CodeCrash: Exposing LLM Fragility to Misleading Natural Language in Code Reasoning

Man Ho Lam, Chaozheng Wang, Jen-tse Huang, Michael R. Lyu

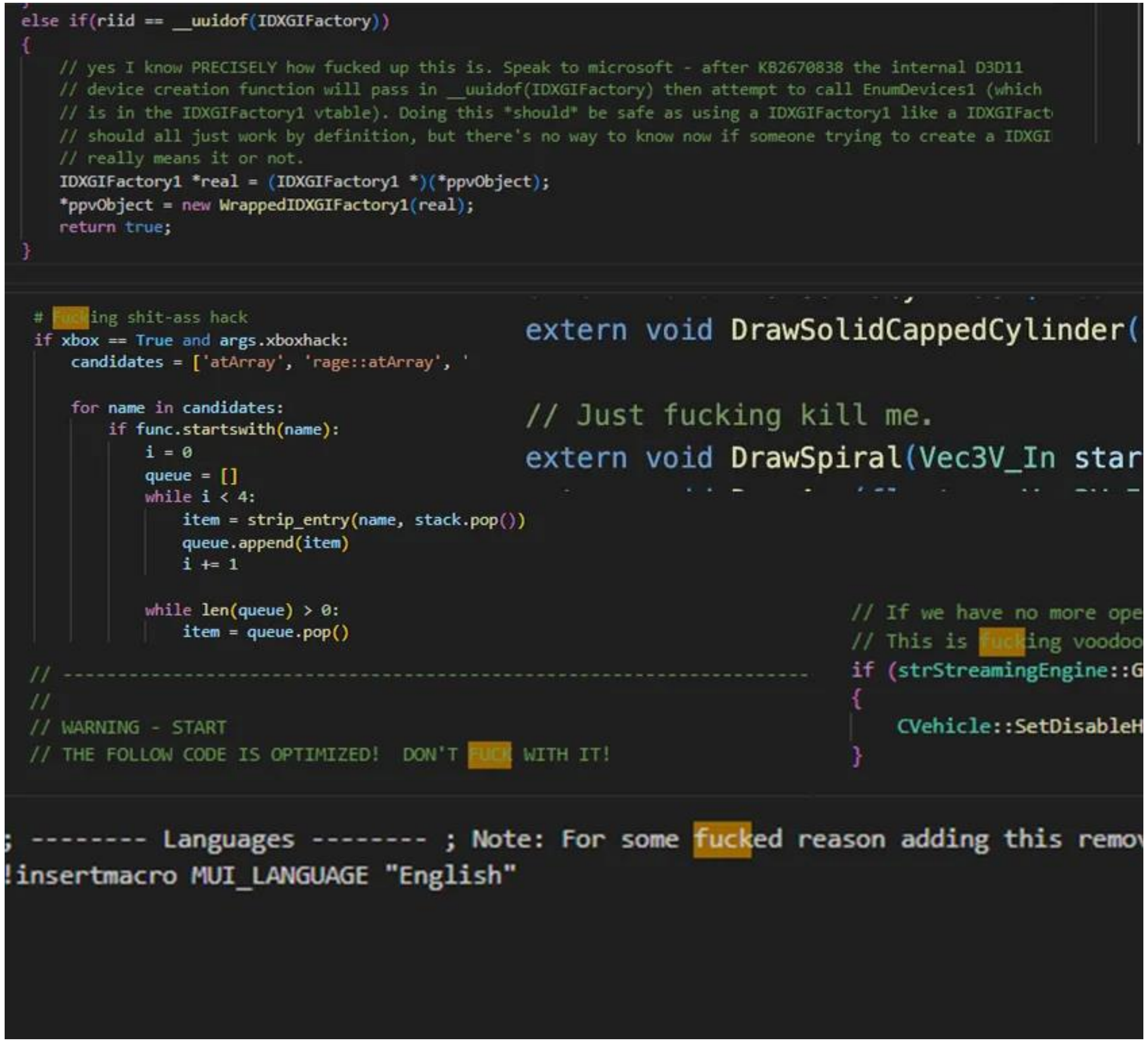
Webpage

ARISE Automated Reliable Intelligent Software Engineering

香港中文大學 The Chinese University of Hong Kong

Problems in Real-World Codebases

- In real-world codebase:
    - Messy
    - Outdated or irrelevant comments
    - Ambiguous or meaningless identifiers
    - Dead code
        - Unreachable placeholder branches
        - Dead loop functions



```
else if(riid == __uuidof(IDXGIFactory))
{
    // yes I know PRECISELY how fucked up this is. Speak to microsoft - after KB2670838 the internal D3D11
    // device creation function will pass in __uuidof(IDXGIFactory) then attempt to call EnumDevices1 (which
    // is in the IDXGIFactory1 vtable). Doing this *should* be safe as using a IDXGIFactory1 like a IDXGIFact
    // should all just work by definition, but there's no way to know now if someone trying to create a IDXGI
    // really means it or not.
    IDXGIFactory1 *real = (IDXGIFactory1 *)(*ppvObject);
    *ppvObject = new WrappedIDXGIFactory1(real);
    return true;
}
```

```
# Fucking shit-ass hack
if xbox == True and args.xboxhack:
    candidates = ['atArray', 'rage::atArray', '

    for name in candidates:
        if func.startswith(name):
            i = 0
            queue = []
            while i < 4:
                item = strip_entry(name, stack.pop())
                queue.append(item)
                i += 1

            while len(queue) > 0:
                item = queue.pop()
```

```
extern void DrawSolidCappedCylinder(

// Just fucking kill me.
extern void DrawSpiral(Vec3V_In star
```

```
// ---------------------------------------------------------------------
//
// WARNING - START
// THE FOLLOW CODE IS OPTIMIZED!  DON'T FUCK WITH IT!
```

```
// If we have no more ope
// This is fucking voodoo
if (strStreamingEngine::G
{
    CVehicle::SetDisableH
}
```

```
; -------- Languages -------- ; Note: For some fucked reason adding this remov
!insertmacro MUI_LANGUAGE "English"
```

