Survey: Research on "Chain-of-Thought (CoT)" Quality in Large Language Model (LLM) Code Generation

Survey Instructions: This survey aims to investigate the quality issues related to CoT encountered by software developers when using reasoning-capable LLMs (such as OpenAI o1, DeepSeek-R1, etc.) for coding assistance. Your feedback will be used for academic research to help improve the reliability of code generation by LLMs. This survey is anonymous and takes about **3-5 minutes** to complete.

Part 1: Background Information

1. **Your software development experience is approximately:**

   [ ] Less than 1 year

   [☑] 1 - 3 years

   [ ] 3 - 5 years

   [ ] More than 5 years

2. **Which reasoning-supported LLMs do you typically use in programming tasks? (Select all that apply)**

   [☑] OpenAI o1 series (o1-preview, o1-mini) [

   [ ] DeepSeek-R1

   [ ] Gemini series (Gemini 2.0 Flash Thinking, Gemini 2.5 pro, Gemini 3 pro, etc.)

   [☑] Claude series (Claude 3.5 Sonnet, Claude 3.7 Sonnet, etc.)

   [ ] Qwen3 series (Qwen3-Max-Thinking-Preview, etc.)

   [ ] Other: _____

3. **When using LLMs to solve complex programming problems, do you read the "Chain-of-Thought/Reasoning Process" generated by the LLM?**

   [ ] Never (I only look at the final generated code) ***(Please stop here, thank you for your participation)***

   [☑] Occasionally

   [ ] Frequently

   [ ] Always

Part 2: Verification of CoT Quality Factors

In our research, we have summarized **5 categories of common CoT quality issues**. Please rate the

frequency with which you encounter these issues based on your actual experience.

*(**Note:** Please refer to "Figure 1: Taxonomy of factors influencing CoT quality" for the classification structure)*

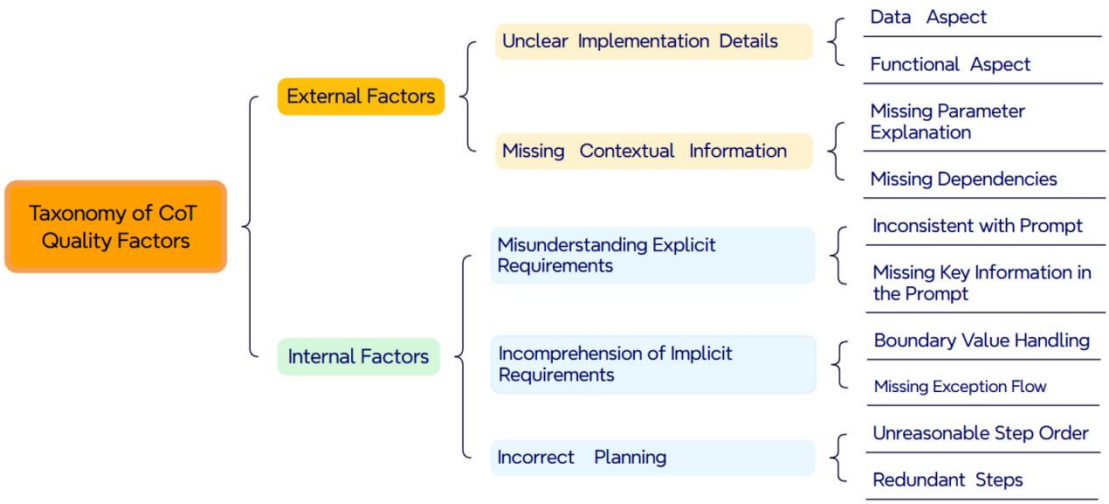

Fig. 1: Taxonomy of factors influencing CoT quality

【External Factors】

4. Unclear Implementation Details

✧ **Data aspect**

- **Description:** Due to a lack of explicit constraints on data structure or data types in the Prompt, the model produces logical deviations during the generation of reasoning steps. *Example:* You asked to "process a set of data" but didn't specify if it was a List or a Set. The CoT proceeded with logic for a List when it should have been a Set.
- **Have you encountered similar situations?**

