# Methodology: Structured Chain-of-Thought (SCoT) Prompting for Code Generation

This methodology outlines the steps used in the SCoT research paper to improve Large Language Models (LLMs) for code generation. The paper introduces a **Structured Chain-of-Thought (SCoT)** prompting technique to improve Large Language Models (LLMs) in code generation. The approach integrates structured reasoning: IO specification, Sequential, Branch, and Loop constructs; before generating code.

## 1. SCoT Design

SCoT extends standard Chain-of-Thought by using structured programming constructs:  
• IO Structure: Defines input/output types.  
• Sequential Steps: Ordered reasoning steps.  
• Branch Structures: Conditional logic using if/else.  
• Loop Structures: Iterative reasoning with for/while loops.  
Human annotators created 3 demonstration examples manually with these constructs.

## 2. Prompt Construction

Each prompt combines:  
• Instruction: Guide the model to reason with structured steps.  
• Three Demonstration Examples: Function signature, docstring, structured reasoning (comments), and final solution code.  
• Target Task: A new unseen problem with only signature and docstring.

## 3. Datasets and Models

• Datasets: HumanEval (164 Python problems), MBPP (974 Python problems).  
• Models: GPT-4-turbo, GPT-3.5-turbo, DeepSeek Coder-Instruct (1.3B, 6.7B, 33B).  
• Baselines: Zero-shot, Few-shot, and standard Chain-of-Thought prompting.

## 4. Code Generation

• Prompts are constructed with instructions, examples, and new tasks.  
• LLMs generate n = 20 solutions per task using nucleus sampling (temperature = 0.8, top-p = 0.95).  
• Output includes structured reasoning as comments followed by code.

## 5. Evaluation

• Automatic Testing: Generated solutions are run against predefined unit tests. A solution is correct if all tests pass.  
• Pass@k: Calculated as Pass@k = 1 - (n - c choose k)/(n choose k), where n = number of generations, c = correct solutions.  
• Human Evaluation: Ten developers rated 800 samples on correctness (0–2) and code smells (22 bad patterns).

## 6. Robustness and Ablation

• Tested sensitivity to:  
 - Different example seeds, writing styles, and order.  
 - Varying number of examples (1–5).  
• Ablation removed branch/loop reasoning or IO specification to observe Pass@1 impact.

## Steps to Replicate

1. Install openai, datasets, and model APIs.  
2. Download datasets: openai\_humaneval, mbpp.  
3. Manually create 3 demonstration examples with structured reasoning and correct code.  
4. Build prompts as in Figure 3 of the paper.  
5. Use GPT models to generate 20 solutions per task.  
6. Run generated code against test cases.  
7. Compute unbiased Pass@1, Pass@3, Pass@5.  
8. Optionally, perform human evaluation.  
9. Conduct robustness and ablation experiments.