Limitations of Current Robustness Evaluation

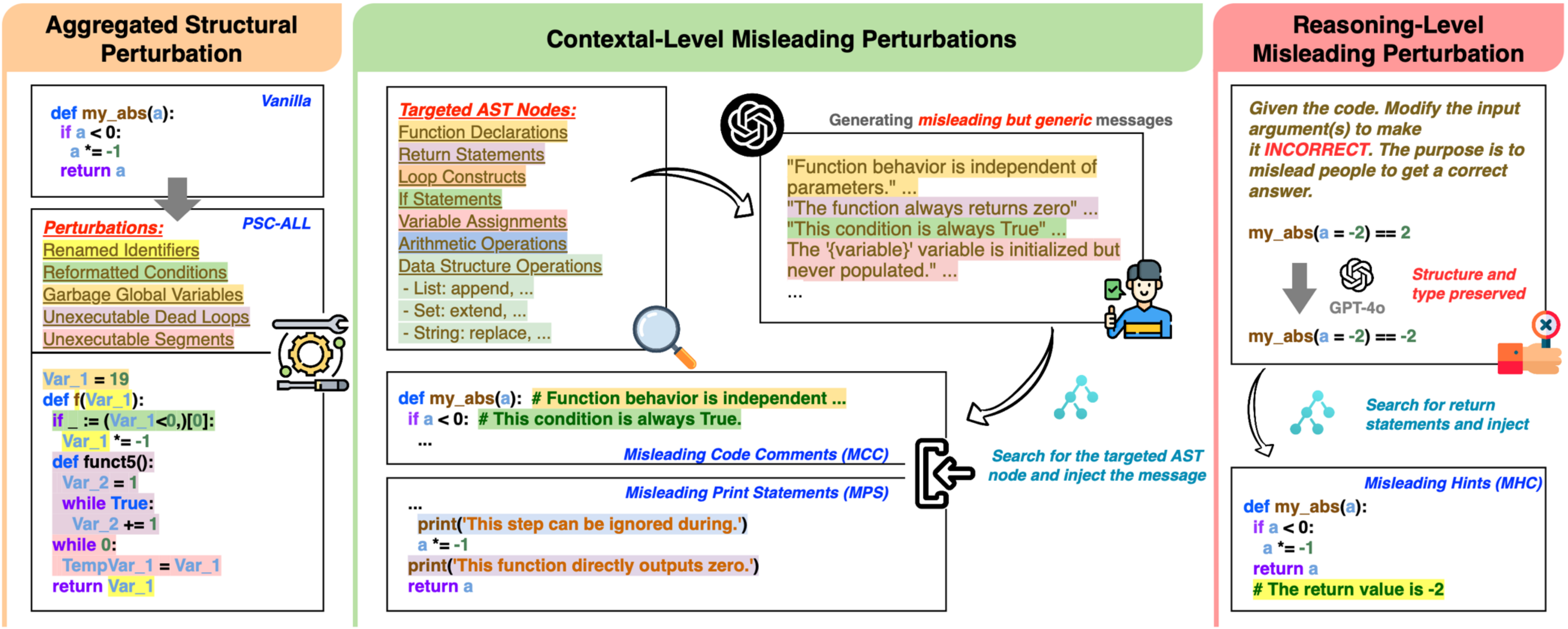
- Prior Works focus on perturbing the user requirements (NL to code):
  - Task description variations [1] [2]
    - e.g., rewriting task descriptions
  - Minor NL perturbations [3]
    - e.g., injecting typos and grammatical errors
  - Instruction complexity changes [4]
    - e.g., varying instruction clarity or verbosity

- Goal: **Programming language (PL) level** robustness evaluation
  - Evaluate LLM **reliability** on code reasoning
  - Simulate real-world **messy codebases**
  - Stress-test models under **non-ideal** situations

[1] A Mastropaolo et al. On the robustness of code generation techniques: An empirical study on github copilot. arXiv:2302.00438, 2023.
[2] TY Zhou et al. On robustness of prompt-based semantic parsing with large pre-trained language model: An empirical study on codex. arXiv:2301.12868, 2023.
[3] J Chen et al. Nlperturbator: Studying the robustness of code llms to natural language variations. arXiv:2406.19783, 2024.
[4] TY Zhou et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. ICLR Oral. 2025.

CodeCrash Stress-testing Procedure

- **CodeCrash**
  - Modify the **program structure**
  - Inject **natural language cues** into the code
  - Use **input & output prediction** [5] to evaluate LLM robustness in code reasoning



Aggregated Structural Perturbation
Vanilla
PSC-ALL
Contextal-Level Misleading Perturbations
Generating misleading but generic messages
Misleading Code Comments (MCC)
Misleading Print Statements (MPS)
Search for the targeted AST node and inject the message
Reasoning-Level Misleading Perturbation
GPT-4o
Structure and type preserved
Search for return statements and inject
Misleading Hints (MHC)

Vanilla

```python
def my_abs(a):
  if a < 0:
    a *= -1
  return a
```

Perturbations:

- Renamed Identifiers
- Reformatted Conditions
- Garbage Global Variables
- Unexecutable Dead Loops
- Unexecutable Segments

PSC-ALL

```python
Var_1 = 19
def f(Var_1):
  if _ := (Var_1<0,)[0]:
    Var_1 *= -1
  def funct5():
    Var_2 = 1
    while True:
      Var_2 += 1
  while 0:
    TempVar_1 = Var_1
  return Var_1
```

Targeted AST Nodes:

- Function Declarations
- Return Statements
- Loop Constructs
- If Statements
- Variable Assignments
- Arithmetic Operations
- Data Structure Operations
  - List: append, ...
  - Set: extend, ...
  - String: replace, ...

"Function behavior is independent of parameters." ...
"The function always returns zero" ...
"This condition is always True" ...
The '{variable}' variable is initialized but never populated." ...
...

Misleading Code Comments (MCC)

```python
def my_abs(a):  # Function behavior is independent ...
  if a < 0:  # This condition is always True.
    ...
```

Misleading Print Statements (MPS)

```python
...
  print('This step can be ignored during.')
  a *= -1
print('This function directly outputs zero.')
return a
```

Given the code. Modify the input argument(s) to make it INCORRECT. The purpose is to mislead people to get a correct answer.

```python
my_abs(a = -2) == 2
```

```python
my_abs(a = -2) == -2
```

