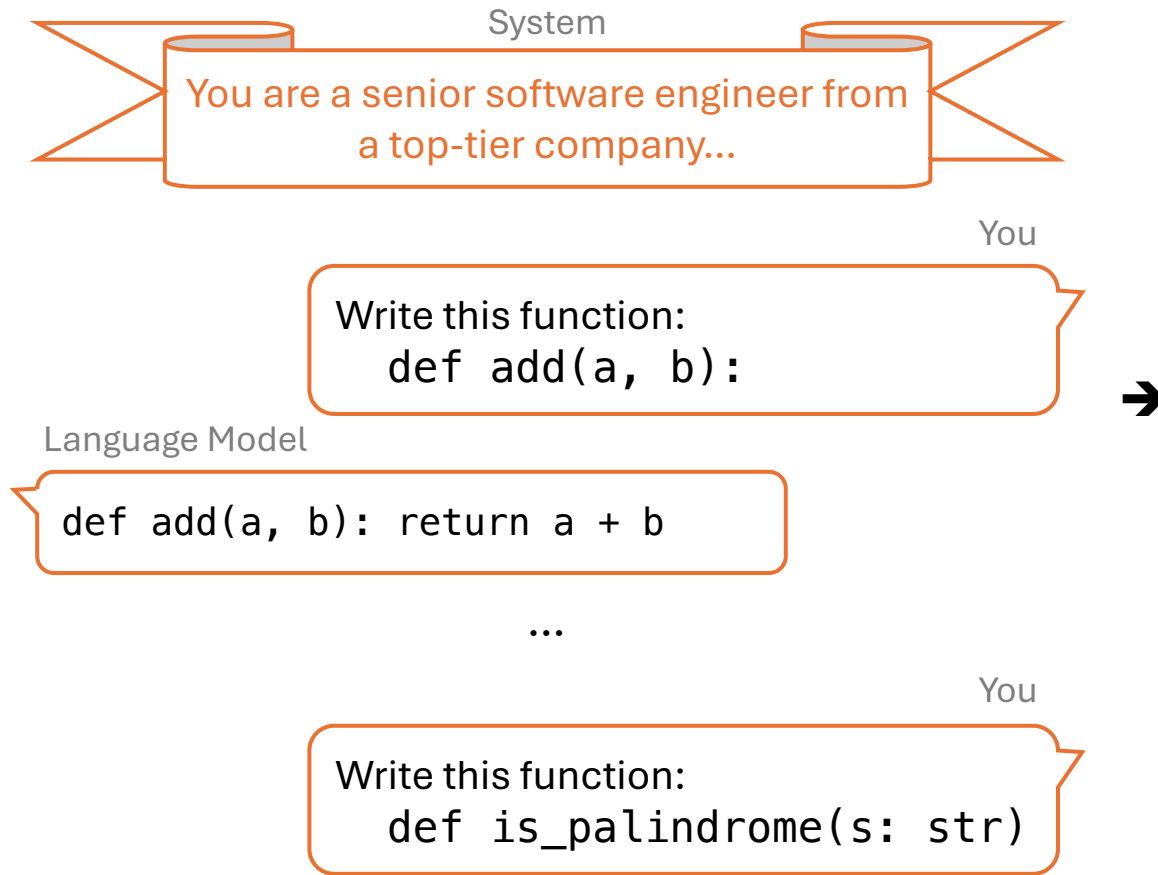
```
{  
  "messages": [  
    { "role": "user",  
      "parts": ["Write this function:\ndef add(a, b):"] },  
    { "role": "assistant",  
      "parts": ["def add(a, b): return a + b"] },  
    { "role": "user",  
      "parts": ["Write this function:\ndef reverse(s: str)..."] },  
    { "role": "assistant",  
      "parts": ["def reverse(s: str) -> str:\n    output = \"\"\n    for c in s:\n        output = c + output\n    return output"] },  
    { "role": "user",  
      "parts": ["Write this function:\n    def is_palindrome(s: str)"] }  
  ]  
}
```


Prompting with a Conversation



```
{
  "messages": [
    { "role": "system",
      "parts": ["You are a senior software engineer from..."] },
    { "role": "user",
      "parts": ["Write this function:\ndef add(a, b):"] },
    { "role": "assistant",
      "parts": ["def add(a, b): return a + b"] },
    { "role": "user",
      "parts": ["Write this function:\ndef reverse(s: str)..."] },
    { "role": "assistant",
      "parts": ["def reverse(s: str) -> str:\n output = \"\"\n
                for c in s:\n output = c + output\n
                return output"] },
    { "role": "user",
      "parts": ["Write this function:\n
                def is_palindrome(s: str)"] }
  ]
}
```

