JUMPCODER: Go Beyond Autoregressive Coder via Online Modification

Mouxiang Chen, Hao Tian, Zhongxin Liu, Xiaoxue Ren, Jianling Sun

The State Key Laboratory of Blockchain and Data Security, Zhejiang University

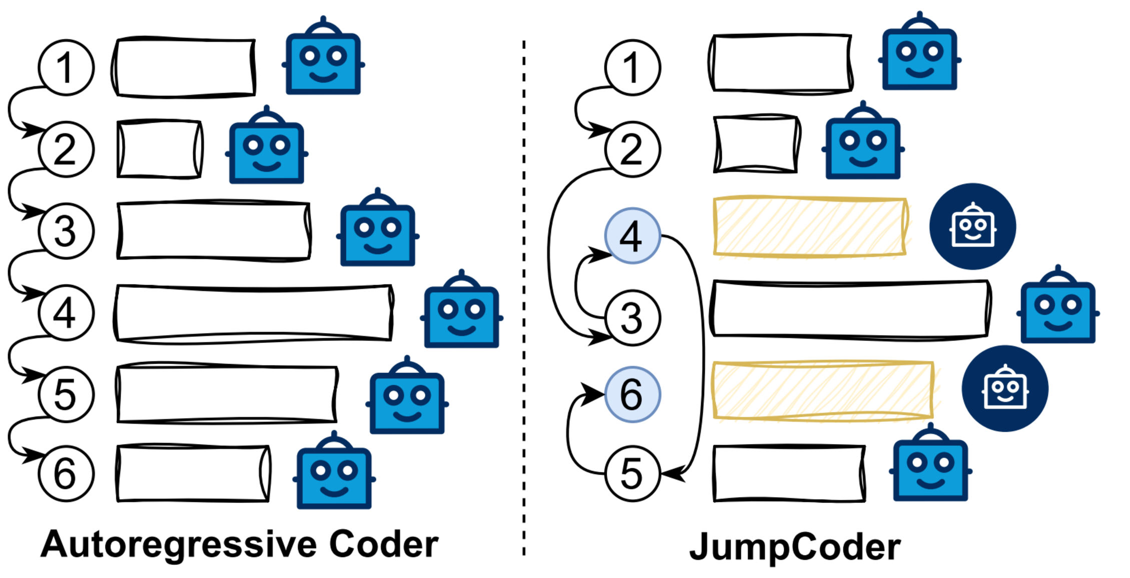
ACL 2024 Bangkok, Thailand

TL; DR

➤**Motivation**: Traditional code LLMs (**Autoregressive Coder**) generate code in a linear, **irreversible** sequence. This can lead to errors accumulating over time.

