JULY 27TH 2026

TEXAS
The University of Texas at Austin

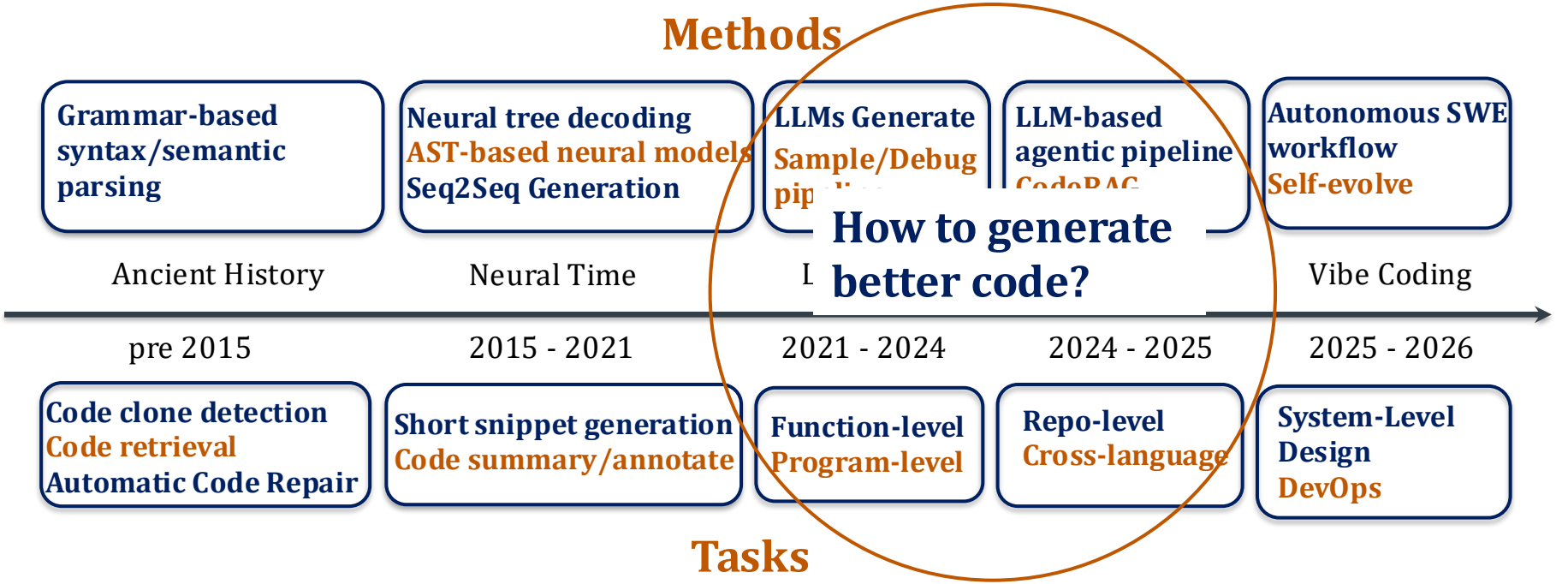
ENHANCING COMPETITIVE-LEVEL CODE GENERATION BY UTILIZING NATURAL LANGUAGE REASONING

Ph.D. Oral Defense

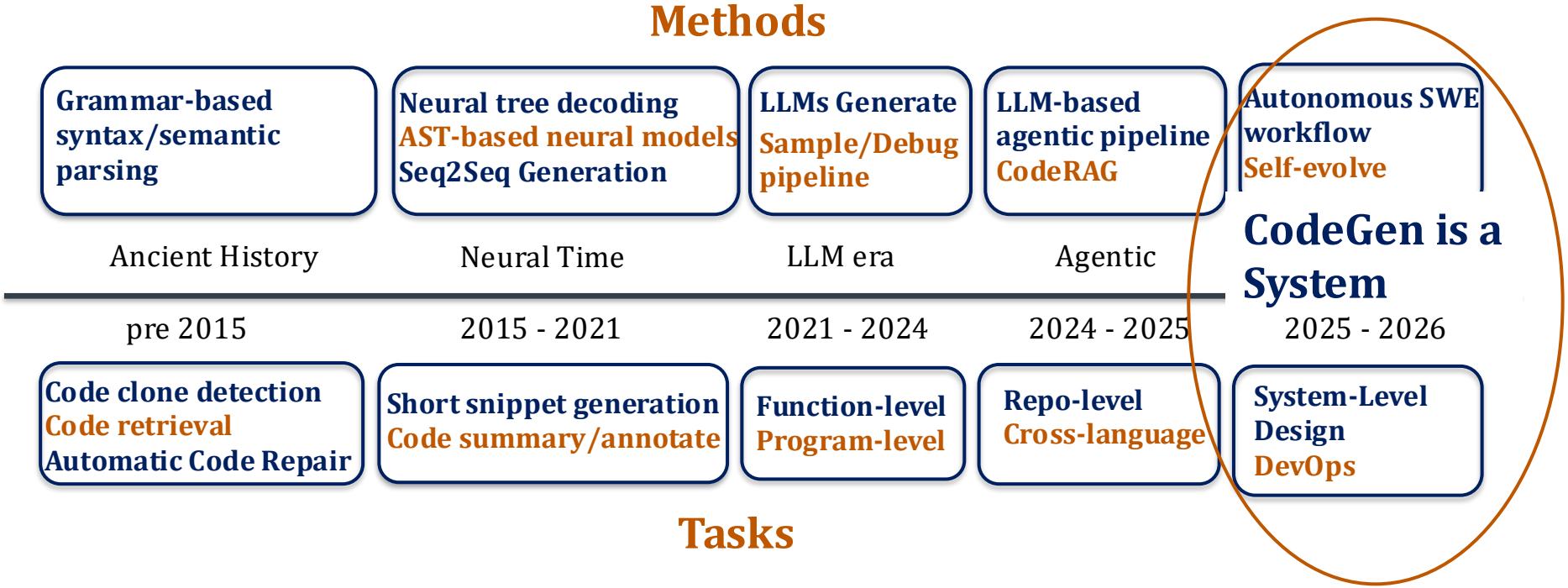
Committee members: Prof. Raymond Mooney(advisor), Prof. Jessy Li, Prof. Milos Gligoric, Prof. Huan Sun