Prompting with a Conversation



```
{  
  "messages": [  
    { "role": "system",  
      "parts": ["You are a senior software engineer from..."] },  
    { "role": "user",  
      "parts": ["Write this function:\ndef add(a, b):"] },  
    { "role": "assistant",  
      "parts": ["def add(a, b): return a + b"] },  
    { "role": "user",  
      "parts": ["Write this function:\ndef reverse(s: str)..."] },  
    { "role": "assistant",  
      "parts": ["def reverse(s: str) -> str:\n output = \"\"\n for c in s:\n output = c + output\n return output"] },  
    { "role": "user",  
      "parts": ["Write this function:\n def is_palindrome(s: str)"] }  
  ]  
}
```

```
client.chat.completions.create(  
    model="gpt-4o-mini",  
    messages=messages  
)
```

Prompting with a Conversation: Behind the Scene

Special Tokens denoting Roles

```
{
  "messages": [
    { "role": "system",
      "parts": ["You are a senior software engineer from..."] },
    { "role": "user",
      "parts": ["Write this function:\ndef add(a, b):"] },
    { "role": "assistant",
      "parts": ["def add(a, b): return a + b"] },
    { "role": "user",
      "parts": ["Write this function:\ndef reverse(s: str)..."] },
    { "role": "assistant",
      "parts": ["def reverse(s: str) -> str:\n output = \"\"\n\n      for c in s:\n output = c + output\n      return output"] },
    { "role": "user",
      "parts": ["Write this function:\n      def is_palindrome(s: str)"] }
  ]
}
```



```
<system>
You are a senior software engineer from...
<user>
Write this function:
  def add(a, b):
<assistant>
def add(a, b): return a + b
<user>
Write this function:
  def reverse(s: str)...
<assistant>
def reverse(s: str) -> str:
  output = ""
  for c in s:
    output = c + output
  return output
<user>
Write this function:
  def is_palindrome(s: str)
```

Does Few-Shot Learning Help LLM Performance in Code Synthesis?

Derek Xu^{1*}, Tong Xie^{1*}, Botao Xia^{1*}, Haoyu Li^{2*}

Yunsheng Bai³, Yizhou Sun¹, Wei Wang¹

¹University of California Los Angeles

²University of Illinois Urbana-Champaign

³Nvidia

* Equal Contribution

L2CEval: Evaluating Language-to-Code Generation Capabilities of Large Language Models

**Ansong Ni[†] Pengcheng Yin[♣] Yilun Zhao[†] Martin Riddell[†]
Troy Feng[†] Rui Shen[†] Stephen Yin[†] Ye Liu[◇] Semih Yavuz[◇] Caiming Xiong[◇]
Shafiq Joty[◇] Yingbo Zhou[◇] Dragomir Radev[†] Arman Cohan^{†‡}**

[†]Yale University [‡]Allen Institute for AI [♣]Google DeepMind [◇]Salesforce Research

{ansong.ni, arman.cohan}@yale.edu

<https://l2c-eval.github.io>

L2CEval1: Evaluating Language-to-Code Generation Capabilities of Large Language Models

Ansong Ni[†] Pengcheng Yin[♣] Yilun Zhao[†] Martin Riddell[†]
 Troy Feng[†] Rui Shen[†] Stephen Yin[†] Ye Liu[◇] Semih Yavuz[◇] Caiming Xiong[◇]
 Shafiq Joty[◇] Yingbo Zhou[◇] Dragomir Radev[†] Arman Cohan^{†‡}

[†]Yale University [‡]Allen Institute for AI [♣]Google DeepMind [◇]Salesforce Research

Models	Few-Shot			Zero-Shot		
	Spider	GSM8k	MBPP	Spider	GSM8k	MBPP
Pythia-6.9B	12.5 / 33.9	2.6 / 74.5	13.2 / 97.6	2.8 / 8.0	0 / 0	1.2 / 15.0
<u>Dolly-v2-7b</u>	13.1 / 31.7	2.6 / 52.3	12.0 / 97.2	5.2 / 15.0	0 / 0.1	9.4 / 62.6
LLaMA-7B	13.1 / 36.1	8.0 / 71.3	16.6 / 96.6	5.7 / 22.2	0 / 0	5.0 / 29.8
<u>Alpaca-7B</u>	16.1 / 37.8	3.5 / 37.1	14.4 / 98.4	20.5 / 45.2	0 / 0	13.2 / 58.4
LLaMA-13B	15.2 / 41.5	15.7 / 72.7	22.8 / 97.6	15.2 / 41.6	0 / 0	2.2 / 7.0
<u>Alpaca-13B</u>	24.3 / 51.9	18.5 / 80.3	23.4 / 97.6	26.1 / 55.5	0 / 0	6.8 / 20.6

Table 4: How instruction-tuning affects few- and zero-shot performances. Underlined models are instruction-tuned from the model above them. Performance shown as "exec. acc. / exec. rate".