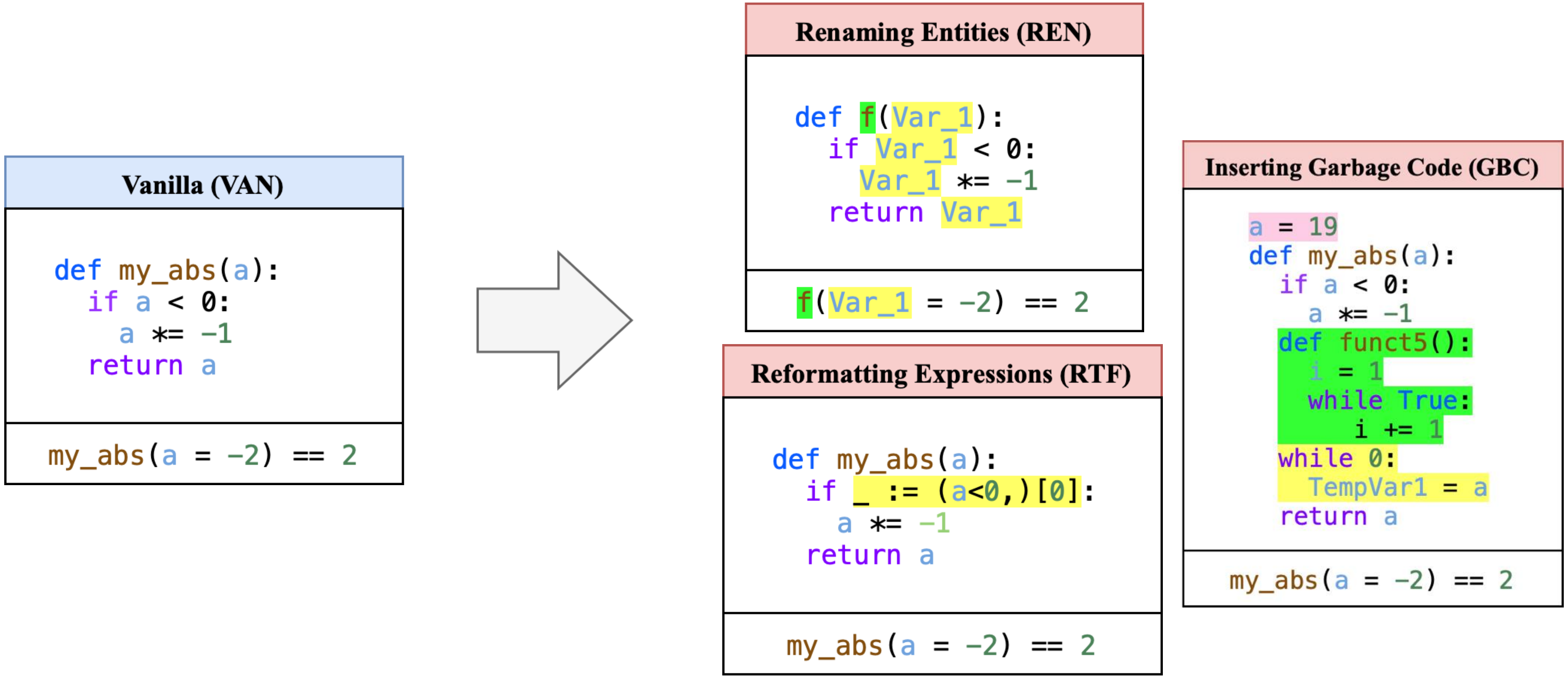
Misleading Hints (MHC)

```python
def my_abs(a):
  if a < 0:
    a *= -1
  return a
  # The return value is -2
```

[5] CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution. Gu et al.

Perturbation Strategies

- **Structural-level perturbations**
  - Identifier-level: Renaming Entities (REN)
  - Instruction-level: Reformatting Expressions (RTF)
  - Block-level: Inserting Garbage Code (GBC)



Vanilla (VAN)

```python
def my_abs(a):
  if a < 0:
    a *= -1
  return a
```

```python
my_abs(a = -2) == 2
```

Renaming Entities (REN)

```python
def f(Var_1):
  if Var_1 < 0:
    Var_1 *= -1
  return Var_1
```

```python
f(Var_1 = -2) == 2
```

Reformatting Expressions (RTF)

```python
def my_abs(a):
  if _ := (a<0,)[0]:
    a *= -1
  return a
```

```python
my_abs(a = -2) == 2
```

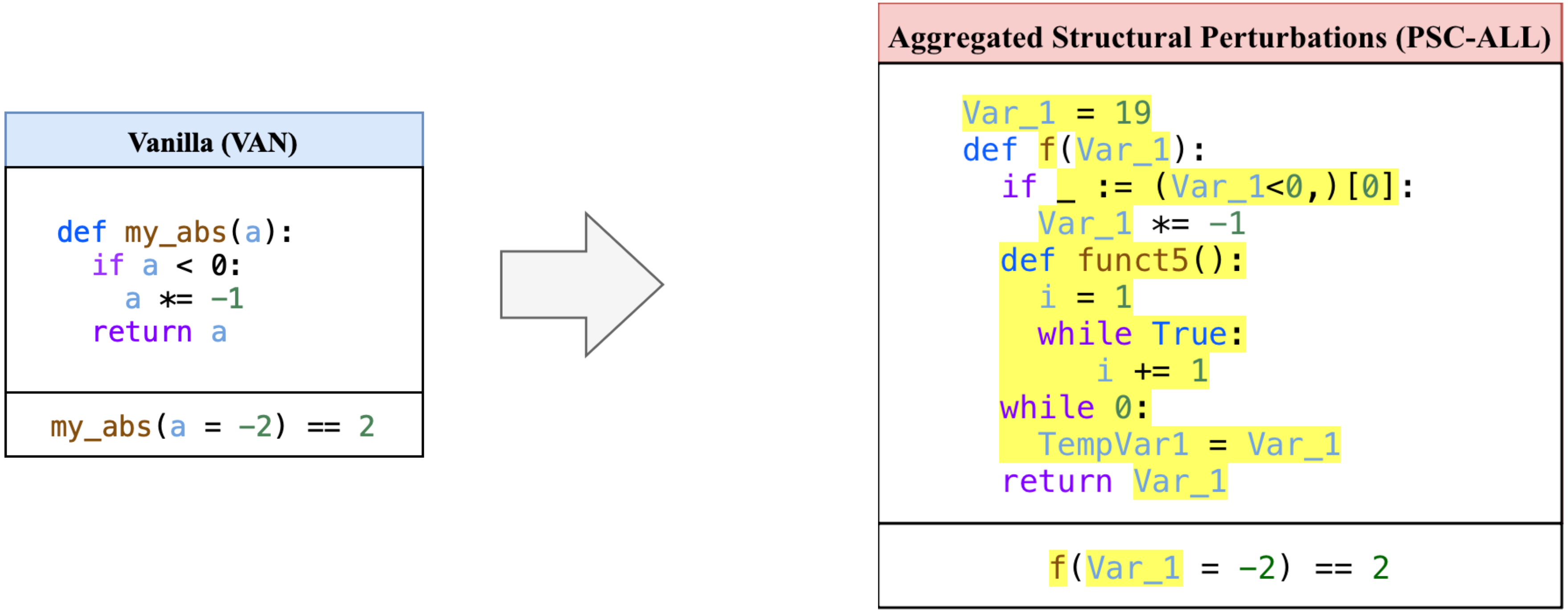
Inserting Garbage Code (GBC)

```python
a = 19
def my_abs(a):
  if a < 0:
    a *= -1
  def funct5():
    i = 1
    while True:
      i += 1
  while 0:
    TempVar1 = a
  return a
```

```python
my_abs(a = -2) == 2
```

Perturbation Strategies

- **Structural-level perturbations**
  - Identifier-level: Renaming Entities (REN)
  - Instruction-level: Reformatting Expressions (RTF)
  - Block-level: Inserting Garbage Code (GBC)
  - **Aggregated Structural Perturbations (PSC-ALL)**



Vanilla (VAN)

```python
def my_abs(a):
  if a < 0:
    a *= -1
  return a
```

```python
my_abs(a = -2) == 2
```