PRESENTER: JIERUI LI
Ph.D. Candidate, The University of Texas at Austin

Stepping into the LLM-era for CodeGen



Stepping into the LLM-era for CodeGen



Outline

Introduction: Competitive Programming, CodeGen

Investigate Code Generation Bottleneck (NLRSE @ACL 2023)

Explaining Competitive-level Programming Solutions Using LLMs
Jierui Li, Szymon Tworkowski, Yingying Wu, Raymond Mooney

Enhance Code Generation (NLRSE @ ACL 2024; NAACL 2025)

Distilling Algorithmic Reasoning from LLMs via Explaining Solution Programs
Jierui Li and Raymond Mooney

CodeTree: Agent-guided Tree Search for Code Generation with LLMs
Jierui Li, Hung Le, Yingbo Zhou, Caiming Xiong, Silvio Savarese, Doyen Sahoo

Outline

Empower the Code Generation System (IJCAI 2026; submitted)

AlgoSimBench: Identifying Algorithmically Similar Problems for Competitive Programming
Jierui Li, Raymond Mooney

InstEmbed: Instruction-Sensitive Code Embeddings via Task-Contrastive Training (Under Review)
Jierui Li, Raymond Mooney

Future Work & Conclusion

Competitive Programming: a crucial task for LLMs

- Formalized Problem, Gold Evaluation


Detailed, Disambiguious Problem Statement


Example Input/output


Massive and Comprehensive Test Cases


Online Judge (web) w/ Time/Space constraints

- Requires a Complete Reasoning-to-Code Pipeline

Understand Problem; Abstract into Math Form → Discover Algorithm; Reason Complexity → Finalize Detail; Implement Solution → Test; Debug

- Distinguishes Genuine Reasoning from Pattern Matching
  - One condition changes in problem, required algorithm is a different one
  - Same background story, different problem logic and algorithms

Competitive Programming: a crucial task for LLMs

- Formalized Problem, Gold Evaluation


Detailed, Disambiguious Problem Statement


Example Input/output


Massive and Comprehensive Test Cases


Online Judge w/ Time/Space constraints

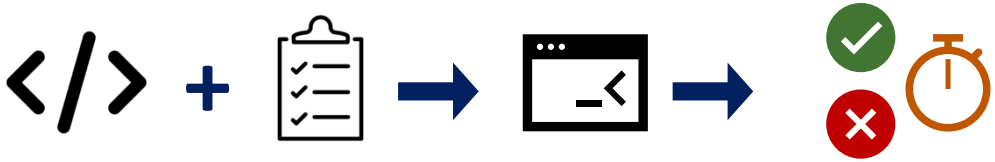
- Requires a Complete Reasoning-to-Code Pipeline

Understand Problem; Abstract into Math Form → Discover Algorithm; Reason Complexity → Finalize Detail; Implement Solution → Test; Debug

- Distinguishes Genuine Reasoning from Pattern Matching
  - One condition changes in problem, required algorithm is a different one
  - Same background story, different mathmatical abstraction

Competitive Programming: Drives Research on LLMs

- Reinforcement Learning with Verifiable Rewards

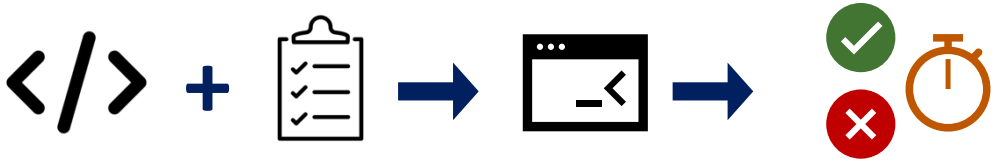


- Optimizing Efficiency beyond Correctness

- Test-time Scaling
  - Sample Different Solutions(scale width); Iteratively Debug(scale depth)
  - Trackable, Verifiable Trajectories

Competitive Programming: Drives Research on LLMs

- Reinforcement Learning with Verifiable Rewards



- Optimizing Efficiency beyond Correctness

- Test-time Scaling
  - Sample Different Solutions(scale width); Iteratively Debug(scale depth)
  - Trackable, Verifiable Trajactories

Challenges in Competitive Programming Problems

2023

Supervised Fine-tuning

Challenges in Competitive Programming Problems

- Learn to generate code (w/ or w/o Chain of Thought)

  Competition-Level Code Generation with AlphaCode (Li et al., 2022)
  Structured Chain-of-Thought Prompting for Code Generation (Li et al., 2023)

- Learn to iteratively refine the code by executing

  Reflexion: Language Agents with Verbal Reinforcement Learning (Shinn et al., 2023)
  Teaching Large Language Models to Self-Debug (Chen et al., 2023)

Challenges in Competitive Programming Problems

Programming Language	Poor Readability	Different Coding Style
Python, Java, C++, C, PHP, JavaScript, ...	"a, aa, lc" to name variables unused "header", Misguiding comment	• Solutions from multiple competitors • Don't follow style guide

Code collected from competitors is noisy in styles

We don't have expert-written trajectories for this

Outline

- Introduction: Competitive Programming, CodeGen

- Explaining CP problems
- Distilling Algorithmic Reasoning Via Explaining Solutions
- CodeTree: Agent-guided Tree search for CodeGen
- AlgoSimBench: Identifying Algorithmically Similar CP problems
- InstEmbed: Instruction-Sensitive Code Embeddings