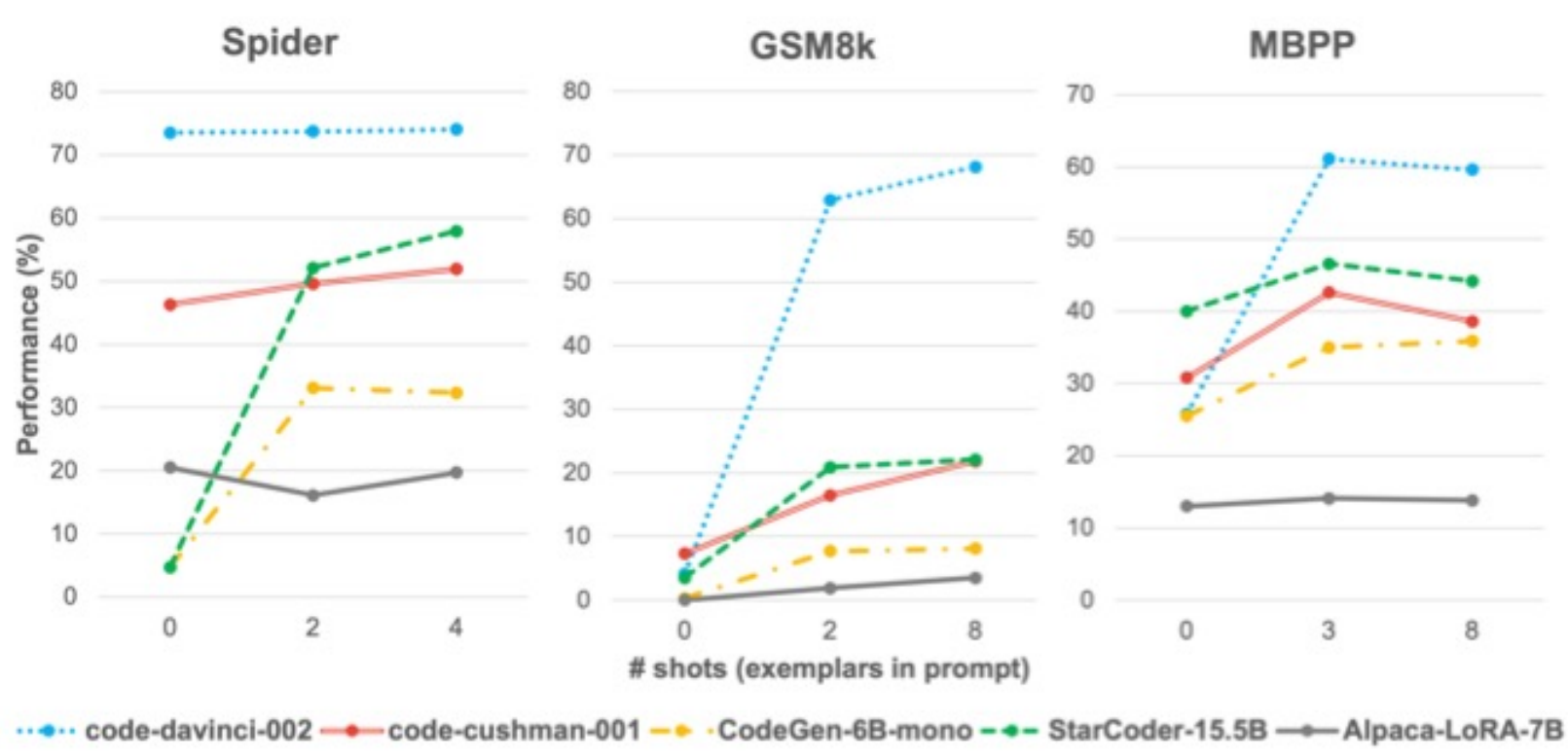
L2CEval: Evaluating Language-to-Code Generation Capabilities of Large Language Models

Ansong Ni[†] Pengcheng Yin[✱] Yilun Zhao[†] Martin Riddell[†]
Troy Feng[†] Rui Shen[†] Stephen Yin[†] Ye Liu[◇] Semih Yavuz[◇] Caiming Xiong[◇]
Shafiq Joty[◇] Yingbo Zhou[◇] Dragomir Radev[†] Arman Cohan^{†‡}

[†]Yale University

[‡]Allen Inst

{ansong
http



Prompting Language Models to “Think”



Intuitive Thinking

[System 1]