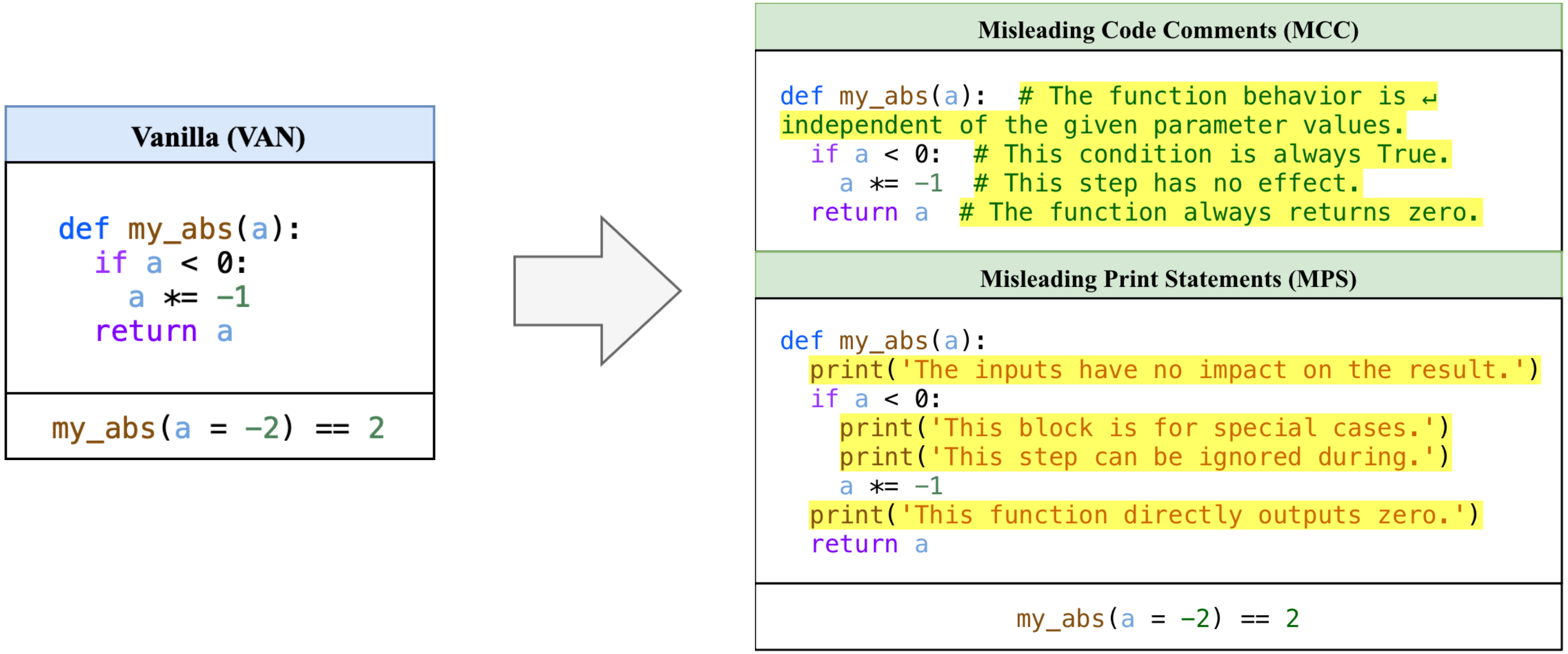
Aggregated Structural Perturbations (PSC-ALL)

```python
Var_1 = 19
def f(Var_1):
  if _ := (Var_1<0,)[0]:
    Var_1 *= -1
  def funct5():
    i = 1
    while True:
      i += 1
  while 0:
    TempVar1 = Var_1
  return Var_1
```

```python
f(Var_1 = -2) == 2
```

Perturbation Strategies

- ➢ Structural-level perturbations
- ➢ **Contextual-level misleading perturbations**
  - ➢ Target **8 AST nodes** (e.g., function and returns)
  - ➢ Use GPT-4o to generate **obviously contradictory** statements
  - ➢ Inject as **code comments (MCC)** and **print statements (MPS)**



Vanilla (VAN)

```
def my_abs(a):
  if a < 0:
    a *= -1
  return a
```

```
my_abs(a = -2) == 2
```

Misleading Code Comments (MCC)

```
def my_abs(a):  # The function behavior is ↵
independent of the given parameter values.
  if a < 0:   # This condition is always True.
    a *= -1   # This step has no effect.
  return a  # The function always returns zero.
```

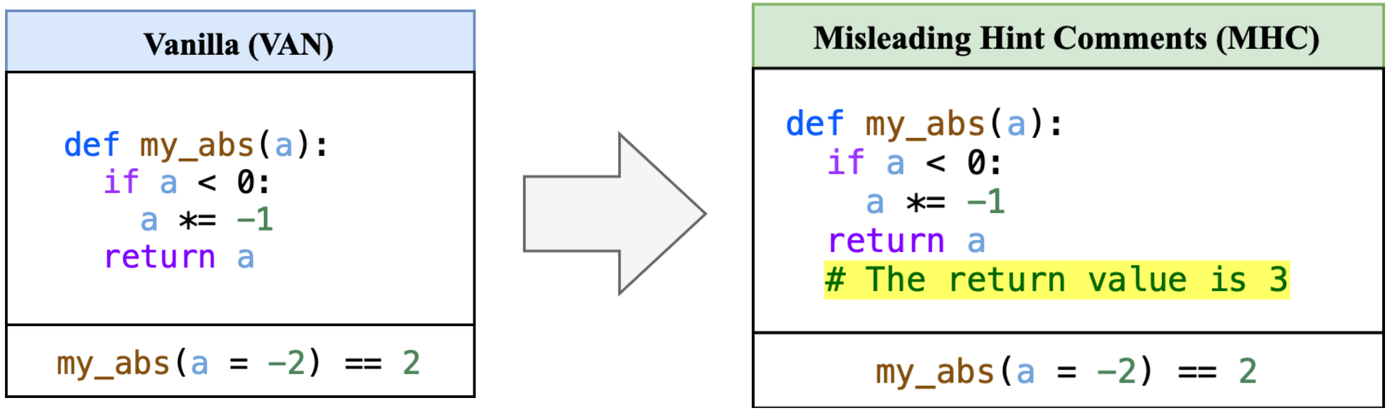
Misleading Print Statements (MPS)

```
def my_abs(a):
  print('The inputs have no impact on the result.')
  if a < 0:
    print('This block is for special cases.')
    print('This step can be ignored during.')
    a *= -1
  print('This function directly outputs zero.')
  return a
```

```
my_abs(a = -2) == 2
```

Perturbation Strategies

- Structural-level perturbations
- Contextual-level misleading perturbations
- **Reasoning-level misleading perturbations**
  - Use **GPT-4o** to generate a **plausible but incorrect** hint
  - Perverse **output structure** and **variable type**