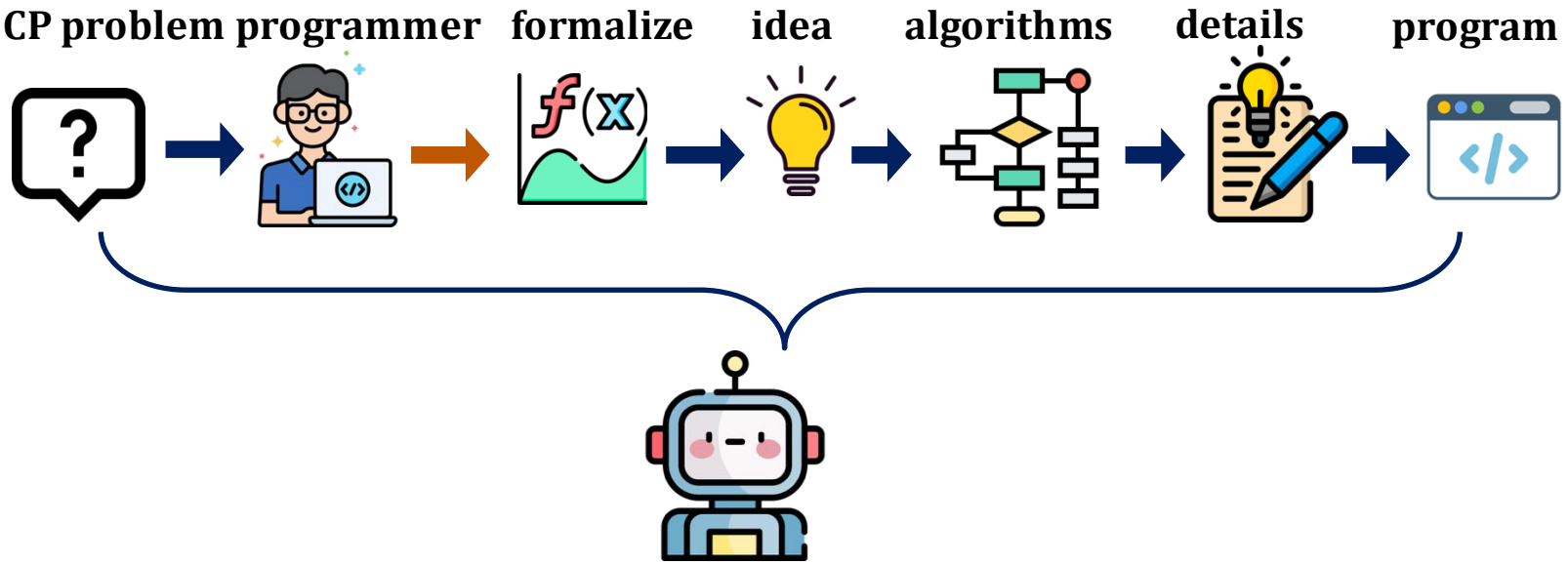
- Future Work & Conclusion

Explaining Competitive-Level Programming Solutions using LLMs

(Li and Mooney, NLRSE@ACL 2023)

RQ1: What's the bottleneck of LLM solve CP? Idea or implementation?
RQ2: Can LLMs explain poor-styled, complex CP solution code and understand the algorithm in it?

How human programmers understand and learn from those solutions?



Can we directly prompt LLMs to do this?

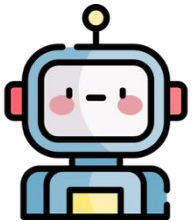
Baseline Prompt: Here's a Coding Challenge below, you

Let's think step-by-step

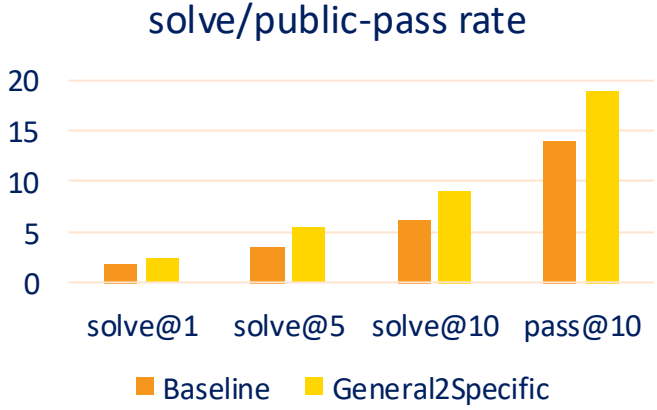
problem.

General-to-Specific Prompt: Baseline Prompt + "Please

Let's think from general idea to implement: 1234567

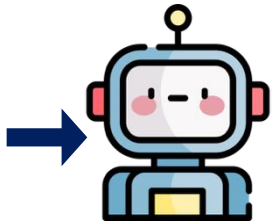


pass@k (Chen et al., 2021): each problem sample k solutions, percentage of problems with at least one solution pass all public tests.
solve@k: accepted by the online judge
Benchmark: CodeContests (165 problems) (Li et al., 2022); Model: GPT-3.5

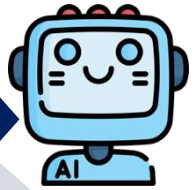


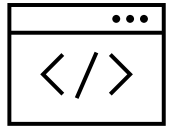
Thinking of the correct idea is hard?
Or, implementing it is hard?

Here's a ***problem*** & its ***human-written solution*** in Python, can you **Explain** it following the steps (1)(2)(3)(4)(5)(6)(7)?



(1)
(2)
(3)
(4)
(5)
(6)
(7)





Can you implement the solution based on (1)(2)(3)(4)(5)(6)(7)?