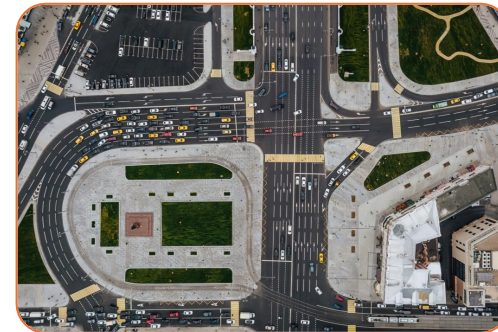
Fast, Spontaneous, Pattern-driven



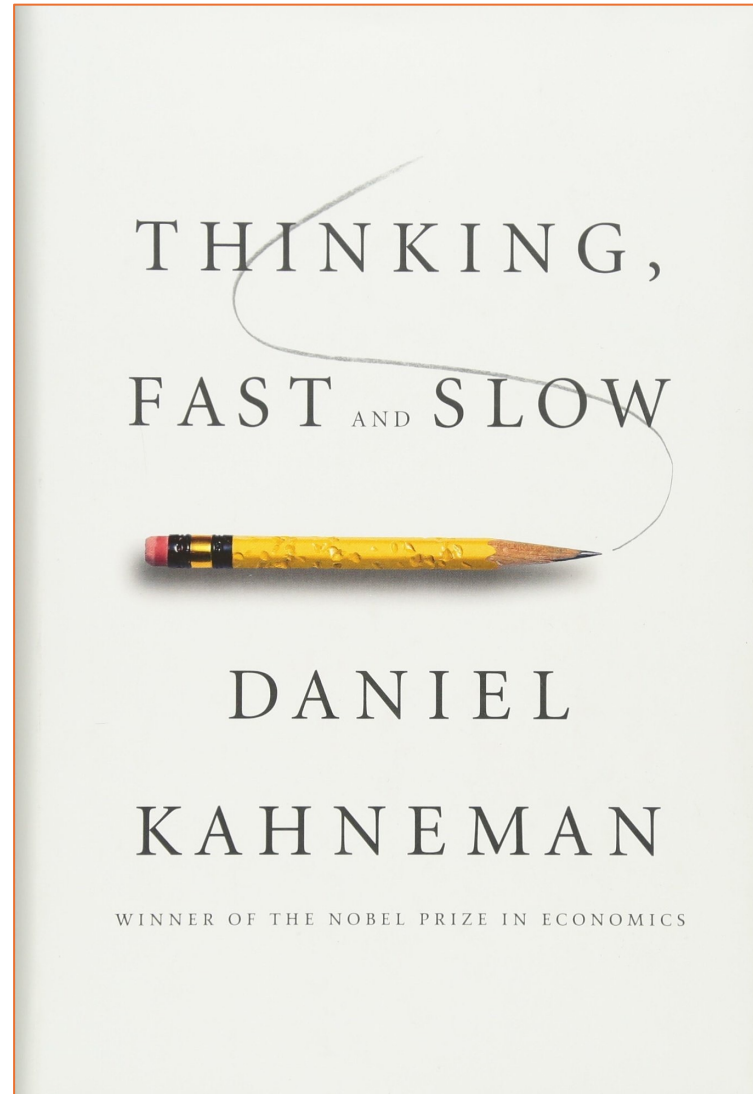
Deliberate Thinking

[System 2]

Slow, Logical, Rule-driven



Prompting Language Models to “Think”



Prompting Language Models to “Think”

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

**Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou**

Google Research, Brain Team
{jasonwei, dennyzhou}@google.com

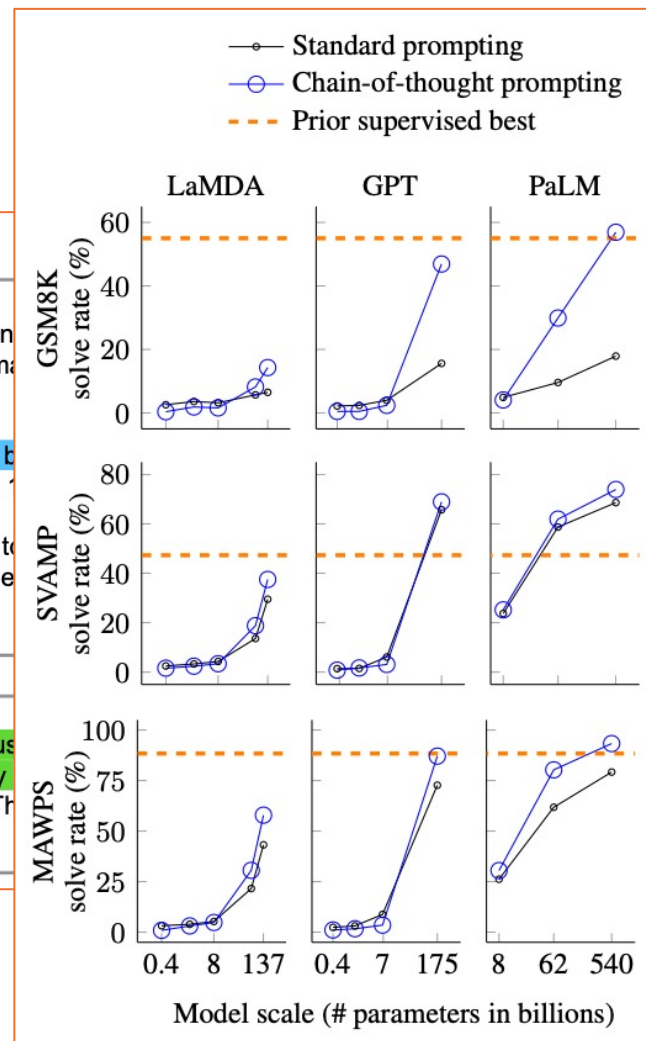
Prompting Language Models to “Think”

Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. ❌</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅</p>

Augmenting Few-Shot (1-shot) with Thinking Process

Prompting Language Models to “Think”

Standard Prompting	Chain-of-Thought Prompting
<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p>Model Input</p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p>Model Output</p> <p>A: The answer is 27. ❌</p>	<p>Model Output</p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅</p>



Prompting Language Models to “Think”

Large Language Models are Zero-Shot Reasoners

Takeshi Kojima

The University of Tokyo

t.kojima@weblab.t.u-tokyo.ac.jp

Shixiang Shane Gu

Google Research, Brain Team

Machel Reid

Google Research*

Yutaka Matsuo

The University of Tokyo

Yusuke Iwasawa

The University of Tokyo

Large Language Models are Zero-Shot Reasoners

The
t.kojima@w

Machel Reid
Google Research

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.**

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Large Language Models are Zero-Shot Reasoners

Takeshi Kojima

The University of Tokyo

t.kojima@weblab.t.u-tokyo.ac.jp

Shixiang Shane Gu

Google Research, Brain Team

Machel Reid

Google Research*

	Arithmetic					
	SingleEq	AddSub	MultiArith	GSM8K	AQUA	SVAMP
zero-shot	74.6/ 78.7	72.2/77.0	17.7/22.7	10.4/12.5	22.4/22.4	58.8/58.7
zero-shot-cot	78.0/78.7	69.6/74.7	78.7/79.3	40.7/40.5	33.5/31.9	62.1/63.7
	Common Sense		Other Reasoning Tasks		Symbolic Reasoning	
	Common SenseQA	Strategy QA	Date Understand	Shuffled Objects	Last Letter (4 words)	Coin Flip (4 times)
zero-shot	68.8/72.6	12.7/ 54.3	49.3/33.6	31.3/29.7	0.2/-	12.8/53.8
zero-shot-cot	64.6/64.0	54.8/52.3	67.5/61.8	52.4/52.9	57.6/-	91.4/87.8

Table 4: Robustness study against template measured on the MultiArith dataset with text-davinci-002. (*1) This template is used in Ahn et al. [2022] where a language model is prompted to generate step-by-step actions given a high-level instruction for controlling robotic actions. (*2) This template is used in Reynolds and McDonell [2021] but is not quantitatively evaluated.

No.	Category	Template	Accuracy
1	instructive	Let's think step by step.	78.7
2		First, (*1)	77.3
3		Let's think about this logically.	74.5
4		Let's solve this problem by splitting it into steps. (*2)	72.2
5		Let's be realistic and think step by step.	70.8
6		Let's think like a detective step by step.	70.3
7		Let's think	57.5
8		Before we dive into the answer,	55.7
9		The answer is after the proof.	45.7
10	misleading	Don't think. Just feel.	18.8
11		Let's think step by step but reach an incorrect answer.	18.7
12		Let's count the number of "a" in the question.	16.7
13		By using the fact that the earth is round,	9.3
14	irrelevant	By the way, I found a good restaurant nearby.	17.5
15		AbraKadabra!	15.5
16		It's a beautiful day.	13.1
-		(Zero-shot)	17.7

Zero-Shot Chain-of-Thought Prompting

You

You are tasked to write this function:

```
def min_cost_k_perio...(x, ...):
```

Before writing it, please first think step-by-step. analyze what needs to be done and sketch out a piece of pseudo code; also verify your pseudo code with some imaginary input-output examples. **Wrap your code in <code></code>.**

Language Model (Actual Generation)

I will write the function `min_cost_k_perio...` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment...

```
<code>
def min_cost_k_perio...(x, ...):
    ...
</code>
```


Zero-Shot Chain-of-Thought Prompting

You

You are tasked to write this function:

```
def min_cost_k_perio...(x, ...):
```

Before writing it, please first think step-by-step. analyze what needs to be done and sketch out a piece of pseudo code; also verify your pseudo code with some imaginary input-output examples.