  [☑] Never encountered

  [ ] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

✧ **Functional aspect**

- **Description:** The Prompt's description of functionality is not precise enough, causing the model to misunderstand the operational logic. *Example:* You asked to "remove separators" but didn't specify "remove all" or "only remove trailing ones." The CoT incorrectly assumed "remove all."
- **Have you encountered similar situations?**

  [ ] Never encountered

[☑] Occasionally encountered

[ ] Frequently encountered

[ ] Extremely frequent

5. Missing Contextual Information

✧ **Missing Parameter Explanation**

- **Description:** The prompt lacks explanations for input parameters in the function signature, causing errors during CoT generation. *Example:* A function has a parameter named flag. The CoT assumed it was a control switch (boolean), but it was actually a bitmask.
- **Have you encountered similar situations?**

  [☑] Never encountered

  [ ] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

✧ **Missing Dependencies**

- **Description:** The task depends on certain external libraries or specific variable names not provided in the Prompt, leading to hallucinations of non-existent variables or dependencies in the reasoning steps. *Example:* The CoT fabricated a function named util.check_valid() to assist with validation during reasoning, but this function does not exist.
- **Have you encountered similar situations?**

  [ ] Never encountered

  [☑] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

【Internal Factors】

6. Misunderstanding Explicit Requirements

✧ **Missing Key Information in the Prompt**

- **Description:** The Prompt clearly stated a key condition, but the CoT failed to extract it during reasoning. *Example:* The Prompt explicitly stated ignore case, but the CoT's reasoning steps did not mention this at all and processed it as case-sensitive.
- **Have you encountered similar situations?**

[☑] Never encountered

[ ] Occasionally encountered

[ ] Frequently encountered

[ ] Extremely frequent

✧ **Inconsistent with Prompt**

- **Description:** The Prompt explicitly stated a requirement, but the reasoning steps conflicted with that requirement.
- **Have you encountered similar situations?**

  [ ] Never encountered

  [☑] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

7. Incomprehension of Implicit Requirements

✧ **Boundary Value Handling**

- **Description:** The CoT only planned for normal cases and ignored boundary cases.
- **Have you encountered similar situations?**

  [ ] Never encountered

  [☑] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

✧ **Missing Exception Flow**

- **Description:** The CoT did not consider potential error scenarios and lacked robustness in design.
- **Have you encountered similar situations?**

  [ ] Never encountered

  [☑] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

8. Incorrect Planning

✧ **Redundant Steps**

- **Description:** The CoT contains completely unnecessary or redundant steps.

- **Have you encountered similar situations?**

  [☑] Never encountered

  [ ] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

✧ **Unreasonable Step Order**

- **Description:** The order of steps in the CoT is incorrect.
- **Have you encountered similar situations?**

  [☑] Never encountered

  [ ] Occasionally encountered

  [ ] Frequently encountered

  [ ] Extremely frequent

Part 3: Impact and Feedback

9. In your experience, when the CoT exhibits the logical defects mentioned above, the final generated code is usually:

[ ] Always incorrect (If CoT is wrong, the code is definitely wrong)

[☑] Frequently incorrect

[ ] Not necessarily (Often the CoT logic is messy, but the model writes the correct code from memory)

[ ] Not really related

10. In your opinion, the capability most lacking in LLMs during code generation is:

[☑] Proactive Clarification (Ability to clarify ambiguous requirements)

[ ] Planning (Complex logical planning ability)

[ ] Robustness (Consideration of boundary/exception cases)

[ ] Self-Correction (Ability to correct its own errors)

11. Have you encountered other types of reasoning errors? (If yes, please briefly describe)

The greatest gap arises from unclear communication: when requirements are not fully articulated, errors are more likely to occur.

None