Evaluate the Quality of Explanation: Performance of the code generated from it

(1)Brief Problem Summary

(2)Used Algorithm

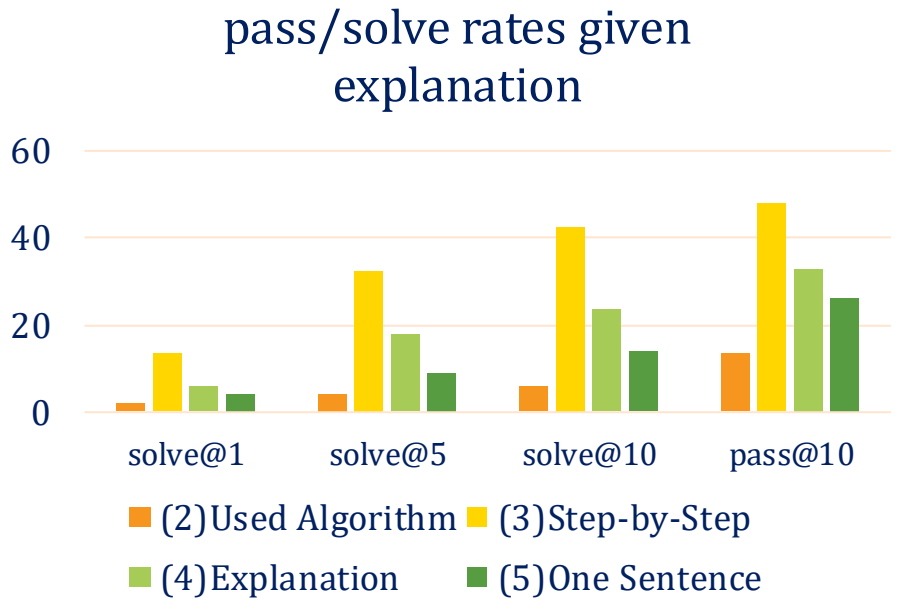
(3)Step-by-Step Solution Description

(4)Explanation of the Solution

(5)One Sentecne of Solution

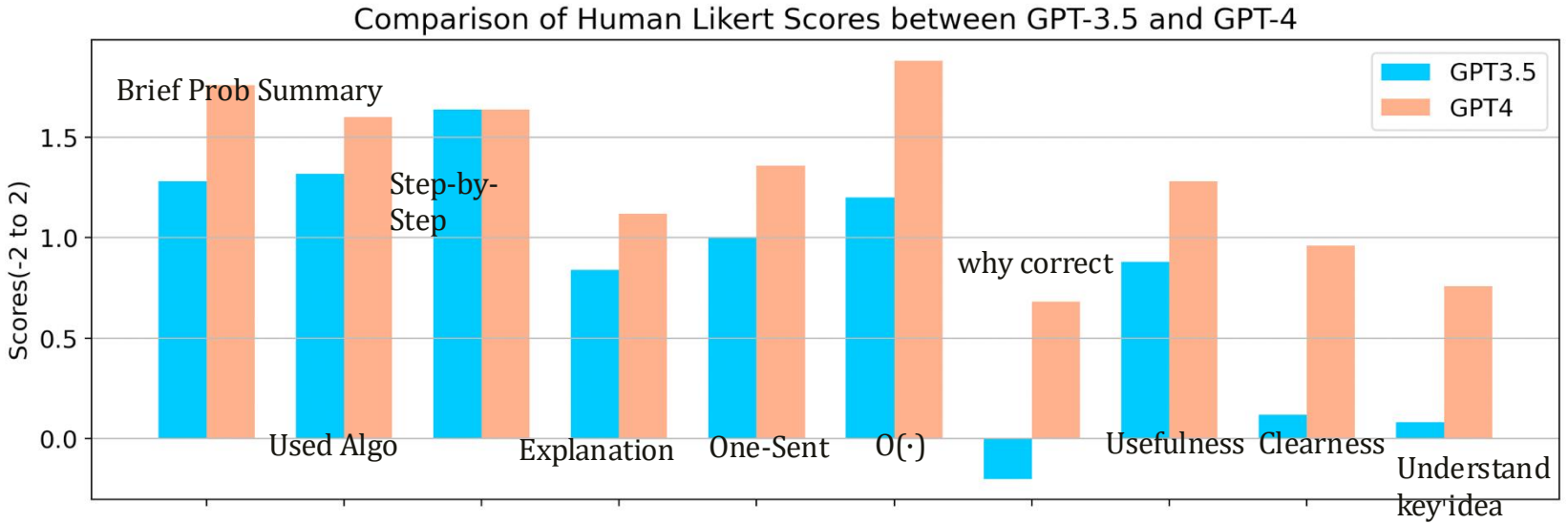
(6)Time Complexity

(7)Why this solution is correct (Key Idea)



Evaluate the Quality of Explanation: Human Author of the solution Rate it

Likert Scores of 50 problems with rankings from 800 to 2000. (-2: strongly dislike; 2: strongly like it)



Takeaways (GPT-3.5, early 2023):

LLMs are bad at:

- Solve CP problems, with best prompting can yield 9% solve@10 [1]
- Explain why a solution is correct, -0.3 likert for [-2,2]

LLMs are good at:

- Explain human-written solutions step by step, 1.6 likert for [-2,2]
- Implement solution given step-by-step description, 42.4% solve@10[1]