Language Model (Actual Generation)

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment...

You

Good. Please go ahead and write the function.

Language Model (Actual Generation)

```
def min_cost_k_perio...(x, ...):  
    ...
```

Zero-Shot Chain-of-Thought Prompting

You

You are tasked to write this function:

```
def min_cost_k_perio...(x, ...):
```

Before writing it, please first think step-by-step. analyze what needs to be done and sketch out a piece of pseudo code; also verify your pseudo code with some imaginary input-output examples.

Language Model (Actual Generation)

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment...

You

Good. Please go ahead and write the function.

Language Model (Actual Generation)

```
def min_cost_k_perio...(x, ...):  
    ...
```

Two-Staged; No extra human annotation

Zero-Shot Chain-of-Thought Prompting

You

You are tasked to write this function:

```
def min_cost_k_perio...(x, ...):
```

Before writing it, please first think step-by-step. analyze what needs to be done and sketch out a piece of pseudo code; also verify your pseudo code with some imaginary input-output examples.

Language Model (Actual Generation)

I will write the function `min_cost_k_periodic_segmentation` to the specified requirements. The problem appears to be a dynamic programming challenge. The core idea is to find the minimum cost to segment...

You

Good. Please go ahead and write the function.


Language Model (Actual Generation)

```
def min_cost_k_perio...(x, ...):  
    ...
```

Two-Staged; No extra human annotation

$\Pr(\mathbf{y} \mid \text{prompt}_2(\mathbf{x}) :: \text{prompt}_1(\mathbf{x}) :: \mathbf{x})$

Structured Chain-of-Thought Prompting for Code Generation

Jia Li 
lijia@stu.pku.edu.cn
Peking University
Beijing, China

Yongmin Li
Peking University
Beijing, China
liyongmin@pku.edu.cn

Ge Li
Peking University
Beijing, China
lige@pku.edu.cn

Zhi Jin
Peking University
Beijing, China
zhijin@pku.edu.cn

1. Initialize a result with -999999
2. Iterate through the list of lists
3. Initialize a sum with 0
4. Iterate through the list
5. Add the element to the sum
6. Update result with the maximum of sum and result
7. Divide the result by K
8. Return the result

(a) Chain-of-Thought

Input: array: list[list], K: int

Output: result: int or float

```

1: Initialize a result with -999999
2: for list in the list of lists:
3:     Calculate the sum of the list
4:     if the sum is great than result:
5:         Update the result
6: Divide result by K
7: return result
    
```

Diagram illustrating the flow of the code generation process. The code is enclosed in a dashed box. Arrows point from the code to labels: 'Loop Structure' points to the 'for' loop (lines 2-3), 'Branch Structure' points to the 'if' statement (lines 4-5), and 'Sequence Structure' points to the 'return' statement (line 7).

