# The Impact of Prompt Engineering on Code Generation Accuracy and Hallucination Patterns in Language Models


Kadin Matotek

Advisor: Dr. Linh B. Ngo

West Chester University, Department of Computer Science

WCU-AIR Lab


## Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in code generation, yet their reliability remains a critical concern. This study investigates how different prompt engineering strategies affect model accuracy and hallucination patterns across coding tasks.

Research Questions:

- How do concise versus verbose instructions impact code generation accuracy?
- What types of errors and hallucinations emerge across different model sizes?
- How do reasoning strategies (Chain-of-Thought, Direct, Program-Aided) affect performance?

## Methodology

We evaluated 5 Qwen2.5-Coder model variants (0.5B to 14B parameters) on 323 programming problems using a systematic prompt engineering framework.

Dataset:

- **Source:** Kattis competitive programming platform
- Used for rigorous competitions including ICPC
- **Difficulty Range:** 0-2 (on 0-10 Kattis scale)
- Focus on fundamental algorithmic challenges

Experimental Design:

- **Base Instructions:** Concise vs. Verbose (2 options)
- **Reasoning Strategies:** Chain-of-Thought, Direct, Program-Aided (3 options)
- **Problem Decomposition:** None vs. Basic (2 options)
- **Output Formats:** Code only, Explanation + Code, Code + Explanation (3 options)
- **Prompt Variations:** $2 \times 3 \times 2 \times 3 = 36$ unique configurations
- **Total Solutions Generated:** $323$ problems $\times$ $36$ prompts $\times$ $5$ models $= 58,140$ code generations

Evaluation Metrics:

- **Accuracy:** Solutions passing all test cases
- **Error Taxonomy:** 8 hallucination categories, 18 error types
- **Reproducibility:** Dual independent runs

```
base_instructions:
  Verbose: |
    You are a Python programming expert who writes clean, efficient code for competitive programming-style
    problems. When given a problem statement and test cases, produce a single Python script that:
    1. Uses only the Python standard library (no external imports).
    2. Reads input silently from stdin using input() without any prompts or additional text.
    3. Chooses descriptive, non-conflicting variable and function names.
    4. Correctly handles edge cases (empty inputs, minimum/maximum values, etc.).
    5. Does not hard-code any test-specific values (your solution must generalize).
    6. Make sure to print the result and nothing else besides the result!

  Concise: |
    You are a Python programming expert. Solve the following problem.
    Provide only Python code that reads from stdin and prints the answer.
```

Figure 1. Comparison of base instruction styles: verbose prompts (top) provide detailed explanations while concise prompts (bottom) use minimal, direct language.

## Key Finding: Conciseness Advantage

Across all 18 prompt configuration comparisons, **concise instructions outperformed verbose ones in 94.4%** of cases with an average improvement of +1.35%.

This effect was most pronounced in smaller models:

- **0.5B model:** +21.0% relative improvement
- **1.5B, 3B models:** +8.2% and +8.3% improvement
- **7B, 14B models:** +5.7% and +0.4% improvement

This suggests verbose instructions introduce noise that smaller models struggle to filter, while larger models can better extract relevant information despite verbosity.

## Results: Prompt Strategy Performance

Top 3 Configurations

| Base | Reasoning | Decomp | Output | Acc. |
|---|---|---|---|---|
| Concise | Direct | None | Code only | **33.10%** |
| Concise | Direct | None | Exp + Code | 32.79% |
| Concise | CoT | None | Exp + Code | 32.04% |

Bottom 3 Configurations

| Base | Reasoning | Decomp | Output | Acc. |
|---|---|---|---|---|
| Verbose | CoT | Basic | Code + Exp | 28.17% |
| Verbose | PAL | Basic | Code + Exp | 27.80% |
| Verbose | PAL | Basic | Exp + Code | **26.78%** |