[1] Only 73.9% of the problems can be solved with Python

Contributions

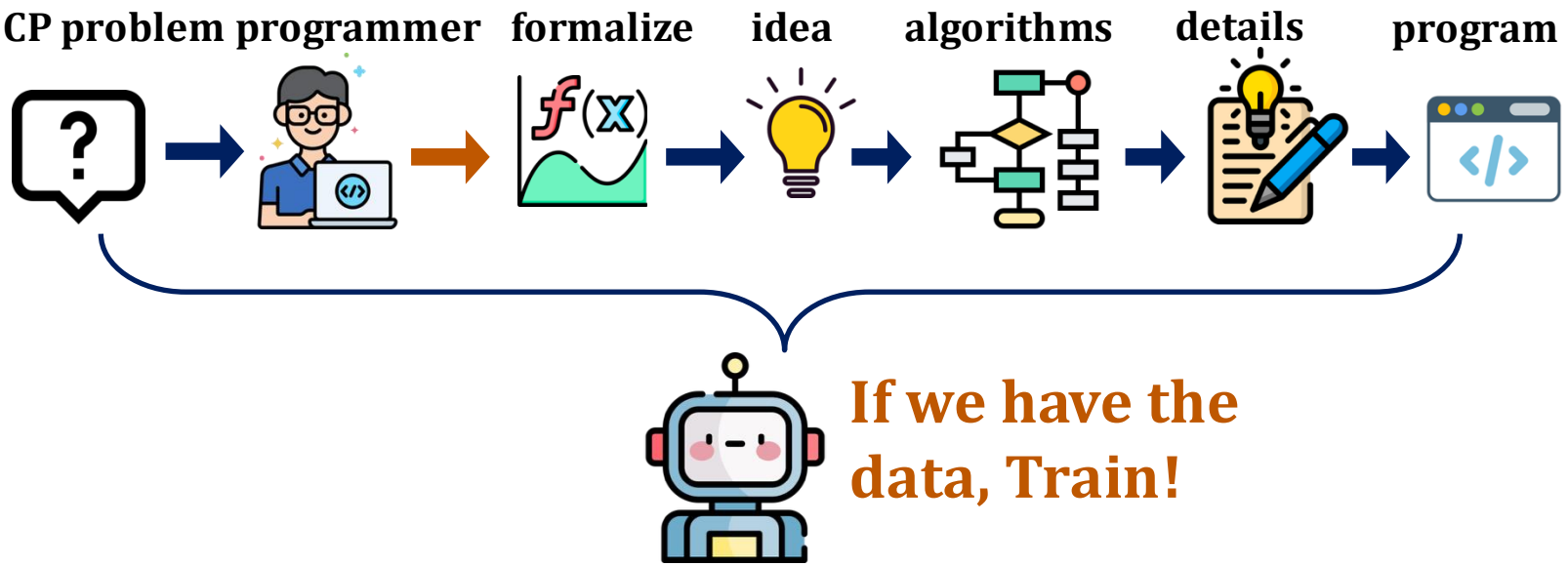
- To the best of our knowledge, we are among the first works to evaluate LLMs' abilities in explaining human-written code, and found that LLMs generated explanations are
  - liked by authors of code
  - serve as effective hints to better solve the problems
- Present insightful findings on what current LLMs are good and bad at in terms of code generation and understanding.
- Propose General2Specific prompting that outperforms CoT.

Distilling Algorithmic Reasoning from LLMs via Explaining Solution Programs

(Li and Mooney, NLRSE@ACL 2024)

RQ3: Can we leverage LLMs' abilities in explaining code and implementation?

Can LLMs learn to solve CP problems like human?



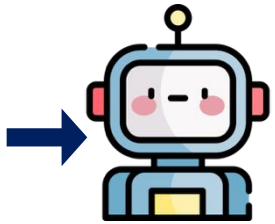
- **Experts write Editorial:** a comprehensive explanation or guide that discusses the problems presented in a programming contest or challenge, which often contains some of the following aspects:
  - Problem restatment
  - Difficulty
  - Prerequisites
  - Quick explanation
  - Explanation
  - Code
  - Pitfalls(what to avoid)

- **Experts write Editorial:** a comprehensive explanation or guide that discusses the problems presented in a programming contest or challenge, which often contains some of the following aspects.
  - Problem restatment
  - Difficulty
  - Prerequisites
  - Quick explanation
  - Explanation
  - Code
  - Pitfalls(what to avoid)

- **Hard to collect at scale:** Experts write editorials of different styles and languages(English, Chinese, Russian, ...) in:
  - Blogs
  - Personal Websites
  - Discussion Forums
  - YouTube Channels
  - Comments to the problem
  - Photos of hand-written drafts
  - ...

Luckily, LLMs can generate quality explanation from:

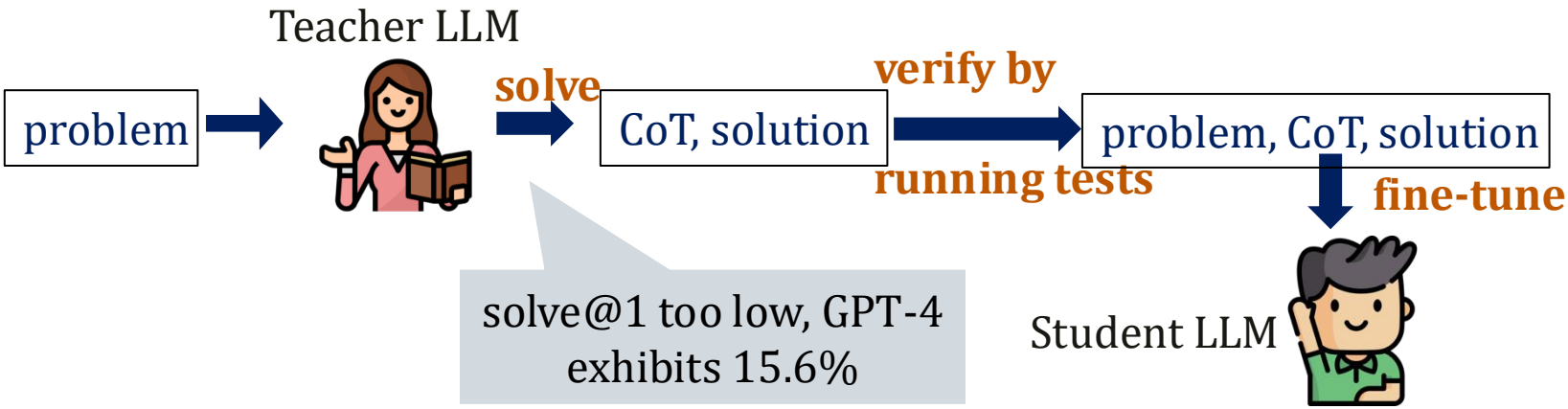
Here's a ***problem*** & its ***human-written solution*** in Python, can you **Explain** it following the steps (1)(2)(3)(4)(5)(6)(7)?



