



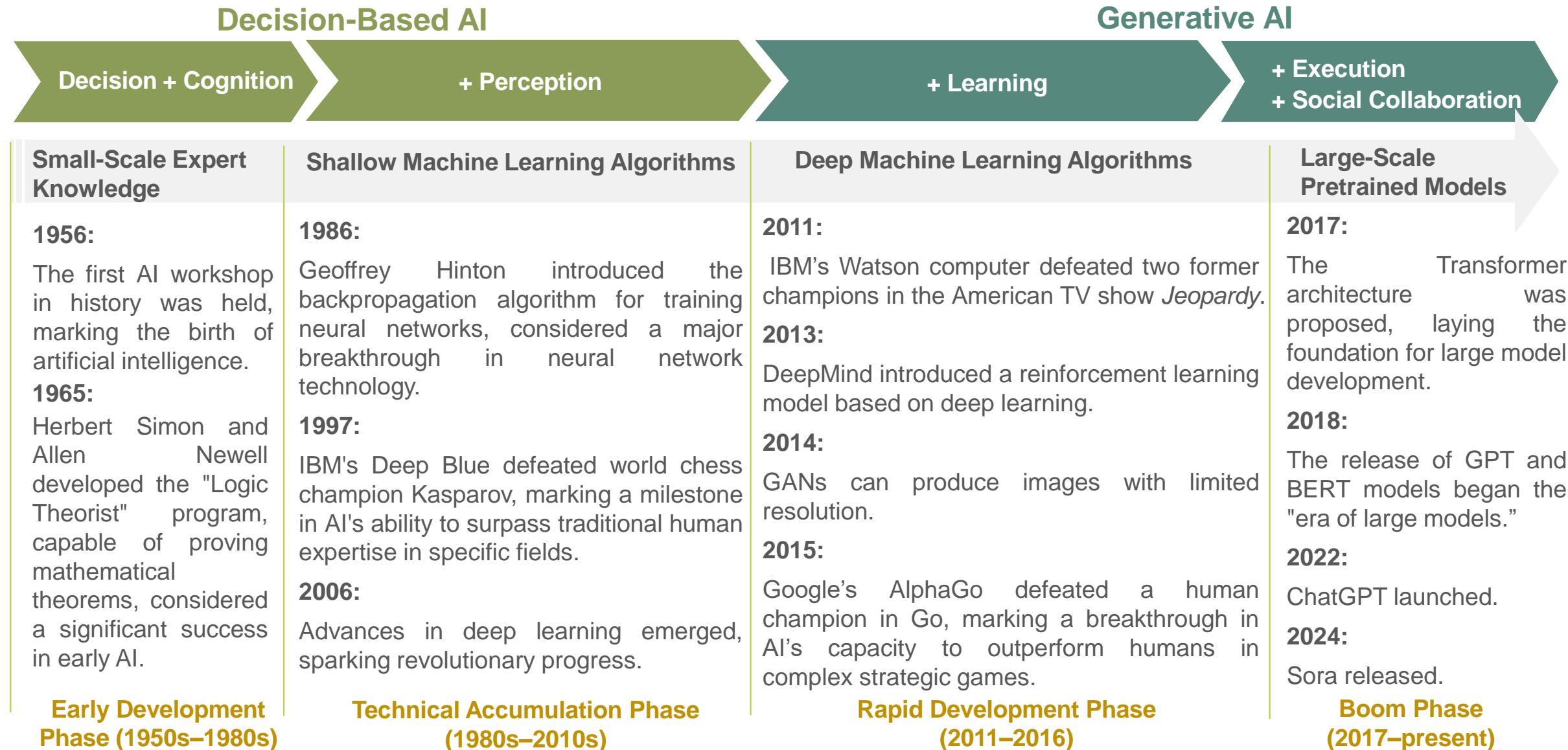
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Introduction to Computer Engineering: Programming and Applications

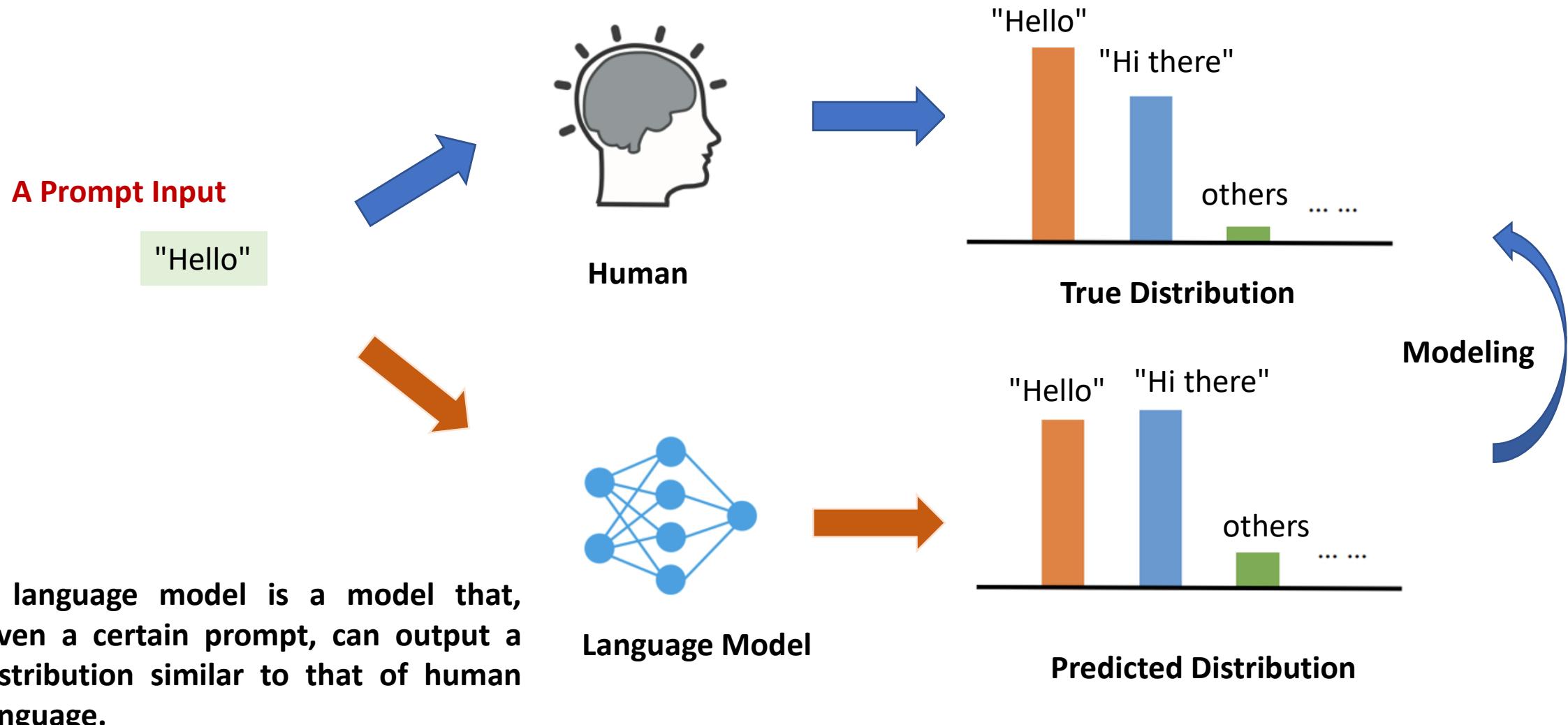
AI Assisted Programming

**Prof. Junhua Zhao
School of Science and Engineering**

Modern History of Artificial Intelligence



What is a Language Model



Large Language Model

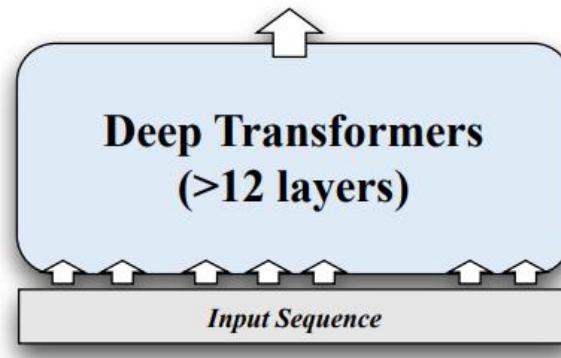


Three Key Elements of Pre-training Large Language Models:

- **Big Data:** The source of knowledge, encompassing various languages, phenomena, and semantic knowledge, which directly determines the learning scope of the model.
- **Large Model:** The carrier of big data, typically composed of deep neural networks, learning statistical information and abstract knowledge from large datasets.
- **High Computational Power:** Parallel computing clusters that handle big data and large models, usually including high-performance computing devices like GPUs, TPUs, etc.



Big Data
(Binary code representation without captions)



Large Model
(Deep Neural Networks)



High Computational Power
(Parallel computing clusters)

Prompt Engineering?



Prompt Engineering refers to the process of designing the **prompts or inputs** that will be given to a language model. The prompts can be anything from a short phrase to a full sentence or even a paragraph. The goal of prompt engineering is to design prompts that will **produce high-quality, relevant, and coherent responses** from the model.

Why is Prompt Engineering important?

Prompt engineering has the potential to transform the way you create, work, and communicate. It enables you to interact with language models in a more **natural and intuitive** way and **elicit powerful results** that will save you time and money and, in many cases, far exceed previous outcomes.

The primary benefit of prompt engineering is the ability to achieve **optimized outputs** with **minimal post-generation effort**. Generative AI outputs can be mixed in quality, often requiring skilled practitioners to review and revise. By crafting precise prompts, prompt engineering ensures that AI-generated output aligns with the desired goals and criteria, reducing the need for extensive post-processing.

In addition, it can also solve your assignment better and more effectively!

Prompting an LLM

You can achieve a lot with simple prompts, but the quality of results depends on **how much information** you provide it and **how well-crafted** the prompt is. A prompt can contain information like the instruction or question you are passing to the model and include other details such as context, inputs, or examples. You can use these elements to instruct the model more effectively to improve the quality of results.