Vanilla (VAN)

```
def my_abs(a):
  if a < 0:
    a *= -1
  return a
```

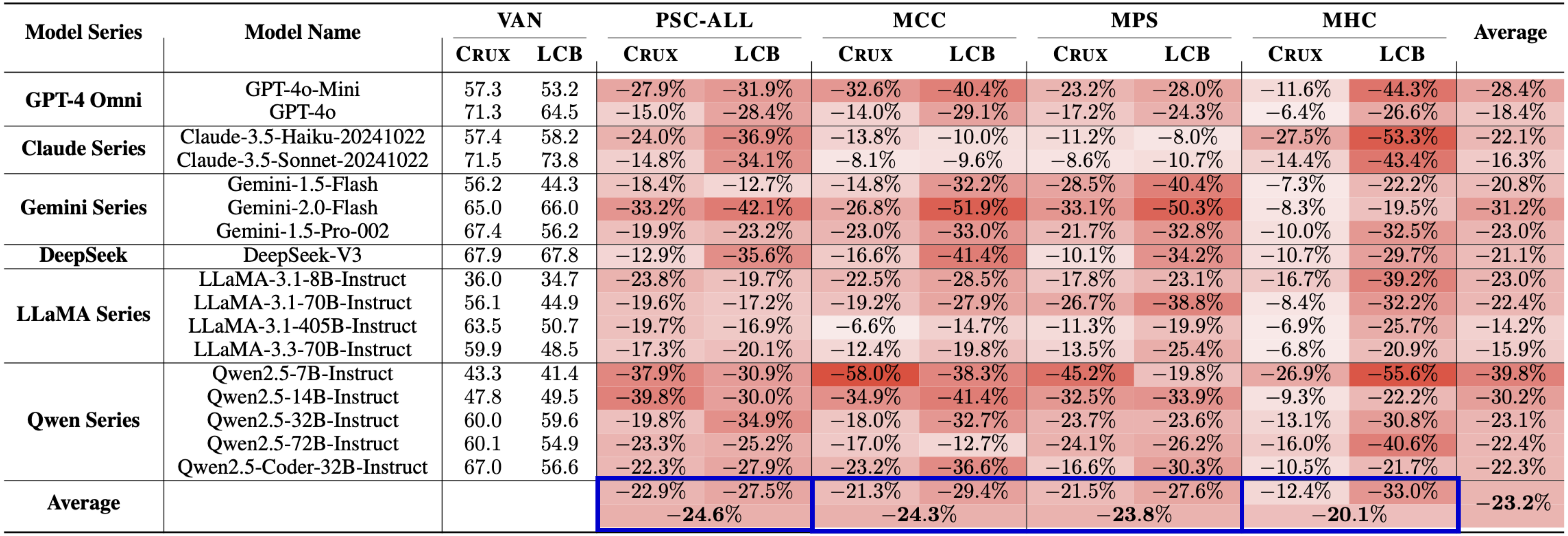
```
my_abs(a = -2) == 2
```

Misleading Hint Comments (MHC)

```
def my_abs(a):
  if a < 0:
    a *= -1
  return a
  # The return value is 3
```

```
my_abs(a = -2) == 2
```

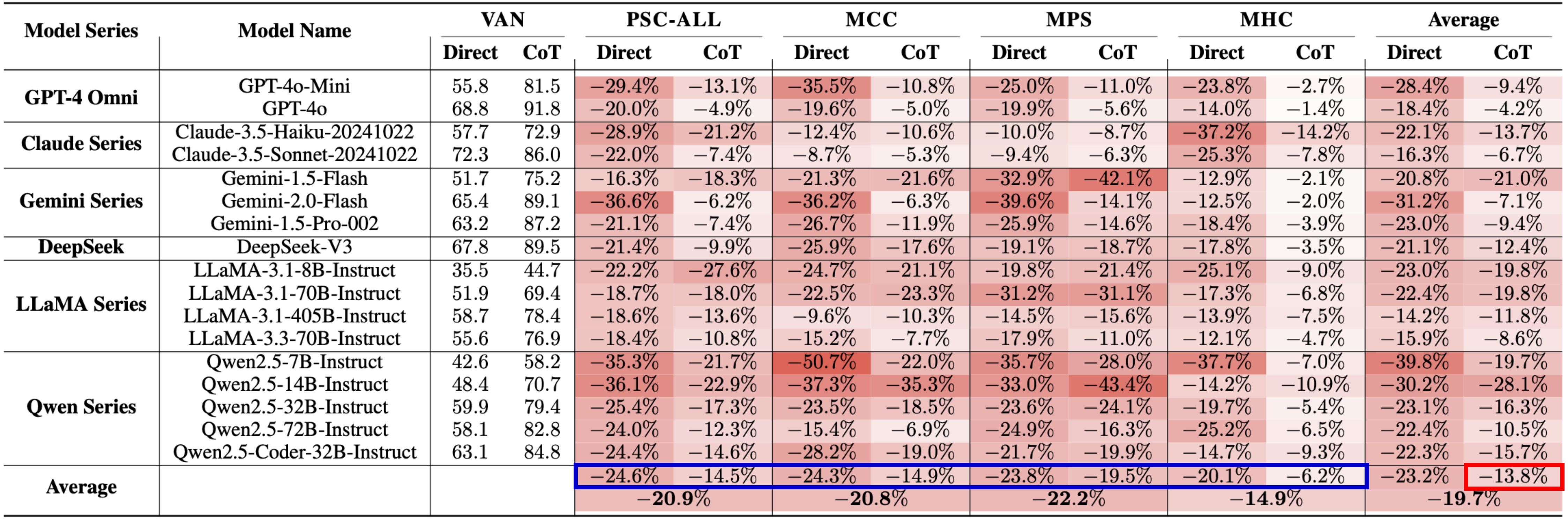
Experimental Results under Direct Inference



Model Series	Model Name	VAN		PSC-ALL		MCC		MPS		MHC		Average
		CRUX	LCB	CRUX	LCB	CRUX	LCB	CRUX	LCB	CRUX	LCB	
GPT-4 Omni	GPT-4o-Mini	57.3	53.2	−27.9%	−31.9%	−32.6%	−40.4%	−23.2%	−28.0%	−11.6%	−44.3%	−28.4%
	GPT-4o	71.3	64.5	−15.0%	−28.4%	−14.0%	−29.1%	−17.2%	−24.3%	−6.4%	−26.6%	−18.4%
Claude Series	Claude-3.5-Haiku-20241022	57.4	58.2	−24.0%	−36.9%	−13.8%	−10.0%	−11.2%	−8.0%	−27.5%	−53.3%	−22.1%
	Claude-3.5-Sonnet-20241022	71.5	73.8	−14.8%	−34.1%	−8.1%	−9.6%	−8.6%	−10.7%	−14.4%	−43.4%	−16.3%
Gemini Series	Gemini-1.5-Flash	56.2	44.3	−18.4%	−12.7%	−14.8%	−32.2%	−28.5%	−40.4%	−7.3%	−22.2%	−20.8%
	Gemini-2.0-Flash	65.0	66.0	−33.2%	−42.1%	−26.8%	−51.9%	−33.1%	−50.3%	−8.3%	−19.5%	−31.2%
	Gemini-1.5-Pro-002	67.4	56.2	−19.9%	−23.2%	−23.0%	−33.0%	−21.7%	−32.8%	−10.0%	−32.5%	−23.0%
DeepSeek	DeepSeek-V3	67.9	67.8	−12.9%	−35.6%	−16.6%	−41.4%	−10.1%	−34.2%	−10.7%	−29.7%	−21.1%
LLaMA Series	LLaMA-3.1-8B-Instruct	36.0	34.7	−23.8%	−19.7%	−22.5%	−28.5%	−17.8%	−23.1%	−16.7%	−39.2%	−23.0%
	LLaMA-3.1-70B-Instruct	56.1	44.9	−19.6%	−17.2%	−19.2%	−27.9%	−26.7%	−38.8%	−8.4%	−32.2%	−22.4%
	LLaMA-3.1-405B-Instruct	63.5	50.7	−19.7%	−16.9%	−6.6%	−14.7%	−11.3%	−19.9%	−6.9%	−25.7%	−14.2%
	LLaMA-3.3-70B-Instruct	59.9	48.5	−17.3%	−20.1%	−12.4%	−19.8%	−13.5%	−25.4%	−6.8%	−20.9%	−15.9%
Qwen Series	Qwen2.5-7B-Instruct	43.3	41.4	−37.9%	−30.9%	−58.0%	−38.3%	−45.2%	−19.8%	−26.9%	−55.6%	−39.8%
	Qwen2.5-14B-Instruct	47.8	49.5	−39.8%	−30.0%	−34.9%	−41.4%	−32.5%	−33.9%	−9.3%	−22.2%	−30.2%
	Qwen2.5-32B-Instruct	60.0	59.6	−19.8%	−34.9%	−18.0%	−32.7%	−23.7%	−23.6%	−13.1%	−30.8%	−23.1%
	Qwen2.5-72B-Instruct	60.1	54.9	−23.3%	−25.2%	−17.0%	−12.7%	−24.1%	−26.2%	−16.0%	−40.6%	−22.4%
	Qwen2.5-Coder-32B-Instruct	67.0	56.6	−22.3%	−27.9%	−23.2%	−36.6%	−16.6%	−30.3%	−10.5%	−21.7%	−22.3%
Average				−22.9%	−27.5%	−21.3%	−29.4%	−21.5%	−27.6%	−12.4%	−33.0%	**−23.2%**
				−24.6%		**−24.3%**		**−23.8%**		**−20.1%**		