(1)
(2)
(3)
(4)
(5)
(6)
(7)

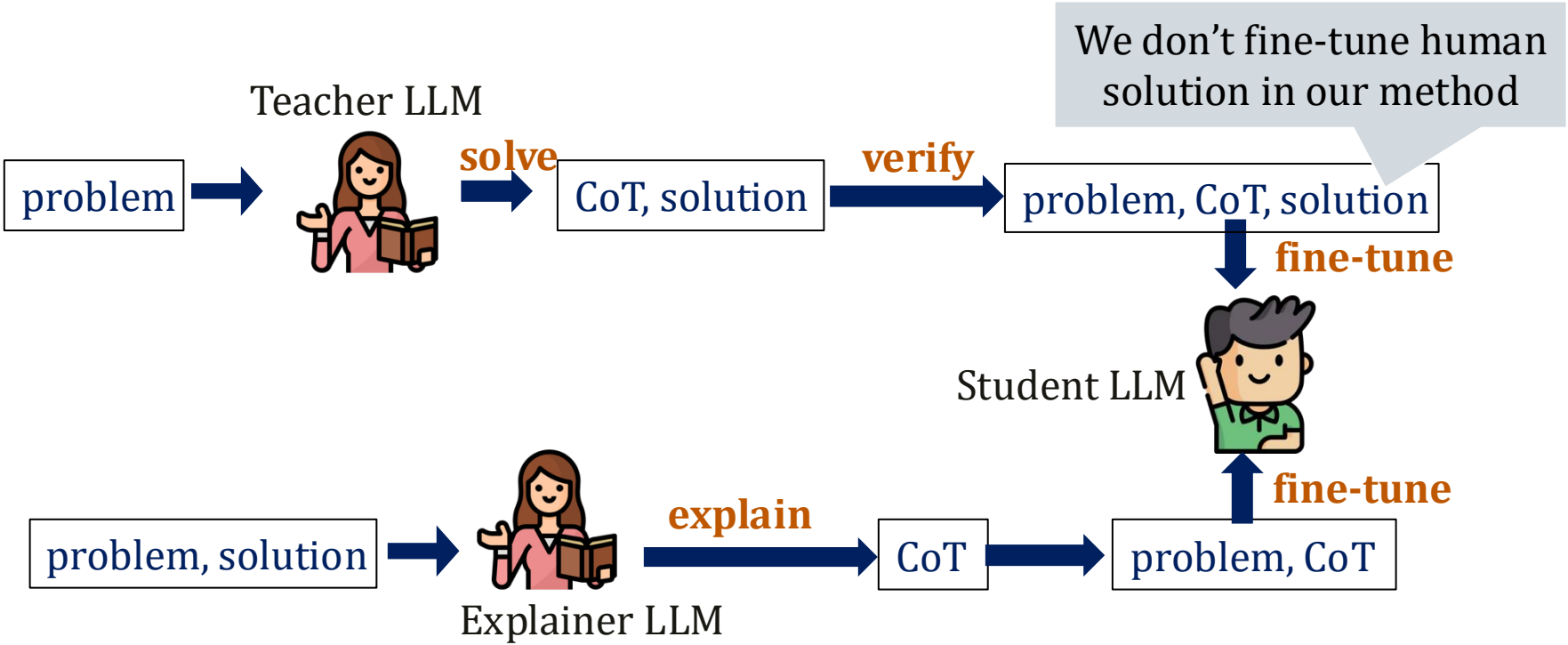
Can explanations be distilled as chain of thoughts to LLMs?

Existing Baseline: LLMs Reasoning Distillation



Hsieh, Cheng-Yu, et al. "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes." *arXiv preprint arXiv:2305.02301* (2023).

Ours: Distill Explaining as Reasoning



Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs

Instead of having a teacher to solve the problem, we have:

- An Explainer explains human solution into CoT 
- A Reasoner to learn to generate CoT from problem 
- A Coder to implement code from generated CoT during infernce

Learn to generate Chain-of-thoughts only, as we trust zero-shot LLMs' capacities in implementing solutions.

Dataset and Evaluation Metrics

Dataset Source: To ensure GPT-3.5/GPT-4 **hasn't seen** the problems. We collect 246 problems with release dates later than Aug 2023, with similar distribution than CodeContests.

Explanation Collection: We generate 8248 editorial style explanations for distinct <problem, solution> pairs from GPT-4.

Metric: Solve@k, For each problem, **Sample k** solutions, submit them to online judge. Total number of problems with **at least one accepted** solution.

Baselines and Our Method

0-shot Coder: Instruct an LLM(Coder) to solve the problem and give the code.