A basic example of a simple prompt:

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



It looks like you're starting to define a function to add two numbers. Here's how you can complete the function:

```
python
```

复制代码

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

You can now call this function with two arguments to get their sum:

```
python
```

复制代码

```
result = add(3, 5)  
print(result) # Output: 8
```

Let me know if you need further assistance with this function or anything else!



You can observe from the prompt example above that the language model responds with a sequence of tokens that make sense given the context "def add(a, b): ". The output might be **unexpected** or **far from** the task you want to accomplish. In fact, this basic example highlights the necessity to **provide more context or instructions** on what specifically you want to achieve with the system. This is what prompt engineering is **all about**.

Let's try to improve it a bit:

Complete the following unfinished code with comments

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



Here's a simple completion of the function with comments:

```
python
```

```
def add(a, b):
    """
    This function takes two numbers as arguments
    and returns their sum.
    """
    # Adding the two numbers
    result = a + b

    # Returning the result
    return result

# Example usage of the add function
if __name__ == "__main__":
    # Adding 3 and 5
    sum_result = add(3, 5)
    print(f"The sum of 3 and 5 is: {sum_result}") # Expected output: The sum of 3 and 5 is 8
```

Let me know if you need further explanation or enhancements for this code.

Zero-shot Prompting

You can format this into a question answering (QA) format, which is standard in a lot of QA datasets, as follows:

Q: <Question>?

A:

When prompting like the above, it's also referred to as zero-shot prompting, i.e., you are directly prompting the model for a response without any examples or demonstrations about the task you want it to achieve.

Prompt:

Q: What **is** the capital **of** France?

A:

Answer:

A: Paris

Few-shot Prompting

Given the standard format above, one popular and effective technique to prompting is referred to as few-shot prompting where you provide exemplars (i.e., demonstrations). You can format few-shot prompts as follows:

Prompt:

Q: What **is** the capital **of** Italy?

A: The capital **of** Italy **is** Rome.

Q: Who discovered penicillin?

A: Penicillin was discovered **by** Alexander Fleming **in** 1928.

Q: What **is** the main ingredient **in** guacamole?

A: The main ingredient **in** guacamole **is** avocado.

Q: What **is** the largest ocean **on** Earth?

A:

Answer:

A: The largest ocean **on** Earth **is** the Pacific Ocean.

General Tips for Designing Prompts

Start Simple

As you get started with designing prompts, you should keep in mind that it is really an **iterative** process that requires a lot of experimentation to get optimal results. You can start with **simple** prompts and keep **adding** more elements and context as you aim for better results.

When you have a big task that involves many different subtasks, you can try to **break down** the task into simpler subtasks and keep building up as you get better results. This avoids adding too much complexity to the prompt design process at the beginning.

General Tips for Designing Prompts

The Instruction

You can design effective prompts for various simple tasks by using **commands** to instruct the model what you want to achieve, such as "Write", "Debug", "Check", etc.

Try different instructions with different keywords, contexts, and data and see what works best for your particular use case and task. Others recommend that you place instructions at the **beginning** of the prompt. Another recommendation is to use **clear separator** like "###" to separate the instruction and context.

Example

Prompt:

```
###Instruction###
Add comments to the function.
###code###
def add(a,b):
    return a+b
```

Answer:

```
def add(a, b):
    """
    This function takes two numbers as input and returns their sum.

    Parameters:
        a (int or float): The first number to add.
        b (int or float): The second number to add.

    Returns:
        int or float: The sum of the two input values.
    """

    # Add the two input values
    return a + b
```

A Better Prompt

Prompt:

```
###Instruction###
```

Rewrite the add function to include:

1. A detailed docstring that describes the function's purpose, parameters, return values, and potential exceptions.
2. Type annotations for inputs and outputs.
3. Input validation to handle cases where the inputs are not numbers.
4. A test example demonstrating usage.

```
###code###
```

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

```
    return a+b
```

```
def add(a: float, b: float) -> float:  
    """  
    Adds two numbers and returns their sum.  
  
    Parameters:  
    a (float): The first number to add. Must be an integer or float.  
    b (float): The second number to add. Must be an integer or float.  
  
    Returns:  
    float: The sum of `a` and `b`.  
  
    Raises:  
    TypeError: If either `a` or `b` is not an integer or float.  
  
    Example:  
    >>> add(3, 5)  
    8  
    >>> add(2.5, 4.3)  
    6.8  
    >>> add("3", 5)  
    TypeError: Both inputs must be integers or floats.  
    """  
  
    # Input validation to ensure both inputs are numbers  
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
        raise TypeError("Both inputs must be integers or floats."  
  
    # Return the sum of the two numbers  
    return a + b  
  
# Example usage  
try:  
    result = add(3, 5)  
    print("Sum:", result) # Expected output: Sum: 8  
except TypeError as e:  
    print("Error:", e)
```

General Tips for Designing Prompts

Be specific

Be very specific about the instruction and task you want the model to perform. The more **descriptive** and **detailed** the prompt is, the better the results. This is particularly important when you have a desired outcome or style of generation you are seeking. There aren't specific tokens or keywords that lead to better results. It's more important to have a good **format** and **descriptive** prompt. In fact, **providing examples** in the prompt is very effective to get desired output in specific formats.

General Tips for Designing Prompts

Be specific (Cont.)

When designing prompts, you should also keep in mind the length of the prompt. Including too many **unnecessary** details is not necessarily a good approach. The details should be **relevant** and **contribute** to the task at hand. This is something you will need to experiment with a lot.

Example

Prompt:

"Write a Python function named calculate_discount that takes two arguments: price (a float representing the original price of an item) and discount_percentage (an integer representing the discount to be applied in percent). The function should return the discounted price rounded to two decimal places. Include error handling to ensure price is positive and discount_percentage is between 0 and 100."

Answer:

```
def calculate_discount(price: float, discount_percentage: int):
    if price <= 0:
        raise ValueError("Price must be a positive number.")
    if not (0 <= discount_percentage <= 100):
        raise ValueError("Discount percentage must be between 0 and 100.")

    discount = price * (discount_percentage / 100)
    discounted_price = price - discount
    return round(discounted_price, 2)

print(calculate_discount(100,10))
```

General Tips for Designing Prompts

Avoid Imprecision

Given the tips above about being detailed and improving format, it's easy to fall into the trap of wanting to be too clever about prompts and potentially creating imprecise descriptions. It's often better to be **specific** and **direct**. The analogy here is very similar to effective communication -- the more direct, the more effective the message gets across.

Bad Example

Prompt:

"Write a function to generate Fibonacci numbers."

Answer:

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Good Example

Prompt:

"Write a Python function named `generate_fibonacci_sequence` that takes an integer `n` as an argument. The function should return a list containing the first `n` numbers in the Fibonacci sequence, starting from 0. Use an iterative approach instead of recursion to handle large values of `n` efficiently. Include error handling to ensure that `n` is a positive integer greater than zero."

Answer:

```
def generate_fibonacci_sequence(n: int) -> list:
    if n <= 0:
        raise ValueError("Input must be a positive integer greater than zero.")

    sequence = [0, 1]
    for _ in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])

    return sequence[:n]
```

General Tips for Designing Prompts

To do or not to do?

Another common tip when designing prompts is to avoid saying **what not to do** but say **what to do** instead. This encourages more specificity and focuses on the details that lead to good responses from the model.

Bad Example

- Negativity: The prompt focuses on what not to include rather than what should be covered.
- Lack of Direction: It leaves too much open to interpretation, making it difficult to understand the user's actual needs.

Don't just give me a simple explanation of Java. Avoid discussing its history or basic syntax. I don't want to hear about its popularity either.



It seems you're looking for a more focused and detailed answer about Java, excluding basic topics like history, syntax, and popularity. Here's a precise and in-depth prompt that avoids those aspects:

Revised Prompt:

"Explain the advanced features of Java that make it suitable for building scalable, high-performance applications. Focus on features such as the JVM's role in optimizing memory management, the concurrency model, and garbage collection mechanisms. Additionally, discuss how Java's platform independence and security features contribute to enterprise application development."

This prompt focuses on advanced aspects and practical features of Java without touching on introductory topics, giving you a deeper insight into Java's strengths in a professional context.

Good Example

Prompt:

Please provide a detailed overview of Java's core features, focusing on its object-oriented programming capabilities, common libraries like Java Collections Framework, and its applications in enterprise software development. If possible, include examples of how these features are utilized in real-world scenarios.



Java is a versatile, high-level programming language with several core features that make it ideal for enterprise software development. Here's a breakdown of its most significant aspects:

1. Object-Oriented Programming (OOP) Capabilities

- **Encapsulation:** Java promotes encapsulation by allowing classes to define public interfaces and private/internal states. This allows developers to control access to data and ensures modular design.
- **Inheritance:** Through inheritance, Java allows classes to derive properties and behaviors from other classes, promoting code reuse and a hierarchical structure.
- **Polymorphism:** Java supports polymorphism, enabling a single interface to represent different underlying forms (types). This is heavily used in designing flexible and maintainable applications.
- **Abstraction:** Java allows developers to define abstract classes and interfaces, which helps in defining a common set of behaviors that can be implemented differently across various classes.

Example: In a banking application, a `Customer` class might inherit from a `Person` class, while different types of accounts (e.g., `CheckingAccount`, `SavingsAccount`) can inherit common properties from a base `Account` class. This setup facilitates modular design and easy extension.

2. Java Collections Framework

- **Core Libraries:** The Java Collections Framework provides essential data structures and algorithms for handling groups of objects. Key components include:
 - **List** (e.g., `ArrayList`, `LinkedList`): An ordered collection of elements that can contain duplicates.
 - **Set** (e.g., `HashSet`, `Treeset`): A collection that cannot contain duplicate elements.
 - **Map** (e.g., `HashMap`, `TreeMap`): A mapping between keys and values, allowing efficient lookups.