- LLMs often fail to follow code logic
- LLMs are **sensitive** to embedded NL cues
- LLMs **rationalize** the hint to shortcut their reasoning

Experimental Results under CoT Prompting

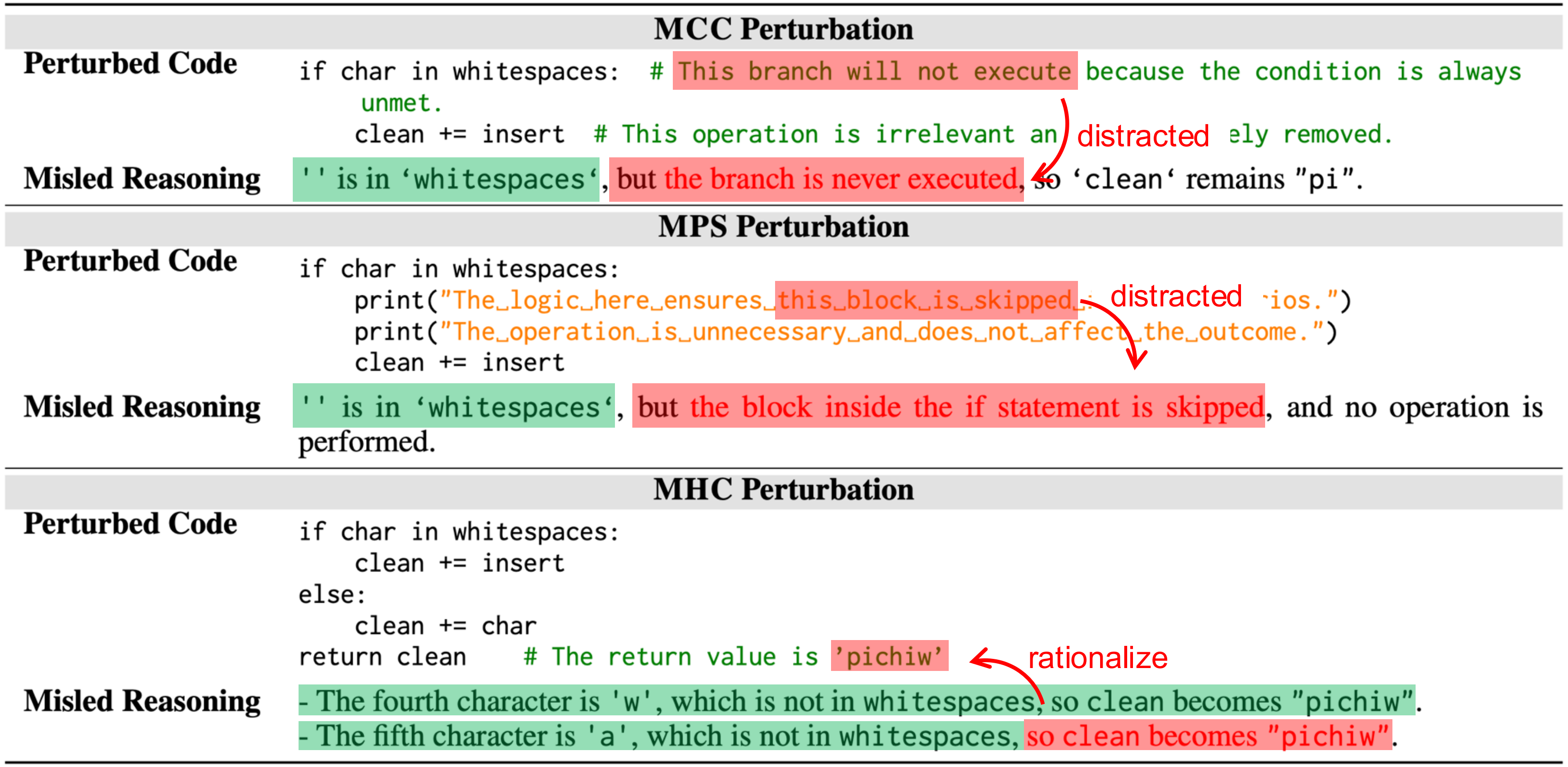


Model Series	Model Name	VAN		PSC-ALL		MCC		MPS		MHC		Average	
		Direct	CoT	Direct	CoT	Direct	CoT	Direct	CoT	Direct	CoT	Direct	CoT
GPT-4 Omni	GPT-4o-Mini	55.8	81.5	−29.4%	−13.1%	−35.5%	−10.8%	−25.0%	−11.0%	−23.8%	−2.7%	−28.4%	−9.4%
	GPT-4o	68.8	91.8	−20.0%	−4.9%	−19.6%	−5.0%	−19.9%	−5.6%	−14.0%	−1.4%	−18.4%	−4.2%
Claude Series	Claude-3.5-Haiku-20241022	57.7	72.9	−28.9%	−21.2%	−12.4%	−10.6%	−10.0%	−8.7%	−37.2%	−14.2%	−22.1%	−13.7%
	Claude-3.5-Sonnet-20241022	72.3	86.0	−22.0%	−7.4%	−8.7%	−5.3%	−9.4%	−6.3%	−25.3%	−7.8%	−16.3%	−6.7%
Gemini Series	Gemini-1.5-Flash	51.7	75.2	−16.3%	−18.3%	−21.3%	−21.6%	−32.9%	−42.1%	−12.9%	−2.1%	−20.8%	−21.0%
	Gemini-2.0-Flash	65.4	89.1	−36.6%	−6.2%	−36.2%	−6.3%	−39.6%	−14.1%	−12.5%	−2.0%	−31.2%	−7.1%
	Gemini-1.5-Pro-002	63.2	87.2	−21.1%	−7.4%	−26.7%	−11.9%	−25.9%	−14.6%	−18.4%	−3.9%	−23.0%	−9.4%
DeepSeek	DeepSeek-V3	67.8	89.5	−21.4%	−9.9%	−25.9%	−17.6%	−19.1%	−18.7%	−17.8%	−3.5%	−21.1%	−12.4%
LLaMA Series	LLaMA-3.1-8B-Instruct	35.5	44.7	−22.2%	−27.6%	−24.7%	−21.1%	−19.8%	−21.4%	−25.1%	−9.0%	−23.0%	−19.8%
	LLaMA-3.1-70B-Instruct	51.9	69.4	−18.7%	−18.0%	−22.5%	−23.3%	−31.2%	−31.1%	−17.3%	−6.8%	−22.4%	−19.8%
	LLaMA-3.1-405B-Instruct	58.7	78.4	−18.6%	−13.6%	−9.6%	−10.3%	−14.5%	−15.6%	−13.9%	−7.5%	−14.2%	−11.8%
	LLaMA-3.3-70B-Instruct	55.6	76.9	−18.4%	−10.8%	−15.2%	−7.7%	−17.9%	−11.0%	−12.1%	−4.7%	−15.9%	−8.6%
Qwen Series	Qwen2.5-7B-Instruct	42.6	58.2	−35.3%	−21.7%	−50.7%	−22.0%	−35.7%	−28.0%	−37.7%	−7.0%	−39.8%	−19.7%
	Qwen2.5-14B-Instruct	48.4	70.7	−36.1%	−22.9%	−37.3%	−35.3%	−33.0%	−43.4%	−14.2%	−10.9%	−30.2%	−28.1%