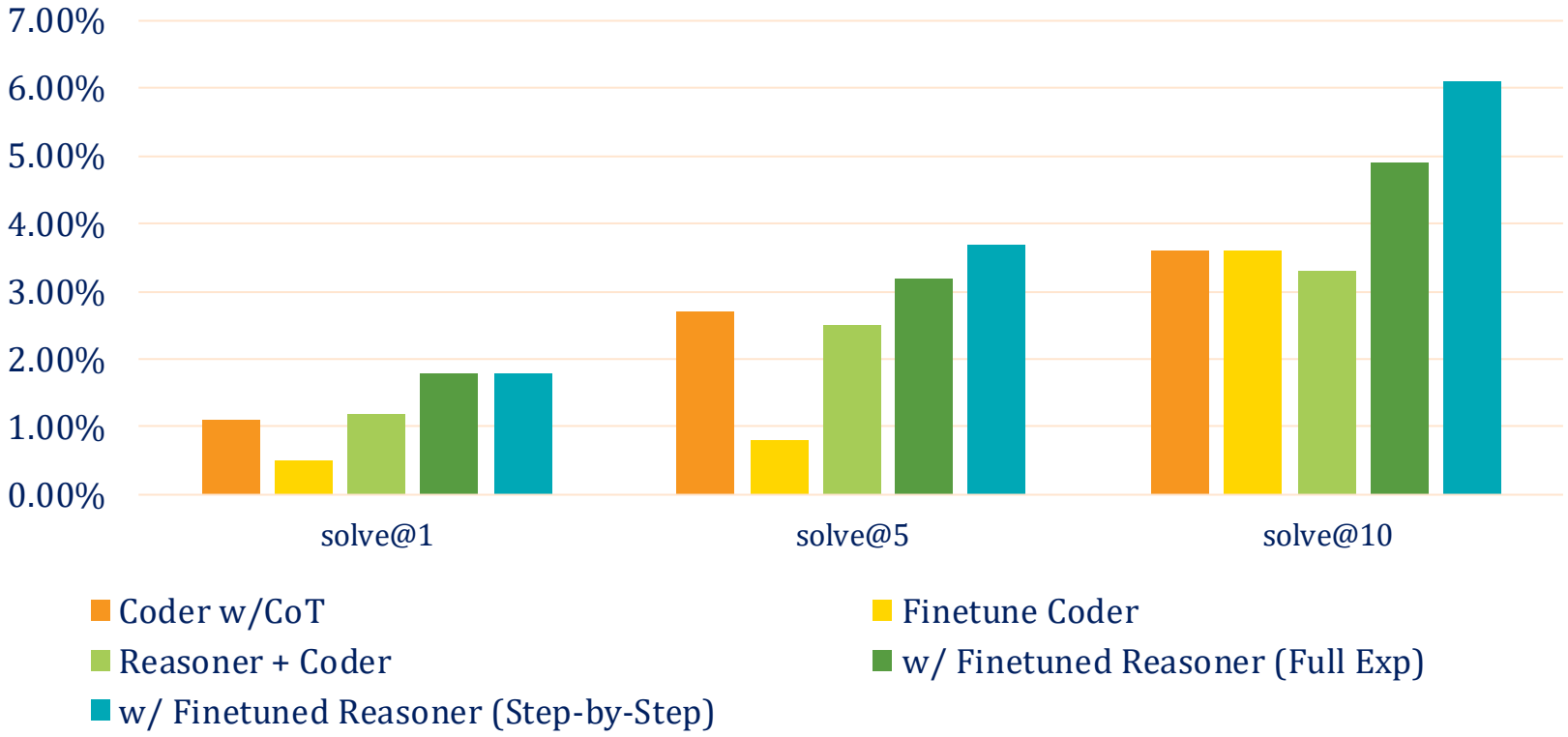
0-shot Coder w/CoT: 0-shot Coder + Chain-of-thought prompting

0-shot Reasoner + Coder: Reasoner to give editorial style CoT; and Coder to implement the program conditioned on the editorial style CoT.

Fine-tuned Coder: Fine-tune an LLM on <problem, solution-program> pairs

Supervised Fine-tune (SFT) for all finetuned mentioned.
All Coder and Reasoner LLMs in this work are **GPT-turbo-3.5.**
We also experimented with DeepSeek-Coder-6.7b (details in paper)

Solve Rates for 0-shot and Funetined Methods



Other Findings

1. Among all passed-public solutions, baselines have an average of 50.7% rejected by online judge due to Time Limit Exceeded.
   Finetuned Step-by-step only has 25.6% of the solutions rejected.

Finetuning on CoT enhance LLM to generate more efficient solutions.

Other Findings

2. k total solutions = $m \times n$, m: #CoTs, n: #solutions per CoT, sampling larger m is always better than larger n

Diverse ideas on how to solve, are better than diverse implementations for the best ideas

Contributions

- We introduce an alternative, adaptable framework to distill complex reasoning processes from LLMs: instead of having LLMs solve problems, it leverages LLMs to explain solutions.

- Our proposed distilling method is data-efficient: by fine-tuning with 8248 data points, 8M tokens in total, consistent performance gains in solve rates are observed on GPT-3.5 and Deepseek coder 7b.

Outline

- Introduction: Competitive Programming, CodeGen

- Explaining CP problems
- Distilling Algorithmic Reasoning Via Explaining Solutions
- CodeTree: Agent-guided Tree search for CodeGen
- AlgoSimBench: Identifying Algorithmically Similar CP problems
- InstEmbed: Instruction-Sensitive Code Embeddings

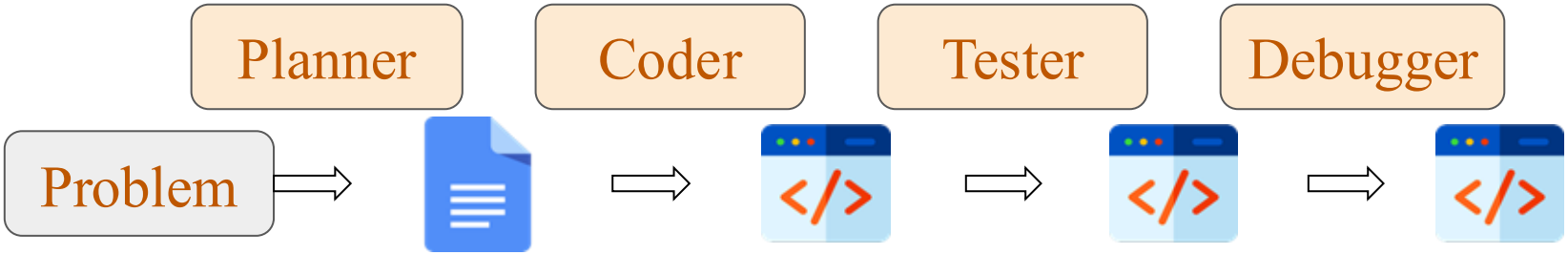
- Conclusion & Future Work

CodeTree: Agent-guided Tree Search for Code Generation with LLMs

(Li et al., NAACL 2025)