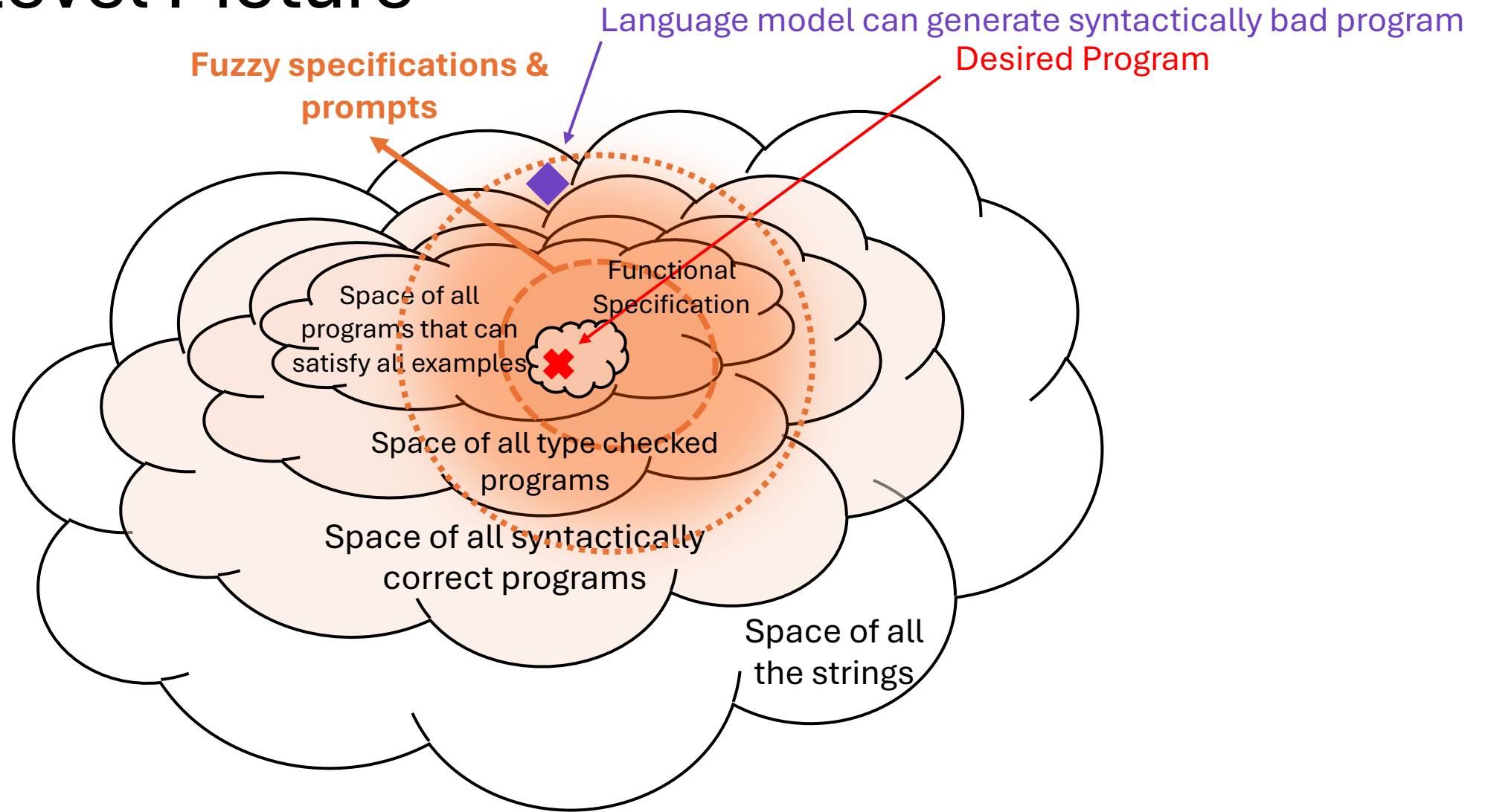
(b) Structured Chain-of-Thought

Table 2: The Pass@k (%) of SCoT prompting and baselines on three code generation benchmarks. The numbers in red denote SCoT prompting’s relative improvements compared to the SOTA baseline - CoT prompting.

Base Model	Prompting Technique	HumanEval			MBPP			MBCPP		
		Pass@1	Pass@3	Pass@5	Pass@1	Pass@3	Pass@5	Pass@1	Pass@3	Pass@5
ChatGPT	Zero-shot prompting	49.73	66.07	71.54	37.07	43.54	48.58	47.53	60.09	64.22
	Few-shot prompting	52.47	69.32	74.10	40.00	49.82	53.13	52.58	63.03	66.11
	CoT prompting	53.29	69.76	75.52	41.83	51.04	54.57	53.51	63.84	67.03
	SCoT Prompting	60.64	73.53	77.32	46.98	55.31	58.36	57.06	65.70	68.70
Relative Improvement		13.79%	5.40%	2.38%	12.31%	8.37%	6.95%	6.63%	2.91%	2.49%
Codex	Zero-shot prompting	40.20	61.78	68.11	27.07	43.81	47.93	40.25	54.17	60.65
	Few-shot prompting	42.93	62.96	70.10	33.17	45.72	49.62	44.12	57.65	62.45
	CoT prompting	43.79	63.41	71.56	35.66	46.57	50.11	45.79	58.92	62.56
	SCoT Prompting	49.82	66.56	75.14	38.29	50.74	53.16	48.34	60.77	64.19
Relative Improvement		13.77%	4.97%	5.00%	7.38%	8.95%	6.09%	5.57%	3.14%	2.61%

Figure 1: The comparison of a Chain-of-Thoughts (CoT) and our Structured Chain-of-Thought (SCoT).