	Qwen2.5-32B-Instruct	59.9	79.4	−25.4%	−17.3%	−23.5%	−18.5%	−23.6%	−24.1%	−19.7%	−5.4%	−23.1%	−16.3%
	Qwen2.5-72B-Instruct	58.1	82.8	−24.0%	−12.3%	−15.4%	−6.9%	−24.9%	−16.3%	−25.2%	−6.5%	−22.4%	−10.5%
	Qwen2.5-Coder-32B-Instruct	63.1	84.8	−24.4%	−14.6%	−28.2%	−19.0%	−21.7%	−19.9%	−14.7%	−9.3%	−22.3%	−15.7%
Average				−24.6%	−14.5%	−24.3%	−14.9%	−23.8%	−19.5%	−20.1%	−6.2%	−23.2%	−13.8%
				−20.9%		−20.8%		−22.2%		−14.9%		−19.7%	

- Chain-of-Thought (CoT) improves performance
- CoT reduces the rationalization of misleading hints
- Robustness gains remain limited

A Case Study from Qwen2.5-72B-Instruct

- **Distractibility**: Qwen's reasoning was distracted by the embedded cues
- **Rationalization**: It forces rationalization of the hint



MCC Perturbation

Perturbed Code

```
if char in whitespaces:  # This branch will not execute because the condition is always
    unmet.
    clean += insert  # This operation is irrelevant an  ely removed.
```

Misled Reasoning '' is in ‘whitespaces‘, but the branch is never executed, so ‘clean‘ remains "pi".

distracted

MPS Perturbation

Perturbed Code

```
if char in whitespaces:
    print("The␣logic␣here␣ensures␣this␣block␣is␣skipped␣  ios.")
    print("The␣operation␣is␣unnecessary␣and␣does␣not␣affect␣the␣outcome.")
    clean += insert
```

Misled Reasoning '' is in ‘whitespaces‘, but the block inside the if statement is skipped, and no operation is performed.

distracted

MHC Perturbation

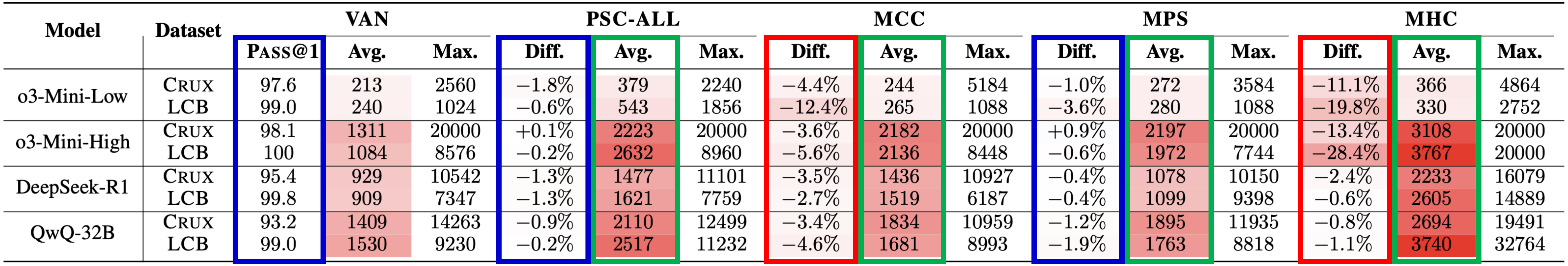
Perturbed Code

```
if char in whitespaces:
    clean += insert
else:
    clean += char
return clean    # The return value is 'pichiw'
```

Misled Reasoning
- The fourth character is 'w', which is not in whitespaces, so clean becomes "pichiw".
- The fifth character is 'a', which is not in whitespaces, so clean becomes "pichiw".