*Max gap:* ***+6.32%***

## Reasoning Strategy Comparison

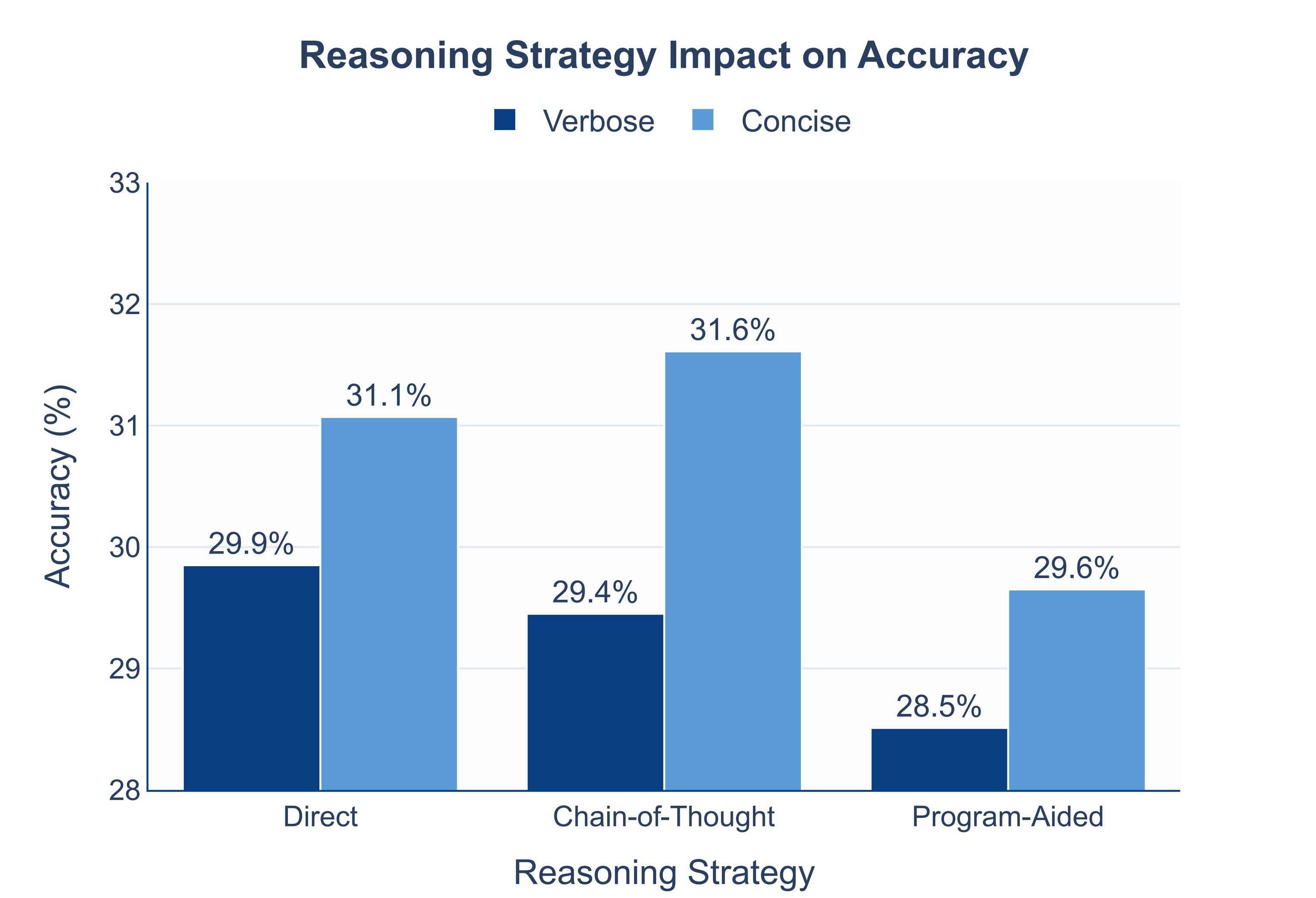


Figure 2. Concise prompts consistently outperform verbose across all reasoning strategies (Direct, Chain-of-Thought, and Program-Aided).

## Error and Hallucination Analysis

We manually defined and annotated 28 distinct error types. Crucially, we observe:

**Highly reproducible hallucination profiles:** Independent runs (v1 vs v2) produce nearly identical error distributions (overlapping solid/dashed lines).

**Systematic shift with model scale:** Larger models dramatically reduce the dominant error mode while maintaining the same overall failure pattern.