Good Example

- **Utilities and Algorithms:** The framework includes a variety of utility methods (e.g., sorting, searching) that make it easier to manipulate data in collections.
- **Generics:** Java collections use generics, allowing developers to specify the type of objects in a collection, which enhances type safety and readability.

Example: In an e-commerce application, `List` might be used to store items in a shopping cart, `Map` to store product inventories with product IDs as keys, and `Set` to store unique user IDs for registered customers.

3. Memory Management and Garbage Collection

- Java handles memory allocation and deallocation automatically through a process known as garbage collection. The Java Virtual Machine (JVM) monitors object references and periodically removes objects that are no longer in use, freeing up memory.
- **Automatic Garbage Collection:** This feature reduces the likelihood of memory leaks and ensures efficient memory usage. The JVM's garbage collector uses different algorithms (e.g., mark-and-sweep, generational garbage collection) to manage memory effectively.

Example: In a high-transaction enterprise application (like a stock trading platform), efficient memory management is crucial. Automatic garbage collection ensures that resources are available for processing new transactions by clearing memory occupied by unreferenced objects.

4. Platform Independence

- Java is compiled into bytecode, an intermediate form, rather than machine code. This bytecode can run on any device with a JVM, making Java applications highly portable.
- **Write Once, Run Anywhere (WORA):** This principle allows Java code to be executed on different platforms (Windows, Mac, Linux) without modification, which is a significant advantage in enterprise environments where software may need to operate across various systems.

Example: Many financial institutions build Java applications that run seamlessly across multiple operating systems, allowing them to deploy their software across different offices and data centers without compatibility issues.

Prompting Techniques



Zero-shot Prompting



Few-shot Prompting



**Chain-of-Thought
Prompting**



Meta Prompting



Self-Consistency



**Generate Knowledge
Prompting**



Prompt Chaining



Tree of Thoughts



**Retrieval Augmented
Generation**



**Automatic Reasoning
and Tool-use**



**Automatic Prompt
Engineer**



Active-Prompt

Chain-of-Thought (CoT) Prompting

Standard Prompting

Model Input

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: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. 

Chain-of-Thought Prompting

Model Input

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: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

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. 

Zero-shot CoT Prompting

One recent idea that came out more recently is the idea of zero-shot CoT (Kojima et al. 2022) that essentially involves adding "**Let's think step by step**" to the original prompt.

Without Zero-Shot CoT Prompting:

"Write a Python function to solve a Sudoku puzzle."

Response:

```
python
```

复制代码

```
def solve_sudoku(board):
    # Function to check if the number can be placed
    def is_valid(board, row, col, num):
        # Check row
        for i in range(9):
```



ERROR!

Zero-shot COT Prompting (Cont.)

With Zero-Shot CoT Prompting:

"Let's think step by step. Write a Python function to solve a Sudoku puzzle."

Response:

1. **Understand the Problem:** I need to write a function that fills in a Sudoku puzzle, which is a 9x9 grid with some numbers already filled in. The goal is to fill the empty cells (denoted as 0) according to Sudoku rules.
2. **Define Valid Placement:** I should first create a helper function to check if placing a number in a specific cell is valid. This involves checking:
 - The row for duplicates
 - The column for duplicates
 - The 3x3 box for duplicates
3. **Implement Backtracking:** The main function will use backtracking to fill the grid. This means I'll:
 - Look for an empty cell
 - Try placing numbers from 1 to 9 in that cell
 - Recursively call the function to fill the next cell
 - If a number doesn't work out, I backtrack by resetting the cell to 0
4. **Code the Solution:** Now I can write the code to implement this logic.



```
python
```

复制代码

```
def solve_sudoku(board):
```

(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?

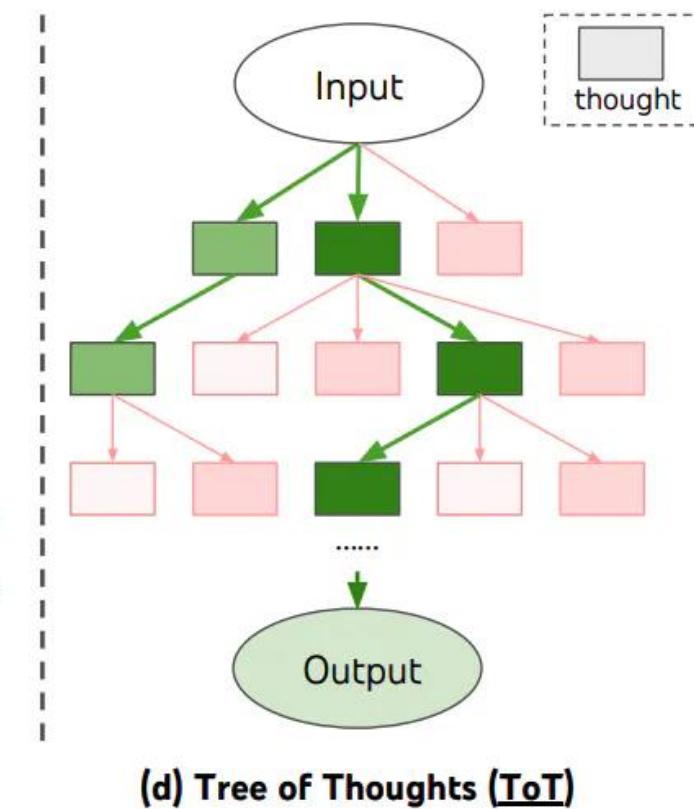
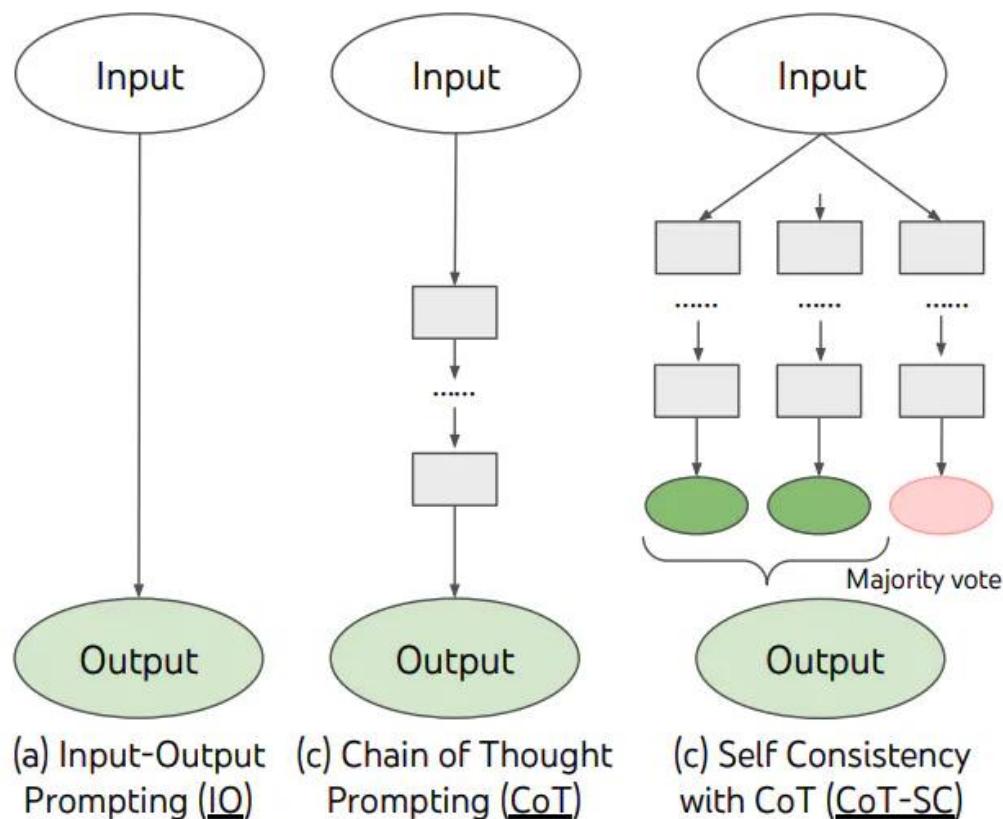
A: Let's think step by step.