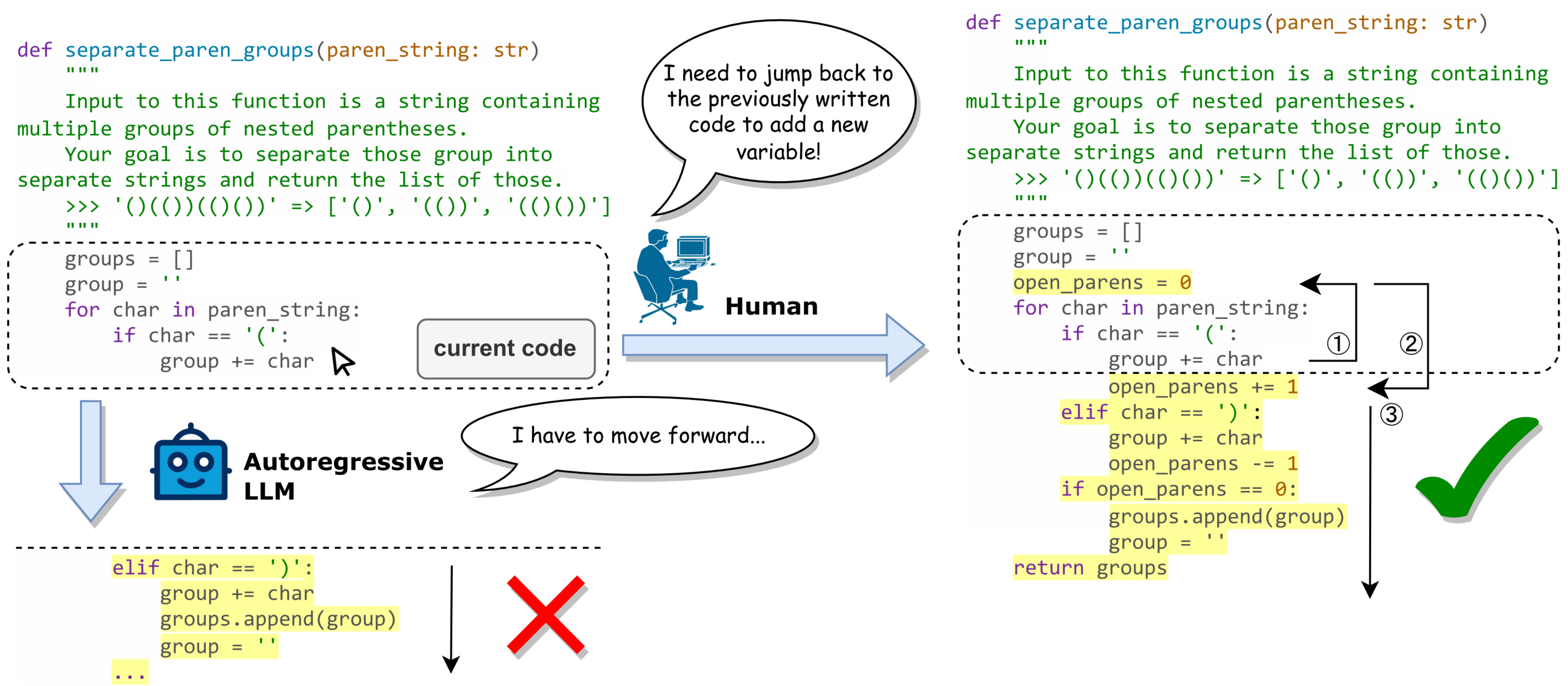
➤**Method:** We introduce **JumpCoder**, a model-agnostic code generation framework for augmenting code LLMs without retraining.

➤**How it Works**: JumpCoder can insert new code into currently generated code on-the-fly with an auxiliary **infilling model.**



◀ Schematic illustrations of traditional autoregressive coder and the proposed JumpCoder.

1. Motivation Example



◀ An illustrative example demonstrating the difference between humans and LLMs.

➤ When a new variable is required, humans can ***jump back*** to the front section to define it.