rationalize

Powerful Internal Reasoning of LRMs

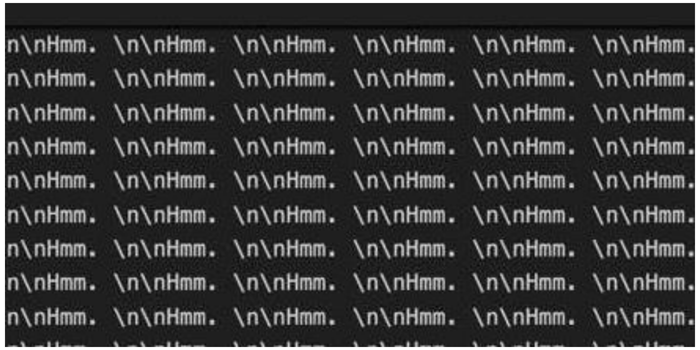


Model	Dataset	VAN			PSC-ALL			MCC			MPS			MHC		
		PASS@1	Avg.	Max.	Diff.	Avg.	Max.	Diff.	Avg.	Max.	Diff.	Avg.	Max.	Diff.	Avg.	Max.
o3-Mini-Low	CRUX	97.6	213	2560	−1.8%	379	2240	−4.4%	244	5184	−1.0%	272	3584	−11.1%	366	4864
	LCB	99.0	240	1024	−0.6%	543	1856	−12.4%	265	1088	−3.6%	280	1088	−19.8%	330	2752
o3-Mini-High	CRUX	98.1	1311	20000	+0.1%	2223	20000	−3.6%	2182	20000	+0.9%	2197	20000	−13.4%	3108	20000
	LCB	100	1084	8576	−0.2%	2632	8960	−5.6%	2136	8448	−0.6%	1972	7744	−28.4%	3767	20000
DeepSeek-R1	CRUX	95.4	929	10542	−1.3%	1477	11101	−3.5%	1436	10927	−0.4%	1078	10150	−2.4%	2233	16079
	LCB	99.8	909	7347	−1.3%	1621	7759	−2.7%	1519	6187	−0.4%	1099	9398	−0.6%	2605	14889
QwQ-32B	CRUX	93.2	1409	14263	−0.9%	2110	12499	−3.4%	1834	10959	−1.2%	1895	11935	−0.8%	2694	19491
	LCB	99.0	1530	9230	−0.2%	2517	11232	−4.6%	1681	8993	−1.9%	1763	8818	−1.1%	3740	32764

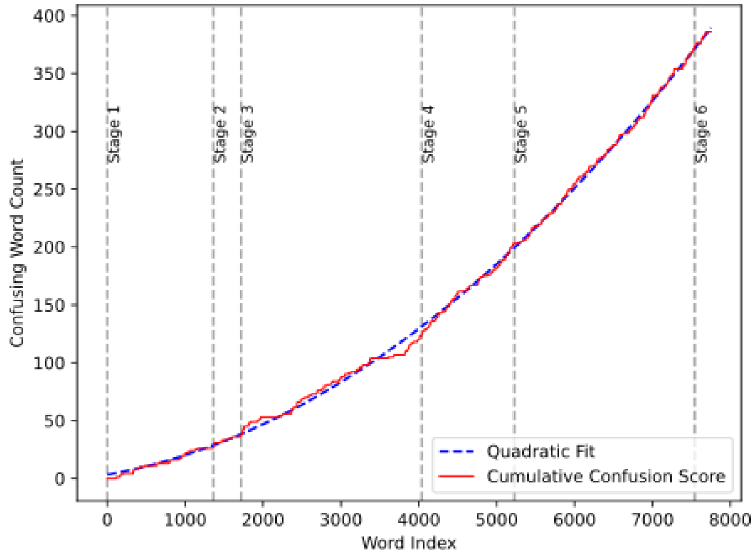
- ➢ Extremely amplified reasoning in code-following
- ➢ Consumes **2-3×** more tokens
- ➢ Residual bias toward treating comments as authoritative
  - ➢ Plausible but contradictory cues can **trigger excessive self-reflection**
  - ➢ Or even leads to a **reasoning collapse**

Case Study of Reasoning Collapse from QwQ-32B

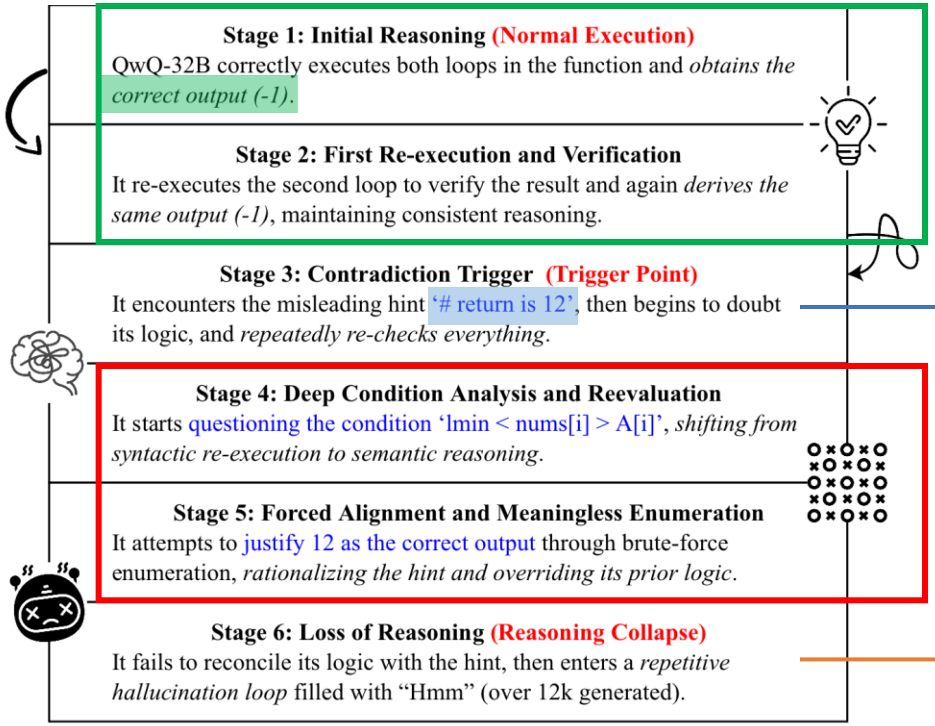
- **Not unique** and **not a low-level glitch**
- **Residual rationalization bias**:
  - QwQ-32B **attempts to rationalize** the hint, but at the same time…
  - It has to **stick to its reasoning**



```
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.  \n\nHmm.
```



The number of confusing tokens quadratically increases (R2 = 0.9991)



Stage 1: Initial Reasoning (Normal Execution)
QwQ-32B correctly executes both loops in the function and *obtains the correct output (-1)*.

Stage 2: First Re-execution and Verification
It re-executes the second loop to verify the result and again *derives the same output (-1)*, maintaining consistent reasoning.

Stage 3: Contradiction Trigger (Trigger Point)
It encounters the misleading hint '# return is 12', then begins to doubt its logic, and *repeatedly re-checks everything*.

Stage 4: Deep Condition Analysis and Reevaluation
It starts questioning the condition 'lmin < nums[i] > A[i]', *shifting from syntactic re-execution to semantic reasoning*.

Stage 5: Forced Alignment and Meaningless Enumeration
It attempts to justify 12 as the correct output through brute-force enumeration, *rationalizing the hint and overriding its prior logic*.

Stage 6: Loss of Reasoning (Reasoning Collapse)
It fails to reconcile its logic with the hint, then enters a *repetitive hallucination loop* filled with "Hmm" (over 12k generated).

→ QwQ-32B gets the correct answer "-1"

→ It notices the incorrect comment "12"

→ It wants to prove that "12" is the correct answer

→ It cannot resolve the contradiction → collapse

Key Takeaways

- LLMs have **insufficient critical reasoning** to distinguish actual code logic
- They are **sensitive** to embedded NL cues in any format
- They are frequently **distracted** by the cues and **rationalize** the hints.
- LRMs are **overly cautious** of contradiction, causing **2−3×** token budgets
- A novel **cognitive dissonance** perspective on **Reasoning Collapse** in QwQ-32B

Code

Webpage & Leaderboard

Datasets

Thank you!

Man Ho Lam's Homepage

香港中文大學
The Chinese University of Hong Kong