(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. ✓

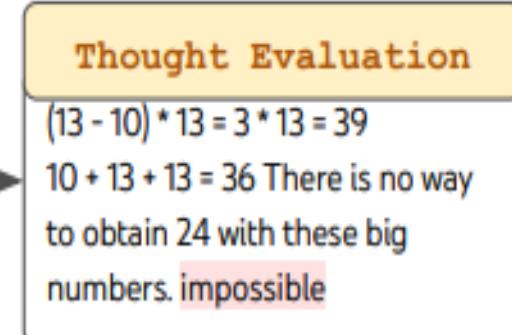
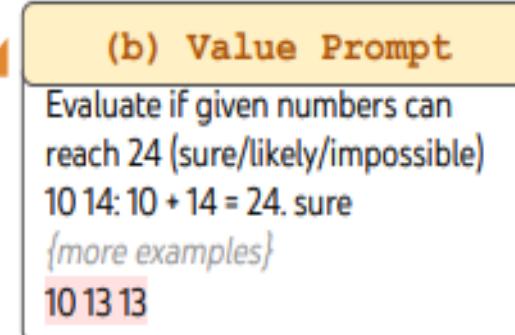
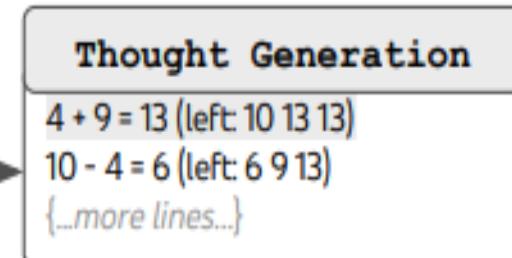
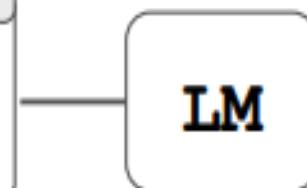
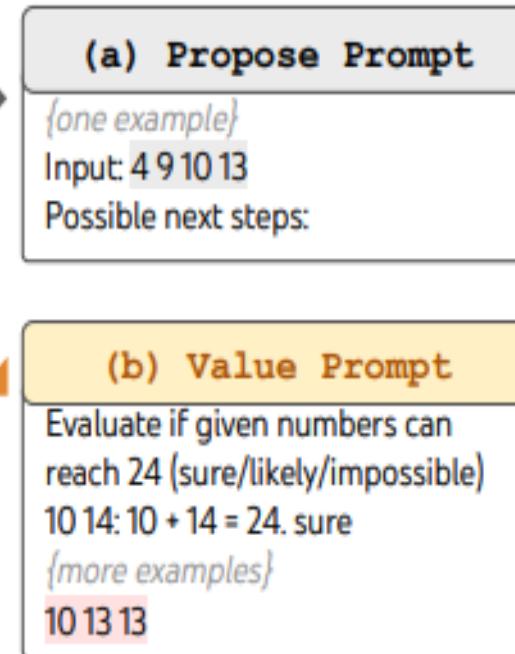
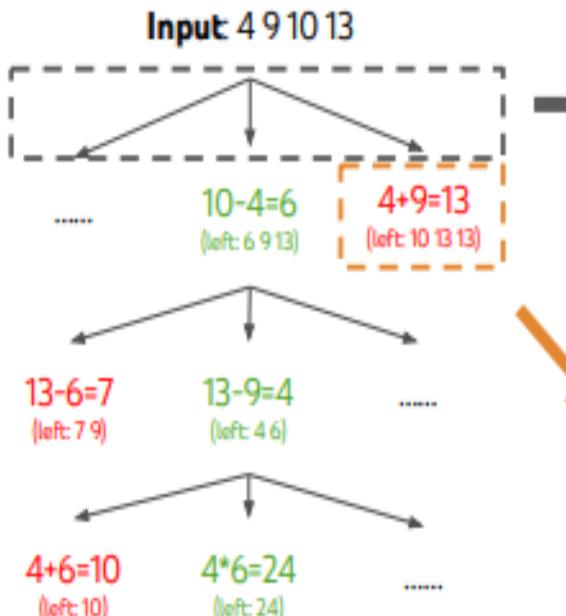
Image Source: Kojima et al. (2022)

Tree of Thoughts (ToT)

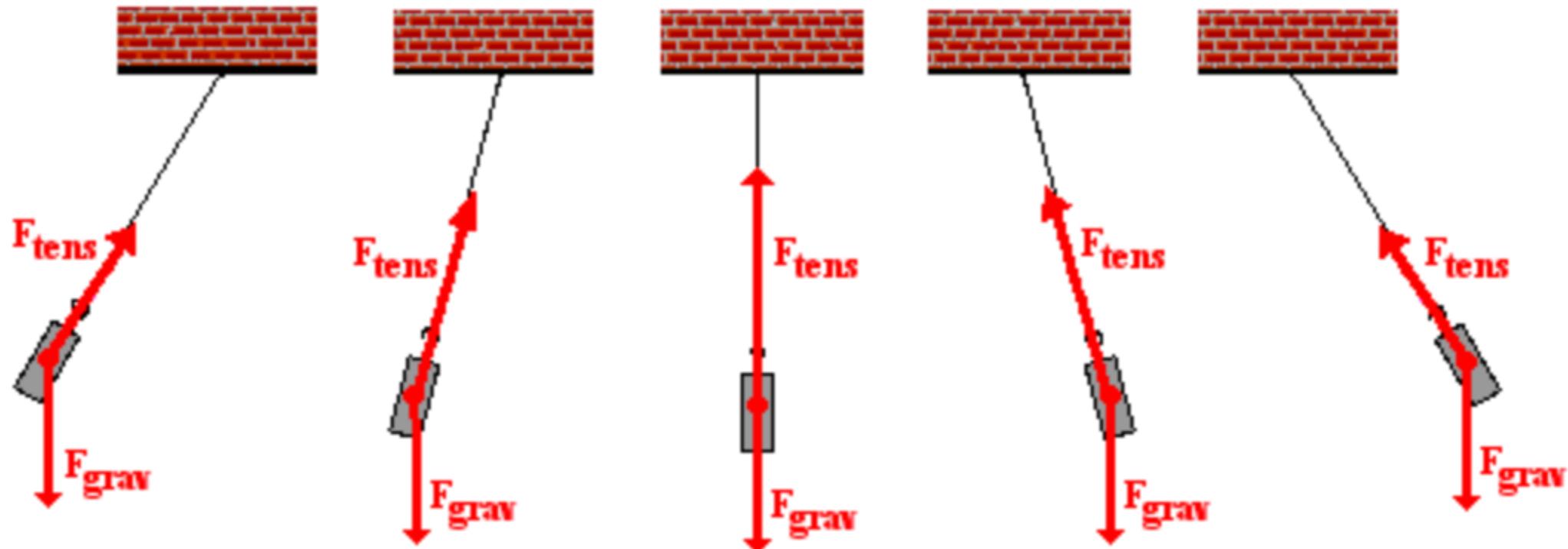
For complex tasks, traditional or simple prompting techniques fall short.



ToT in Game of 24 -- a better way to think



Example: Motion of a Pendulum with Air Resistance



The **force of gravity** is always directed downward; its magnitude never changes. The **tension force** is always directed towards the pivot; it's magnitude varies over the course of a vibrational cycle.

Mathematical Model of Pendulum Motion

Forces Acting on the Pendulum

1. **Gravitational Force:** The weight of the pendulum bob acts downward, providing the restoring force that drives the oscillation.
2. **Tension Force:** The string exerts an upward tension force that changes direction as the bob swings.
3. **Air Resistance:** This force acts in the opposite direction to the motion of the bob and can be modeled as:

$$F_b = -cv$$

where c is a constant that depends on factors like the shape and size of the bob, and v is the velocity of the bob ① ③.

Mathematical Model

For small angular displacements (θ), we can use a linear approximation where $\sin(\theta) \approx \theta$. The equation of motion for a damped pendulum can be expressed as:

$$\ddot{\theta} + 2c\dot{\theta} + \frac{g}{L}\theta = 0$$

where:

- g is the acceleration due to gravity,
- L is the length of the pendulum,
- c is a damping coefficient related to air resistance ③ ④⑥.

ToT Prompt for Solving the Pendulum Motion Problem

Write Python code to simulate the motion of a pendulum with air resistance. Begin by enclosing all thoughts within <thinking> tags, exploring multiple angles and approaches. Break down the solution into clear steps within <step> tags. Start with a 20-step budget, requesting more for complex problems if needed. Use <count> tags after each step to show the remaining budget. Stop when reaching 0. Continuously adjust your reasoning based on intermediate results and reflections, adapting your strategy as you progress. Synthesize the final answer within <answer> tags, providing a clear, concise summary. Conclude with a final reflection on the overall solution, discussing effectiveness, challenges, and solutions. Assign a final reward score.

Results



<thinking> To simulate the motion of a pendulum with air resistance, I will explore various approaches, considering both the physics involved and numerical methods for solving the equations of motion. The goal is to provide an accurate and efficient simulation. </thinking>

<step>1. Identify key parameters: Initial angle (θ_0), initial velocity (v_0), length of the pendulum (L), drag coefficient (b), time step (dt), and total simulation time (T).</step>
<count>19</count>

<step>2. Determine the governing equations: The motion of the pendulum can be described by the equation $\theta'' = -\frac{g}{L} \sin(\theta) - \frac{b}{m}v$, where g is the acceleration due to gravity and m is the mass.</step> <count>18</count>

<step>3. Choose a numerical integration method: Options include Euler's method (simpler but less accurate) and the Runge-Kutta method (more accurate). I will choose the Euler method for simplicity, but will ensure it's implemented correctly.</step>
<count>17</count>

<step>4. Set up the simulation loop: Initialize variables for time, angle, and velocity, then loop over the time steps to update these values according to the equations of motion.
</step> <count>16</count>

<step>5. Implement the drag force: The drag force should be proportional to the velocity and will affect the pendulum's motion. Include it in the acceleration calculation.</step>
<count>15</count>

<step>6. Write the Python function: Structure the code into a function that takes parameters and returns time and angle values over the simulation period.</step> <count>14</count>

<step>7. Create a plotting function: Use Matplotlib to visualize the angle of the pendulum over time. This helps in understanding the motion clearly.</step> <count>13</count>

<step>8. Test the function with sample inputs: Use reasonable values for θ_0 , v_0 , L, b, dt, and T to check if the simulation behaves as expected.</step> <count>12</count>

<step>9. Adjust parameters based on test results: If the simulation does not produce expected results (e.g., unrealistic oscillations), refine the drag coefficient or time step.
</step> <count>11</count>

<step>10. Add comments and documentation: Ensure the code is well-documented for clarity and future reference.</step> <count>10</count>

<step>11. Consider edge cases: Explore how the simulation behaves with extreme values of drag or initial conditions.</step> <count>9</count>

<step>12. Implement error handling: Add checks for input parameters to ensure they are within reasonable bounds.</step> <count>8</count>

<step>13. Optimize performance: If the simulation is slow, consider optimizing the loop or the method used for calculating new angles and velocities.</step> <count>7</count>

<step>14. Conduct further testing: Run multiple test cases to ensure robustness and accuracy of the simulation.</step> <count>6</count>

<step>15. Refine visualizations: Add more features to the plots, such as labels and grid lines, for better readability.</step> <count>5</count>

<step>16. Prepare the final code: Ensure all components work seamlessly together in a clean format.</step> <count>4</count>

<step>17. Reflect on the simulation's accuracy: Analyze results to verify that they match theoretical expectations for a pendulum.</step> <count>3</count>

<step>18. Document final observations: Note any discrepancies and suggest potential improvements for future simulations.</step> <count>2</count>

<step>19. Gather feedback: Share the code with peers to get insights on further improvements or alternative methods.</step> <count>1</count>

<step>20. Finalize the report: Summarize findings and document the entire process, ensuring a comprehensive understanding of the simulation.</step> <count>0</count>