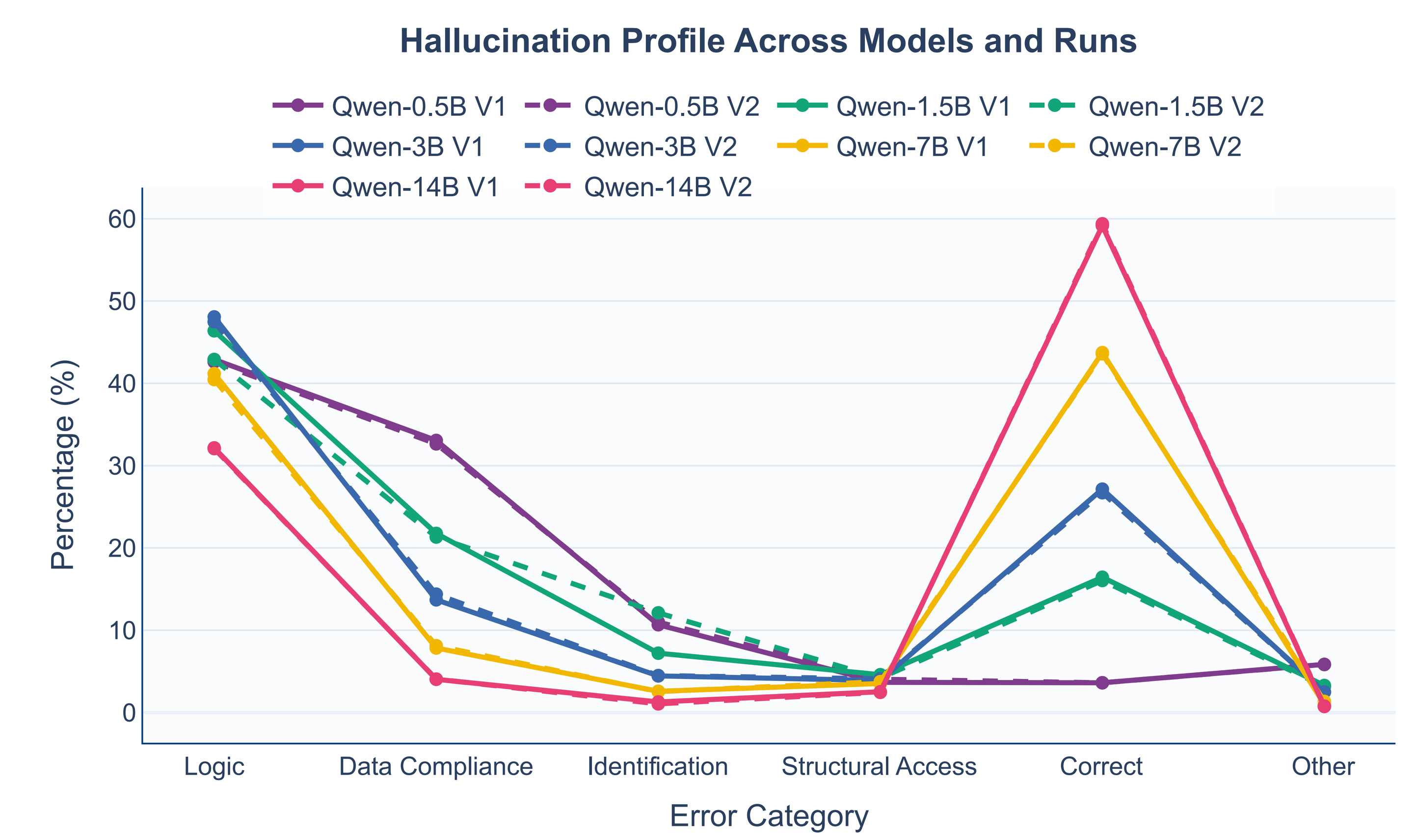


Figure 3. **Hallucination profiles are remarkably stable across independent runs** (solid vs dashed lines nearly overlap) but **shift predictably with model size**.

Most Common Error Types:

- **Logic Deviation** (41.6%): Incorrect algorithmic approach
- **ValueError** (12.9%): Invalid input handling
- **TypeError** (2.9%): Type mismatches
- **IndexError** (3.1%): Array boundary violations
- **NameError** (2.8%): Undefined variables

## Conclusions

Key Takeaways:

- Concise prompts consistently outperform verbose alternatives
- Smaller models are more sensitive to prompt engineering
- Logic deviations are the dominant failure mode
- Reasoning strategy and output format significantly affect accuracy

Future Directions:

- Extend evaluation to more complex problems (difficulty 3-10)
- Test multilingual code generation across Python, Java, C++
- Investigate few-shot prompting with example solutions
- Develop automated prompt optimization frameworks