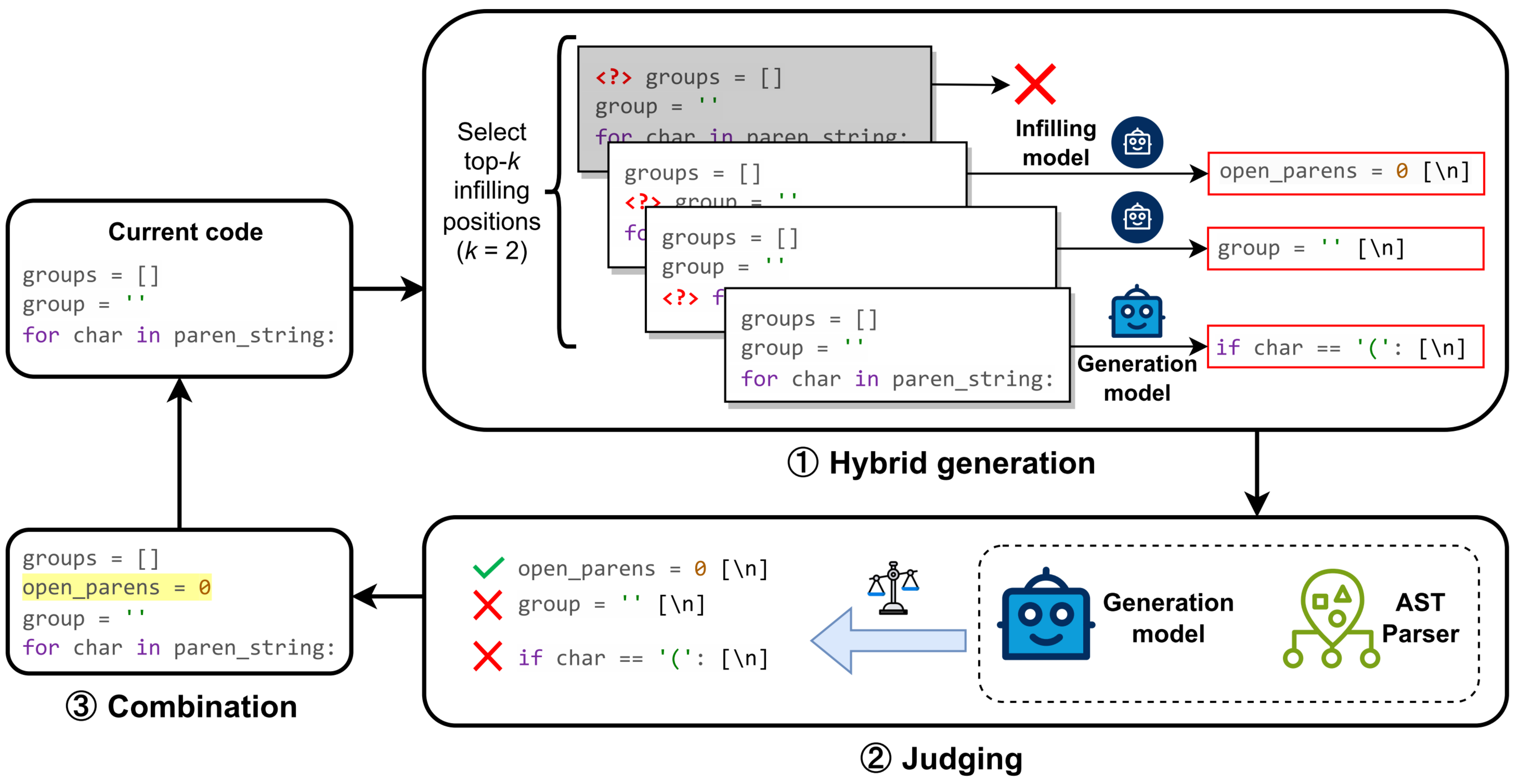
➤ But LLMs, constrained by their autoregressive nature, can ***only continue generation*** and lead to error propagation.

2. Challenges

➤ **Challenge 1:** How to infill a line?
  ➤ Use a pre-trained infilling model.
  ➤ *e.g.*, InCoder, CodeLlama-Instruct.

➤ **Challenge 2:** Whether (and where) to infill, or continue generation?
  ➤ **infill first, judge-later:** ① let infill model experiment with filling at the start of the k most critical lines; ② judge their contributions to the current generation.

3. Method



▲ JumpCoder Framework. The iterative code update process comprises three important stages: Hybrid generation, Judging and Combination. Each iteration inserts a new line of code.

① **Hybrid Generation**

➤ Generate $k+1$ lines of code: k infills, 1 line of continual generation

➤ Efficiency Optimization: Parallel generation, Speculative infilling

② **Judging**

➤ **AST Parser:** accepts the infill that adds the missing declaration.

➤ **Generation Model Scoring:** scores the code following the infill. If improved, accept the infill.

➤ **Other case:** continue generation.