Results

```
def pendulum_with_drag(theta0, v0, length, drag_coefficient, dt, total_time):
    """
        Simulate the motion of a pendulum with air resistance.

    Parameters:
        theta0: Initial angle (in radians)
        v0: Initial velocity (in m/s)
        length: Length of the pendulum (in meters)
        drag_coefficient: Drag coefficient (affecting air resistance)
        dt: Time step for the simulation (in seconds)
        total_time: Total time for the simulation (in seconds)

    Returns:
        t: Time array
        theta_vals: Array of angles over time (in radians)
    """

    g = 9.81 # acceleration due to gravity in m/s^2
    time_steps = int(total_time / dt)
    theta = theta0 # initial angle in radians
    v = v0          # initial velocity in m/s
    t = np.linspace(0, total_time, time_steps)
    theta_vals = []

    for _ in range(time_steps):
        # Calculate acceleration
        a = -(g / length) * np.sin(theta) - drag_coefficient * v
        v += a * dt # update velocity
        theta += v * dt # update angle
        theta_vals.append(theta)

    return t, theta_vals
```

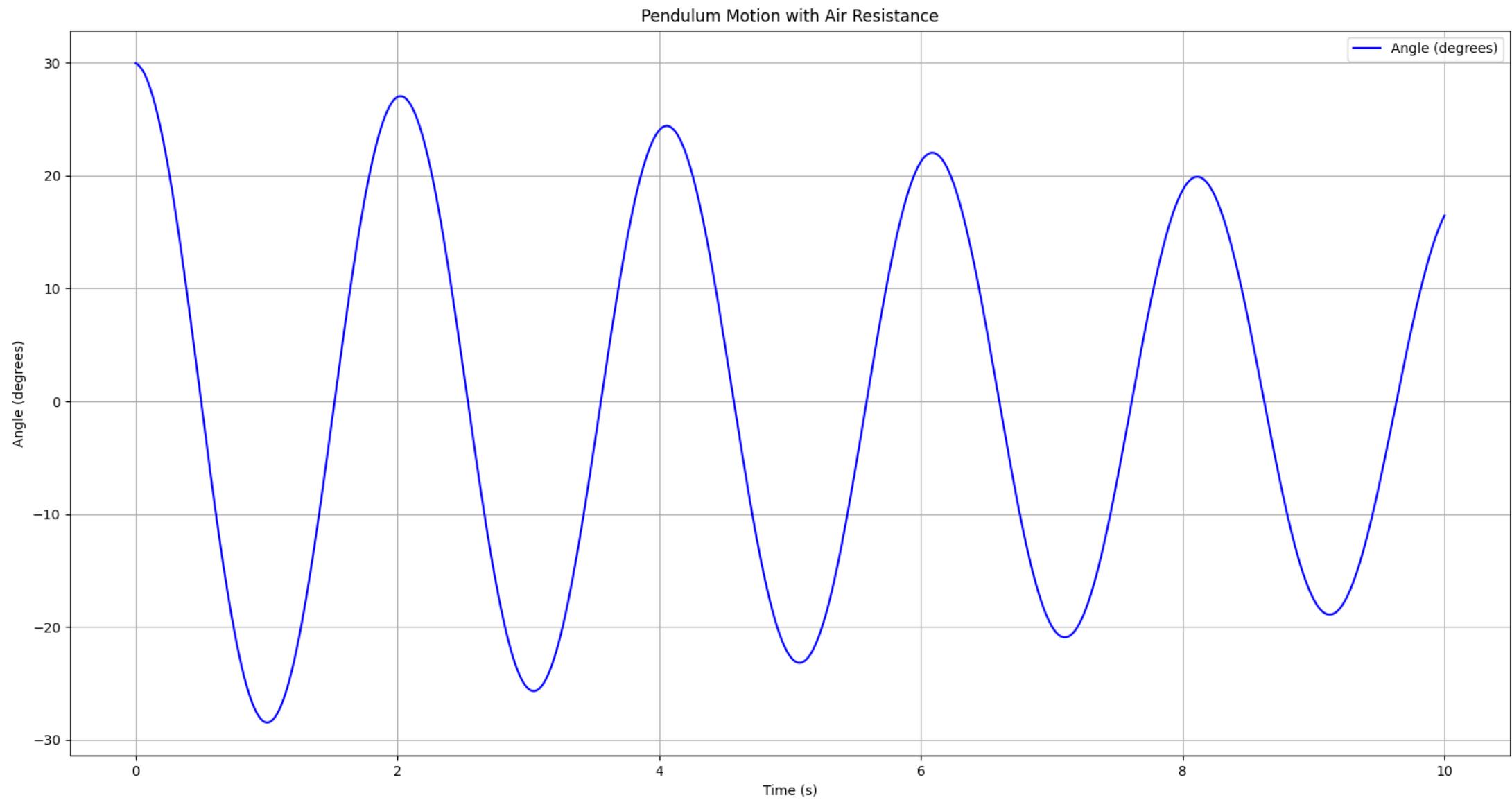
Results

```
# Example usage
# Initial conditions and parameters
theta0 = np.radians(30) # initial angle in radians
v0 = 0 # initial velocity in m/s
length = 1 # length of pendulum in meters
drag_coefficient = 0.1 # drag coefficient
dt = 0.01 # time step in seconds
total_time = 10 # total time in seconds

# Run the simulation
t, theta_vals = pendulum_with_drag(theta0, v0, length, drag_coefficient, dt, total_time)

# Plotting the results
plt.figure(figsize=(10, 5))
plt.plot(t, np.degrees(theta_vals), label='Angle (degrees)', color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Angle (degrees)')
plt.title('Pendulum Motion with Air Resistance')
plt.grid()
plt.legend()
plt.show()
```

Results



Prompt Chaining

To improve the reliability and performance of LLMs, one of the important prompt engineering techniques is to **break tasks** into its subtasks. Once those subtasks have been identified, the LLM is prompted with a subtask and then its **response** is used as **input** to another prompt. This is what's referred to as prompt chaining, where a task is **split** into subtasks with the idea to create a **chain of prompt operations**.

Prompt Chaining (Cont.)

Prompt chaining can be used in different scenarios that could involve several operations or transformations. It helps if you design **two different prompts** where the first prompt is responsible for **extracting relevant functions** to answer a question and a second prompt takes as input the **functions** to answer a given question.

You are a programming assistant. Please extract and provide the relevant functions needed to calculate the average sales amount per product from a sales dataset. Include any necessary libraries or modules and a brief description of each function.



To calculate the average sales amount per product from a sales dataset, you can use Python along with the pandas library. Below are the relevant functions and descriptions for achieving this:

Required Libraries

```
python
```

复制代码

```
import pandas as pd
```

Functions

1. Load Data

- **Description:** Loads the sales dataset from a specified file (CSV, Excel, etc.) into a pandas DataFrame.
- **Function:**

```
python
```

复制代码

```
def load_data(file_path):
```

Prompt2:

Using the functions provided below, write a complete script that loads a sales dataset and calculates the average sales amount per product.

```
# Function to load data from a CSV file
def load_data(file_path):
    return pd.read_csv(file_path)

# Function to calculate average sales per product
def average_sales_per_product(sales_data):
    return sales_data.groupby('product')['sales_amount'].mean()
```

Please provide the complete code, including file path definition and output display.

Out2:



Here's a complete script that uses the provided functions to load a sales dataset and calculate the average sales amount per product. The script includes the necessary imports, defines the file path, and displays the output.

```
python 复制代码

import pandas as pd

# Function to load data from a CSV file
def load_data(file_path):
    return pd.read_csv(file_path)

# Function to calculate average sales per product
def average_sales_per_product(sales_data):
    return sales_data.groupby('product')['sales_amount'].mean()

if __name__ == "__main__":
    # Define the file path to the sales dataset
    file_path = 'path/to/your/sales_data.csv' # Update this path with your CSV file location

    # Load the sales data
    sales_data = load_data(file_path)

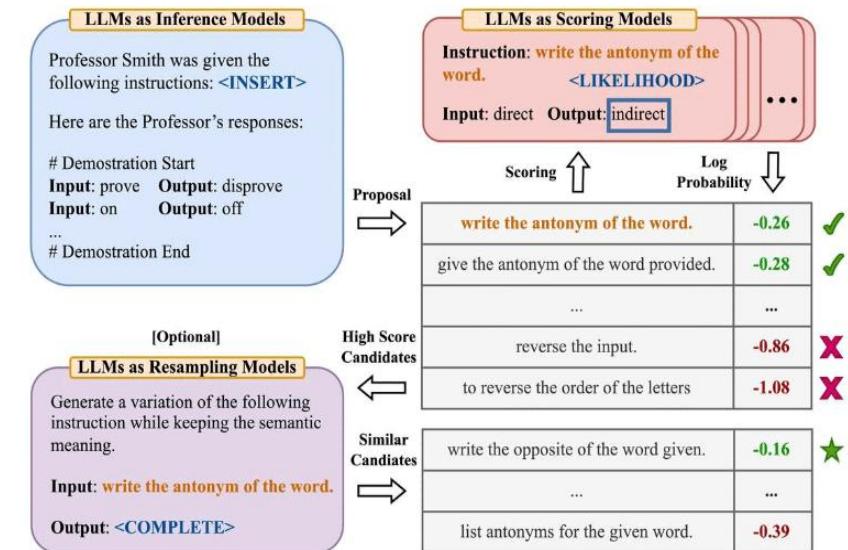
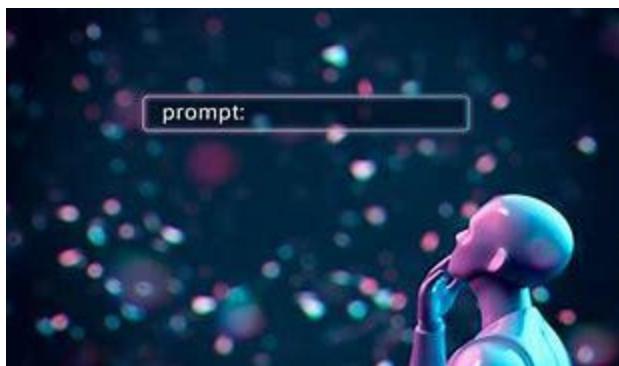
    # Calculate the average sales per product
    average_sales = average_sales_per_product(sales_data)

    # Display the results
    print("Average Sales Amount per Product:")
    print(average_sales)
```



Automatic Prompt Engineer (APE)

Since LLM can do so many things, why not let it help us design Prompt???