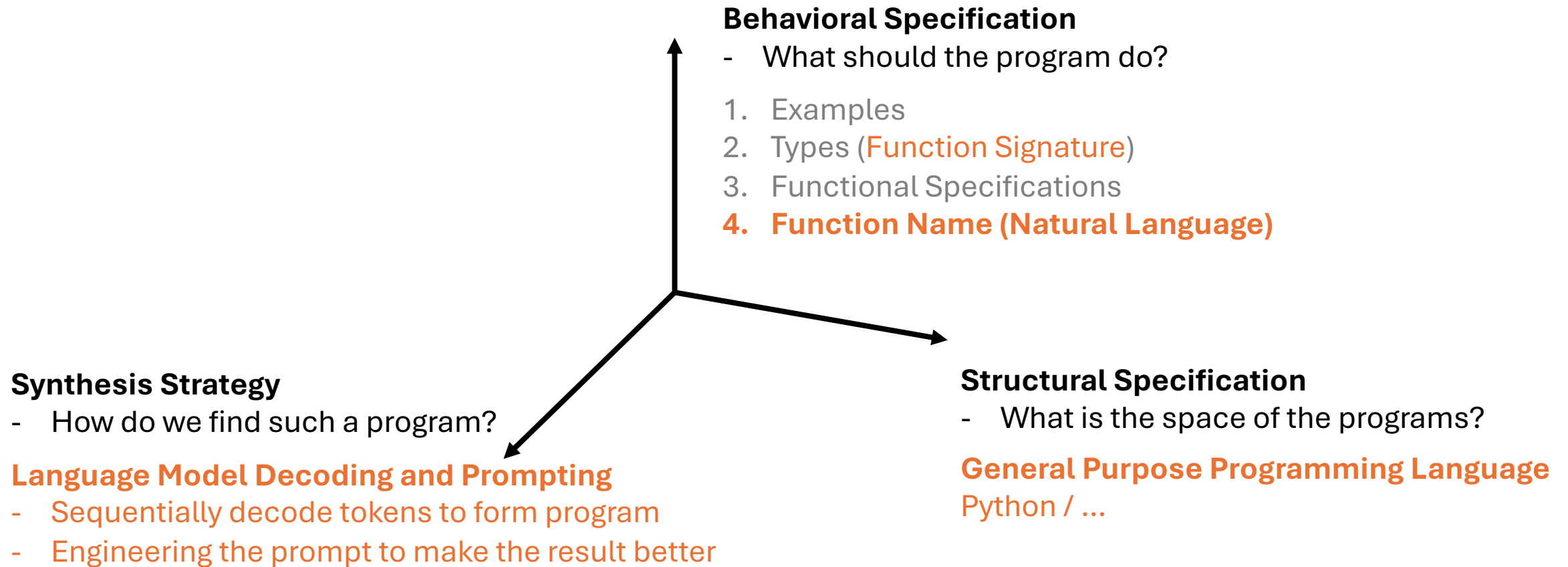
High Level Picture



Summary

- Topics we have covered
 - Basics of decoding
 - From next token prediction to sequence prediction
 - Sampling algorithms for decoding
 - Basics of prompting
 - Crafting the prompt in specific manners to elicit behaviors in language models
 - Zero-shot prompting, Role-prompting
 - Few-shot prompting
 - Chain-of-thought (CoT prompting), along with FS-CoT & ZS-CoT
- Topics we have not covered
 - Advanced decoding (Aligning grammar, syntax, and semantics)
 - Other prompting strategies (Self-Consistency, Self-Reflection, Tree-of-thought, ...)
 - Technical details: how to evaluate correctness, how to parse LLM output, etc.
 - How is LLM trained? Why does it work? When does it not work? What do we do then?

Today



Week 3

- Assignment 1
 - <https://github.com/machine-programming/assignment-1>
 - Due next Tuesday (Sep 16), 5 days left!
- Assignment 2
 - Evaluating language models and testing different prompting strategies!
 - Will be released during the Weekend!
- Attendance:
 - Starting checking next week, will send courselore post
- Oral Presentation:
 - Start bidding topics/papers/slots from next week, will send courselore post

Recommended Readings

- [A Survey on Large Language Models for Code Generation](#), Jiang et. al., 2024
- [Is Self-Repair a Silver Bullet for Code Generation?](#) Olausson et. al., 2024
- [Automatically Correcting Large Language Models: Surveying the landscape of diverse self-correction strategies](#), Pan et. al., 2023
- [The Curious Case of Neural Text Degeneration](#), Holtzman et. al., 2020
- [Planning with Large Language Models for Code Generation](#), Zhang et. al., 2023
- [Tree of Thoughts: Deliberate Problem Solving with Large Language Models](#), Yao et. al., 2023
- [Syntactic and Semantic Control of Large Language Models via Sequential Monte Carlo](#), Loula et. al., 2025
- [Large Language Models are Zero-Shot Reasoners](#), Kojima et. al., 2023
- [Chain-of-Thought Prompting Elicits Reasoning in Large Language Models](#), Wei et. al., 2022
- [Tree-of-Code: A Hybrid Approach for Robust Complex Task Planning and Execution](#), Li et. al., 2024
- [Defeating Nondeterminism in LLM Inference](#), Horace He in collaboration with others at Thinking Machines, 2025