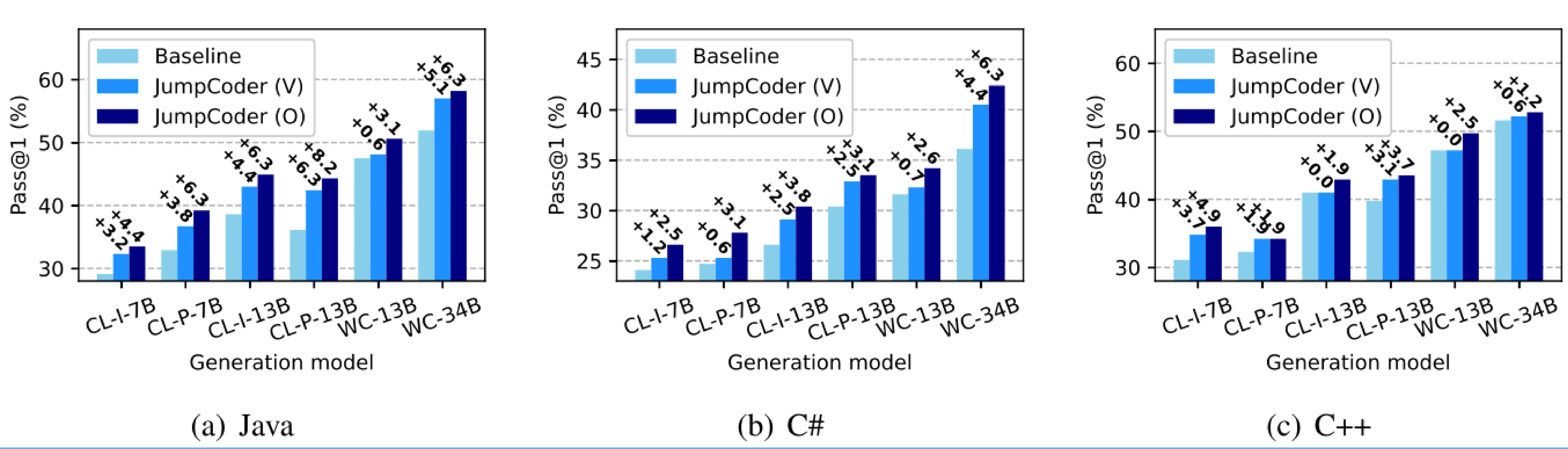
③ **Combination**

➤ Combine the line of code after judging into existing n lines of code

4. Experiments

Generation model	Method	HumanEval	MBPP
CodeLlama-Instruct (7B)	-	36.0	42.4
	+ JC (V)	37.8 (+1.8)	44.8 (+2.4)
	+ JC (F)	**39.6 (+3.6)**	**45.2 (+2.8)**
	+ JC (O)	*39.6 (+3.6)*	*45.2 (+2.8)*
CodeLlama-Python (7B)	-	38.4	43.2
	+ JC (V)	40.2 (+1.8)	45.4 (+2.2)
	+ JC (F)	**41.5 (+3.1)**	**45.6 (+2.4)**
	+ JC (O)	*41.5 (+3.1)*	*46.8 (+3.6)*
CodeLlama-Instruct (13B)	-	40.9	45.8
	+ JC (V)	**44.5 (+3.6)**	**46.8 (+1.0)**
	+ JC (F)	43.9 (+3.0)	46.6 (+0.8)
	+ JC (O)	*45.7 (+4.8)*	*48.0 (+2.2)*
CodeLlama-Python (13B)	-	43.9	50.0
	+ JC (V)	**45.7 (+1.8)**	**51.0 (+1.0)**
	+ JC (F)	**45.7 (+1.8)**	50.8 (+0.8)
	+ JC (O)	*47.0 (+3.1)*	*53.2 (+3.2)*
WizardCoder-Python (13B)	-	64.0	56.8
	+ JC (V)	64.6 (+0.6)	**57.2 (+0.4)**
	+ JC (F)	**65.2 (+1.2)**	**57.2 (+0.4)**
	+ JC (O)	*65.9 (+1.9)*	*57.2 (+0.4)*
WizardCoder-Python (34B)	-	73.8	59.2
	+ JC (V)	**74.4 (+0.6)**	59.2 (+0.0)
	+ JC (F)	**74.4 (+0.6)**	**59.6 (+0.4)**
	+ JC (O)	*75.0 (+1.2)*	*60.0 (+0.8)*

▲ **Results of Pass@1 (%) on HumanEval and MBPP using greedy generation.** JC (V): Use code from JumpCoder. JC (F): use code from JumpCoder and Autoregressive Coder based on the lower perplexity. JC (O): use code from the above two based on evaluation test cases, served as an upper bound.



(a) Java (b) C# (c) C++

◀ **Results on MultiPL-E.** On average, JumpCoder passes an additional 5.8% (Java), 3.6% (C#) and 2.7% (C++) problems.

chenmx@zju.edu.cn