Automatic Prompt Engineer (APE)

Zhou et al., (2022) propose automatic prompt engineer (APE) , a framework for **automatic instruction generation** and **selection**. The instruction generation problem is framed as natural language synthesis addressed as a black-box optimization problem **using LLMs** to **generate** and **search** over candidate solutions.

The first step involves a large language model (as an inference model) that is **given output demonstrations** to generate instruction candidates for a task. These candidate solutions will guide the search procedure. The instructions are **executed** using a target model, and then the most appropriate instruction is selected based on computed evaluation scores.

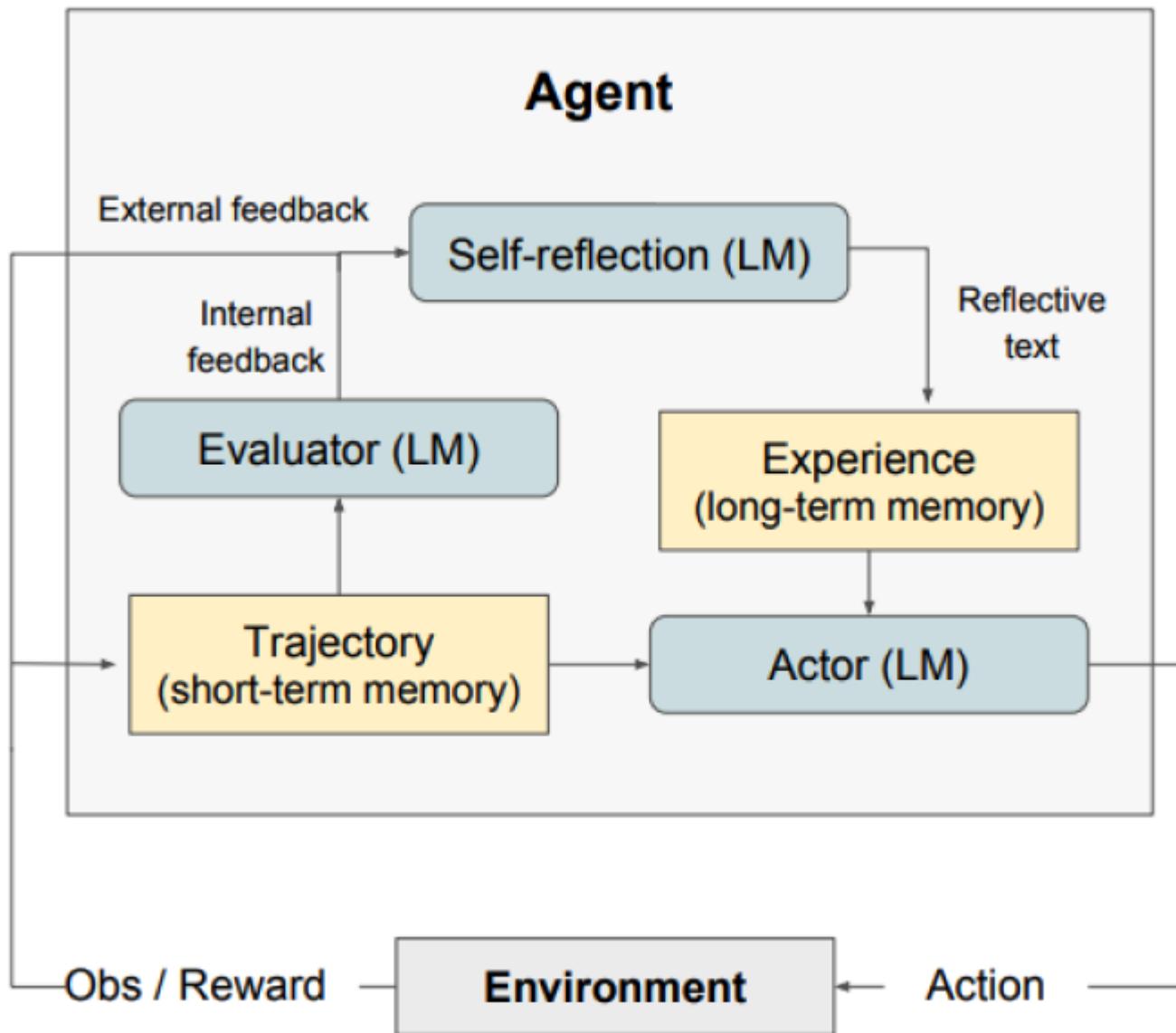
APE discovers a better zero-shot CoT prompt than the human engineered "Let's think step by step" prompt .

The prompt "Let's work this out in a step by step way to be sure we have the right answer." elicits chain-of-thought reasoning and improves performance on the MultiArith and GSM8K benchmarks:

No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	APE	Let's work this out in a step by step way to be sure we have the right answer.	82.0
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4		Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.5
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7
-		(Zero-shot)	17.7

Reflexion

Reflexion is a framework to **reinforce** language-based agents through **linguistic feedback**. Reflexion converts feedback from the environment, also referred to as **self-reflection**, which is provided as context for an LLM agent in the next episode. This helps the agent rapidly and effectively learn from prior mistakes leading to performance improvements on many advanced tasks.



- **An Actor:** Generates text and actions based on the state observations.
- **An Evaluator:** Scores outputs produced by the Actor. Concretely, it takes as input a generated trajectory (also denoted as short-term memory) and outputs a reward score.
- **Self-Reflection:** Generates verbal reinforcement cues to assist the Actor in self-improvement.

(a) Task
 ↓
 (b) Trajectory
 ↓
 (c) Evaluation
 (internal / external)
 ↓
 (d) Reflection
 ↓
 (e) Next Trajectory

1. Decision making

You are in the middle of a room
 [...] **Task:** clean some pan and put it in countertop.

[...]
Action: take pan1 from stoveburner1
Obs: Nothing happens. [...]
Action: clean pan1 with sinkbasin1
Obs: Nothing happens. [...]

Rule/LM Heuristic:

Hallucination.

[...] tried to pick up the pan in stoveburner 1 [...] but the pan was not in stoveburner 1. [...]

[...] **Action:** take pan 1 from stoveburner 2
 [...] **Obs:** You put the pan 1 in countertop 1.

2. Programming

Task: You are given a list of two strings [...] of open '(' or close ')' parentheses only [...]

```
def match_parens(lst):
    if s1.count('(') + s2.count('(') == s1.count(')') + s2.count(')'):
        return 'No'
```

Self-generated unit tests fail:

```
assert match_parens(...)
```

[...] wrong because it only checks if the total count of open and close parentheses is equal [...] order of the parentheses [...]

```
[...]
return 'Yes' if check(S1) or
check(S2) else 'No'
```

3. Reasoning

Task: What profession does John Lanchester and Alan Dean Foster have in common?

Think: [...] novelist, journalist, critic [...] novelist, screenwriter [...] common is novelist and screenwriter.
Action: "novelist, screenwriter"

Environment Binary Reward:

0

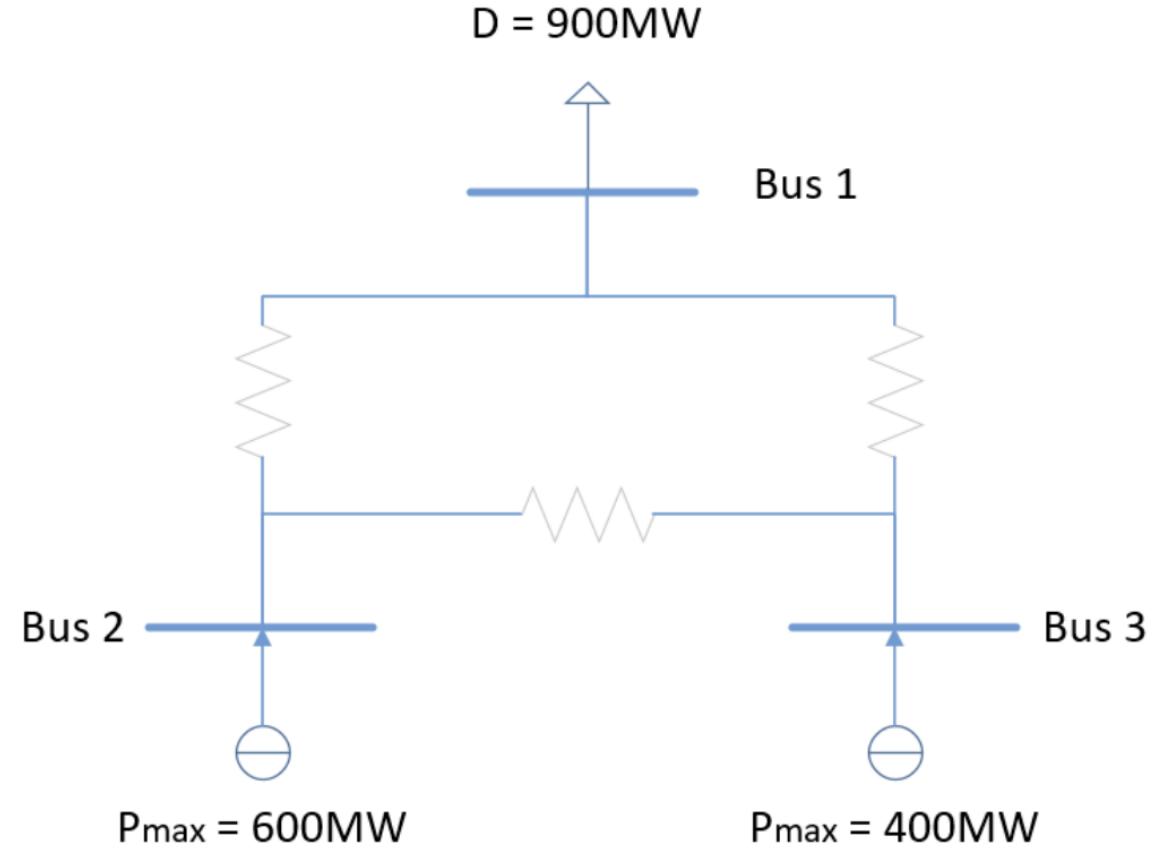
[...] failed because I incorrectly assumed that they both had the same multiple professions [...] accurately identifying their professions.

Think: [...] So the profession John Lanchester and Alan Dean Foster have in common is novelist.
Action: "novelist"

Example: Solving Power Flow Problem

The **power flow problem** (also known as the **load flow problem**) is a fundamental issue in electrical power systems. It involves determining the steady-state voltages, currents, and power flows in an interconnected network of electrical components (such as generators, transmission lines, transformers, and loads). The goal is to analyze the system's behavior under given operating conditions and ensure the power system operates efficiently, reliably, and safely.

Power flow models are mathematical representations used to solve this problem. They typically use a set of nonlinear algebraic equations based on **Kirchhoff's laws**, which describe the relationship between voltages, currents, and powers in the network.



Initial Prompt

You are an experienced power systems engineer and proficient Python programmer. Your task is to develop a Python program that solves a three-bus power flow problem using the Newton-Raphson method. The program should be well-structured, efficient, and include comprehensive comments explaining each part of the code. The program should perform the following tasks:

1. Define the bus data and line data for the system, including bus types (Slack, PV, PQ), specified voltages, power generation, and load values.
2. Compute the bus admittance matrix (Y_{bus}) using the provided line data.
3. Set up initial guesses for voltage magnitudes and angles for all buses except the slack bus.
4. Implement the Newton-Raphson iterative method to solve for the bus voltages and angles, including the calculation of the Jacobian matrix, mismatch vector, and updates to the voltages.
5. Include appropriate convergence criteria (e.g., maximum mismatch below a specified tolerance).
6. After convergence, output the final bus voltages (magnitude and angle in degrees) and power flows on each line.

Use the following system data:

- Bus Data:
 - Bus 1: Slack bus (Type 1), $V = 1.06$ p.u., angle = 0 degrees.
 - Bus 2: PV bus (Type 2), $V = 1.045$ p.u., $P_{gen} = 1.632$ p.u., $Q_{gen} = ?$, $P_{load} = 0$, $Q_{load} = 0$.
 - Bus 3: PQ bus (Type 3), $P_{load} = 0.942$ p.u., $Q_{load} = 0.19$ p.u.
- Line Data:
 - Line 1-2: $R = 0.02$ p.u., $X = 0.06$ p.u., $B = 0$.
 - Line 1-3: $R = 0.08$ p.u., $X = 0.24$ p.u., $B = 0$.
 - Line 2-3: $R = 0.06$ p.u., $X = 0.18$ p.u., $B = 0$.

Additional requirements:

- Organize your code into functions for readability, such as functions to compute Y_{bus} , the mismatch vector, the Jacobian matrix, and the update step.
- Use NumPy for matrix operations.
- Ensure that the program can be easily extended to larger systems.

Provide the complete code below with comments and explanations.

Results

Traceback (most recent call last):

```

File "c:\Users\fuxih\cursor-tutor\power flow Newton.py", line 169, in <module>
    v_final, delta_final = newton_raphson_power_flow(bus_data, line_data)
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "c:\Users\fuxih\cursor-tutor\power flow Newton.py", line 156, in newton_raphson_power_flow
    dx = np.linalg.solve(J, mismatch)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^

File "c:\Users\fuxih\cursor-tutor\.venv\Lib\site-packages\numpy\linalg\_linalg.py", line 413, in solve
    r = gufunc(a, b, signature=signature)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^

File "c:\Users\fuxih\cursor-tutor\.venv\Lib\site-packages\numpy\linalg\_linalg.py", line 104, in _raise_linalgerror_singular
    raise LinAlgError("Singular matrix")
numpy.linalg.LinAlgError: Singular matrix

```

```

import numpy as np

# Constants
MAX_ITER = 100 # Maximum number of iterations
TOL = 1e-6      # Convergence tolerance

# Bus Types
SLACK = 1
PV = 2
PQ = 3

# ----- Bus and Line Data Definition -----
# Bus Data: [Bus Number, Bus Type, V_spec (p.u.), Angle (deg), P_gen (p.u.), Q_gen (p.u.)]

```

```

for line in line_data:
    fb = int(line[0]) - 1 # From bus index
    tb = int(line[1]) - 1 # To bus index
    r = line[2]
    x = line[3]
    b = line[4]
    z = complex(r, x)
    y = 1 / z
    ybus[fb, fb] += y
    ybus[tb, tb] += y
    ybus[fb, tb] -= y
    ybus[tb, fb] -= y
return ybus

```

Second Prompt

As an experienced power systems engineer and proficient Python programmer, your task is to develop a Python program that solves a three-bus power flow problem using the Newton-Raphson method. Employ advanced problem-solving techniques by thoroughly planning, evaluating, and reflecting on your approach to ensure accuracy and efficiency. The program should be well-structured, optimized, and include comprehensive comments explaining each part of the code. Perform the following tasks:

1. Define the bus data and line data for the system, including bus types (Slack, PV, PQ), specified voltages, power generation, and load values.
2. Compute the bus admittance matrix (Y_{bus}) using the provided line data.
3. Set up initial guesses for voltage magnitudes and angles for all buses except the slack bus.
4. Implement the Newton-Raphson iterative method to solve for the bus voltages and angles, including the calculation of the Jacobian matrix, mismatch vector, and updates to the voltages.
5. Include appropriate convergence criteria (e.g., maximum mismatch below a specified tolerance).
6. After convergence, output the final bus voltages (magnitude and angle in degrees) and power flows on each line.

Use the following system data:

- Bus Data:
 - Bus 1: Slack bus (Type 1), $V = 1.06$ p.u., angle = 0 degrees.
 - Bus 2: PV bus (Type 2), $V = 1.045$ p.u., $P_{gen} = 1.632$ p.u., $P_{load} = 0$, $Q_{load} = 0$.
 - Bus 3: PQ bus (Type 3), $P_{load} = 0.942$ p.u., $Q_{load} = 0.19$ p.u.
- Line Data:
 - Line 1-2: $R = 0.02$ p.u., $X = 0.06$ p.u., $B = 0$.
 - Line 1-3: $R = 0.08$ p.u., $X = 0.24$ p.u., $B = 0$.
 - Line 2-3: $R = 0.06$ p.u., $X = 0.18$ p.u., $B = 0$.

Additional requirements:

- Organize your code into functions for readability, such as functions to compute Y_{bus} , the mismatch vector, the Jacobian matrix, and the update step.
- Use NumPy for matrix operations.
- Ensure that the program can be easily extended to larger systems.

Approach the problem methodically:

- Plan your implementation steps carefully before coding.
- At each step, evaluate your code for correctness and efficiency.
- Reflect on potential issues (e.g., singular matrices) and adjust your approach accordingly.

Provide the complete code below with comments and explanations.

Results

```
Traceback (most recent call last):
  File "c:\Users\fuxih\cursor-tutor\power flow Newton.py", line 269, in <module>
    v_final, delta_final = newton_raphson_power_flow(bus_data, line_data)
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\fuxih\cursor-tutor\power flow Newton.py", line 229, in newton_raphson_power_flow
    dx = np.linalg.solve(J, mismatch)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\fuxih\cursor-tutor\.venv\lib\site-packages\numpy\linalg\_linalg.py", line 413, in solve
    r = gufunc(a, b, signature=signature)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: solve1: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (m,m),(m)->(m) (size 4 is different from 3)
```

Final Prompt

As an experienced power systems engineer and proficient Python programmer, your task is to develop a Python program that solves a three-bus power flow problem using the Newton-Raphson method. However, instead of implementing the Newton-Raphson method from scratch, you will utilize an appropriate Python package to perform the power flow calculations. The program should be well-structured, include comprehensive comments, and make effective use of the package's functionalities. The program should perform the following tasks:

1. Define the bus data and line data for the system, including bus types (Slack, PV, PQ), specified voltages, power generation, and load values.
2. Create the power system network model using the selected Python package.
3. Run the power flow analysis using the Newton-Raphson method provided by the package.
4. After convergence, output the final bus voltages (magnitude and angle in degrees) and power flows on each line.

Use the following system data:

- Bus Data:
 - Bus 1: Slack bus (Type 1), $V = 1.06$ p.u., angle = 0 degrees.
 - Bus 2: PV bus (Type 2), $V = 1.045$ p.u., $P_{gen} = 1.632$ p.u., $P_{load} = 0$, $Q_{load} = 0$.
 - Bus 3: PQ bus (Type 3), $P_{load} = 0.942$ p.u., $Q_{load} = 0.19$ p.u.
- Line Data:
 - Line 1-2: $R = 0.02$ p.u., $X = 0.06$ p.u., $B/2 = 0$.
 - Line 1-3: $R = 0.08$ p.u., $X = 0.24$ p.u., $B/2 = 0$.
 - Line 2-3: $R = 0.06$ p.u., $X = 0.18$ p.u., $B/2 = 0$.

Additional requirements:

- Organize your code into functions for readability.
- Use an appropriate Python package such as pandapower or PYPOWER for power flow analysis.
- Ensure that the program can be easily extended to larger systems.

Provide the complete code below with comments and explanations.

Final Results

```

Line Power Flows:
From Bus      To Bus   P_from (MW)    Q_from (MVar)   P_to (MW)    Q_to (MVar)
Bus 1          Bus 2    -92.584678    60.814232     94.768782   -54.261921
Line Power Flows:
From Bus      To Bus   P_from (MW)    Q_from (MVar)   P_to (MW)    Q_to (MVar)
Bus 1          Bus 2    -92.584678    60.814232     94.768782   -54.261921
Bus 1          Bus 3    29.279456     19.578932     -28.396137   -16.928976
Bus 1          Bus 2    -92.584678    60.814232     94.768782   -54.261921
Bus 1          Bus 3    29.279456     19.578932     -28.396137   -16.928976
Bus 1          Bus 3    29.279456     19.578932     -28.396137   -16.928976
Bus 2          Bus 3    68.431218     9.953089      -65.803863   -2.071024
PS C:\Users\fuxih\.cursor-tutor> []

```

python

```

# Import necessary packages
import pandapower as pp
import pandapower.networks as pn

def create_three_bus_network():
    """
    Creates a three-bus power system network using pandapower.
    """

    # Create empty network
    net = pp.create_empty_network()

    # Add buses
    bus1 = net.bus.append(voltage=230, name="Bus 1") # Slack Bus
    bus2 = net.bus.append(voltage=230, name="Bus 2") # PV Bus
    bus3 = net.bus.append(voltage=230, name="Bus 3") # PQ Bus

    # Add slack bus
    net.bus.append(vm_pu=1.06, va_degree=0.0, name="Slack Bus")

    # Add PV generator
    net.gen.append(p_kw=1.632 * 100, vm_pu=1.045, name="PV Generator")

    # Add PQ load
    net.load.append(p_kw=0.942 * 100, q_mvar=0.19 * 100, name="PQ Load")

    # Convert to actual units
    net.base_mva = 100
    net.base_kv = 230
    net.base_ohms = (net.base_kv ** 2) / net.base_mva # Base impedance in Ohms

    # Line impedances in Ohms
    z_line12 = complex(0.02, 0.06) * Z_base
    z_line13 = complex(0.08, 0.24) * Z_base
    z_line23 = complex(0.06, 0.18) * Z_base

    # Create standard line types with calculated impedances
    pp.create_std_type(net, {
        "r_ohm_per_km": z_line12.real,
        "x_ohm_per_km": z_line12.imag,
        "c_nf_per_km": 0.0,
        "max_i_ka": 1.0}, name="Line12_Type", element="line")

```

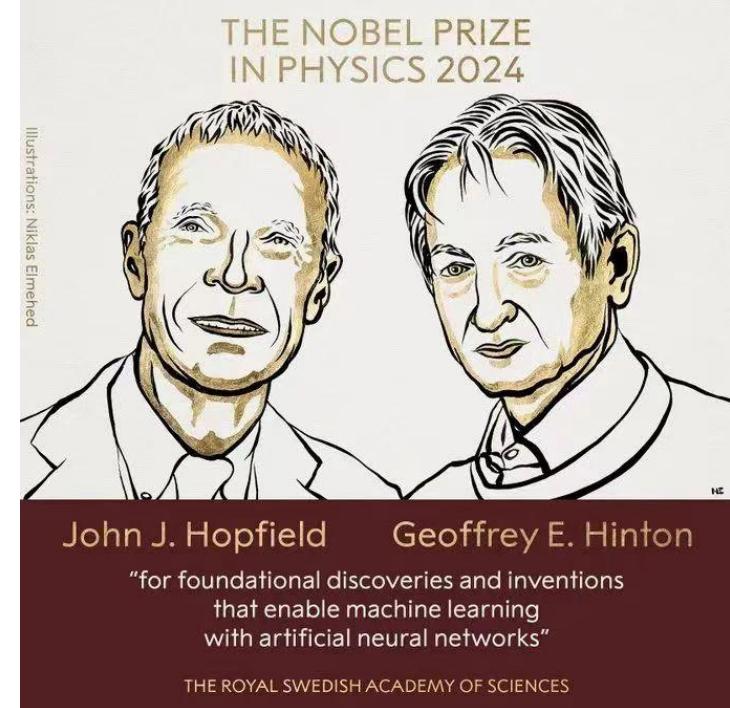
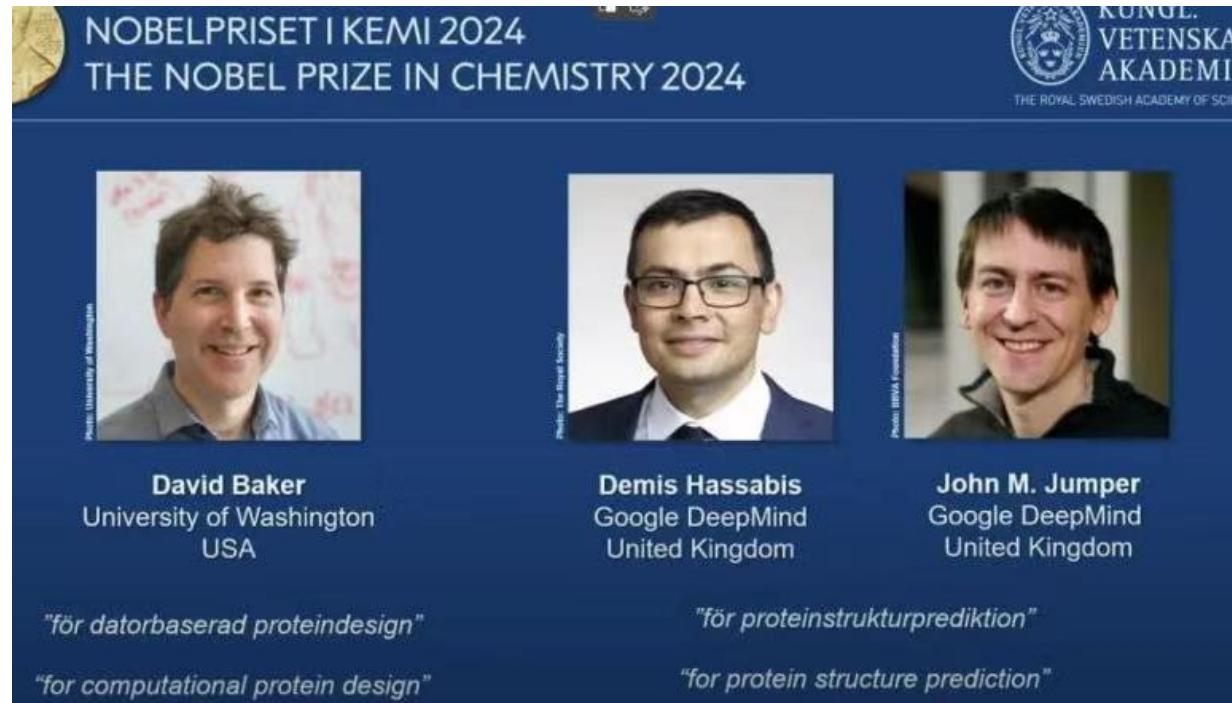
```
pp.create_std_type(net, {
```

Could be useful, could be a joke.

Research claims that LLMs tend to **underperform** in the **afternoon**,
on **weekends**, and during **winter**.

This may be due to lower efficiency in people during these times.
So the model may learn this weakness when time input.

It might be better to inquire in the summer if you're not in a
hurry.....



Is Nobel Prize in Literature possible for Chatgpt?

Do you think mathematicians are safer?

- Initiation Date:** The project, known as the "Equation Theory Project," was launched on September 25, 2024.
- Objective:** The aim was to explore the implications of algebraic structures called magmas and to establish a graph of implications among 4,694 equations.
- Participants:** The project attracted a diverse group of participants, including mathematicians, computer scientists, students, and hobbyists. It utilized various AI tools such as ChatGPT, Claude, and GitHub Copilot.

Achievements

- Number of Proofs:** Within just 57 days, the team successfully processed over 22 million implications, with approximately 8.2 million confirmed as true and about 13.9 million identified as false. Only 162 implications remained unresolved.
- Progress Speed:** By the ninth day after the project's launch, they had achieved an impressive progress rate of 99.866%.

The screenshot shows a Weibo post from the official account of QbitAI (@量子位). The post features a large headline in Chinese: "陶哲轩宣布“等式理论计划”成功，人类AI协作，57天完成2200万+数学关系证明". Below the headline, there is a profile picture of Terence Tao and his name. The post includes a timestamp (2024-11-23 17:41) and a source note (来源: 澎湃新闻·澎湃新闻·澎湃号·湃客). On the left side of the post, there is a vertical column of social media interaction icons (comment, like, heart, message, share).

澎湃新闻

要闻 深度 直播 视频 时事 国际 财经 科技 暖闻 澎湃号 智库 思想 生活 问吧 更多

陶哲轩宣布“等式理论计划”成功，人类AI协作，57天完成2200万+数学关系证明

量子位 | + 关注

2024-11-23 17:41 北京 来源：澎湃新闻·澎湃新闻·澎湃号·湃客

鱼羊一水 发自 凹非寺

量子位 | 公众号 QbitAI

57天，人类和AI合作搞定了4694个等式之间22028942个蕴含关系！

大神陶哲轩激动宣布：等式理论计划，成功。

Terence Tao
@tao@mathstodon.xyz

As of yesterday, the Equational Theories Project teorth.github.io/equational_th... has (provisionally) attained its aim of resolving all 22028942 implications between the 4694 equational laws of magmas involving at most four multiplication operations. We are not fully done yet - there are 52 resolutions that currently only have a human-readable proof but not a Lean-formalized proof (actually, there are a few that are not even human-readable, but are instead computer-generated by some non-Lean program) - but it should now just be a matter of time before we can declare

AI + X is your potential way out...

- Learning only **one professional subject** is not enough in the AI era, whether it is computer, math, finance or any other field
- **AI+X is the future**
- Strive to become an **expert in areas like finance, energy, medicine, law, etc.,** while also mastering AI tools
- **Learning programming is still important,** but don't aim to be just a programmer
- **AI for science (AI4S) is the future if you aspire to become a professor**