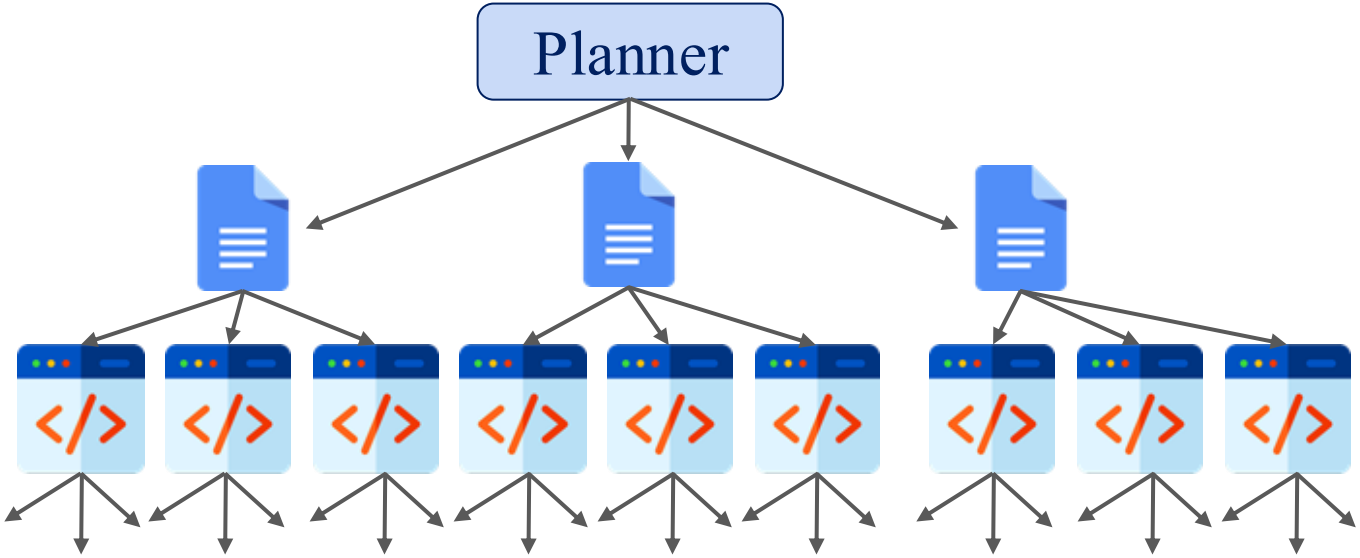
RQ4: Can we apply findings from previous works to build an agentic code generation pipeline?

Agent Collaboration for Code Generation with LLMs



[3]Zhong, Li, Zilong Wang, and Jingbo Shang. "Debug like a human: A large language model debugger via verifying runtime execution step-by-step." *arXiv preprint arXiv:2402.16906* (2024).

Existing Work: Tree Search for Code Generation



[4] Islam, Md Ashraful, Mohammed Eunus Ali, and Md Rizwan Parvez. "Mapcoder: Multi-agent code generation for competitive problem solving." arXiv preprint arXiv:2405.11403 (2024).

Tree Search Trade-off:

- Go Deeper: keep refining one solution
- Go Wider: try different solutions/plans

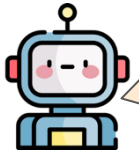
Carefully set the width & depth according to task

Limitations of Previous Works

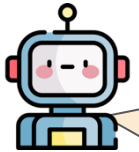
- Agent pipeline is fixed process
- Tree exploration relies on width&depth parameters
- Exit condition is simple (e.g., pass public test cases)

Motivation

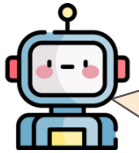
- Agent pipeline is flexible
- Tree Expanding is dynamic
- Exit Condition (Verified)



Should I iterate this method and debug or should I try a different strategy?

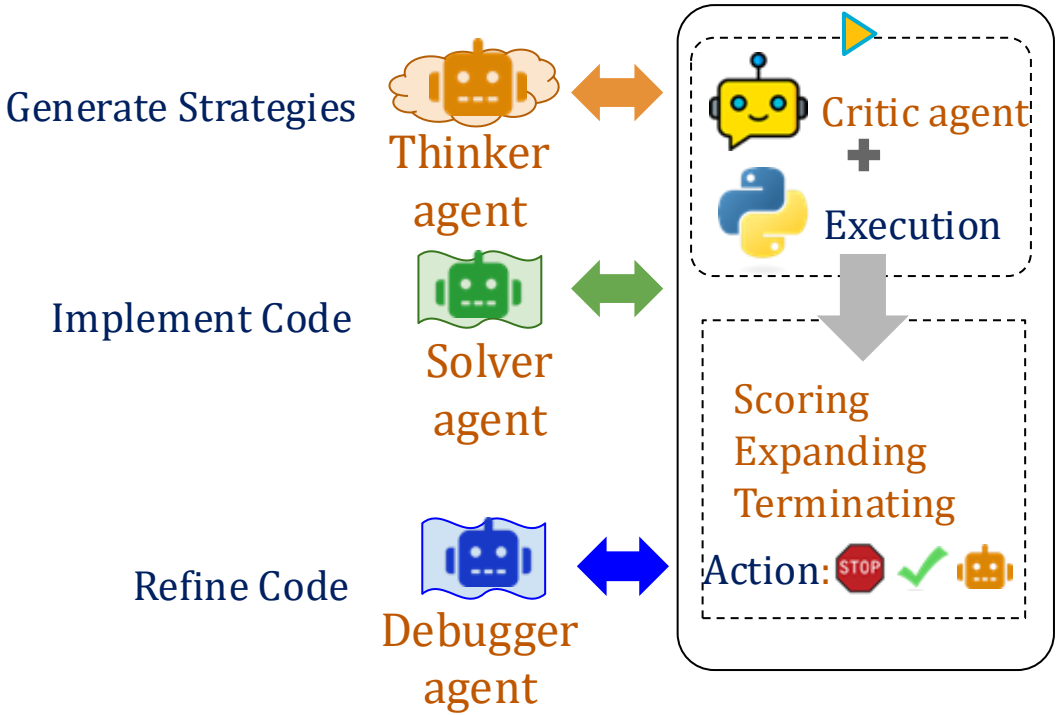


for this problem, maybe try 3 different strategies…



Given test cases are all positive integers, what if they are negative?

Agent-guided tree search



Critic Agent:

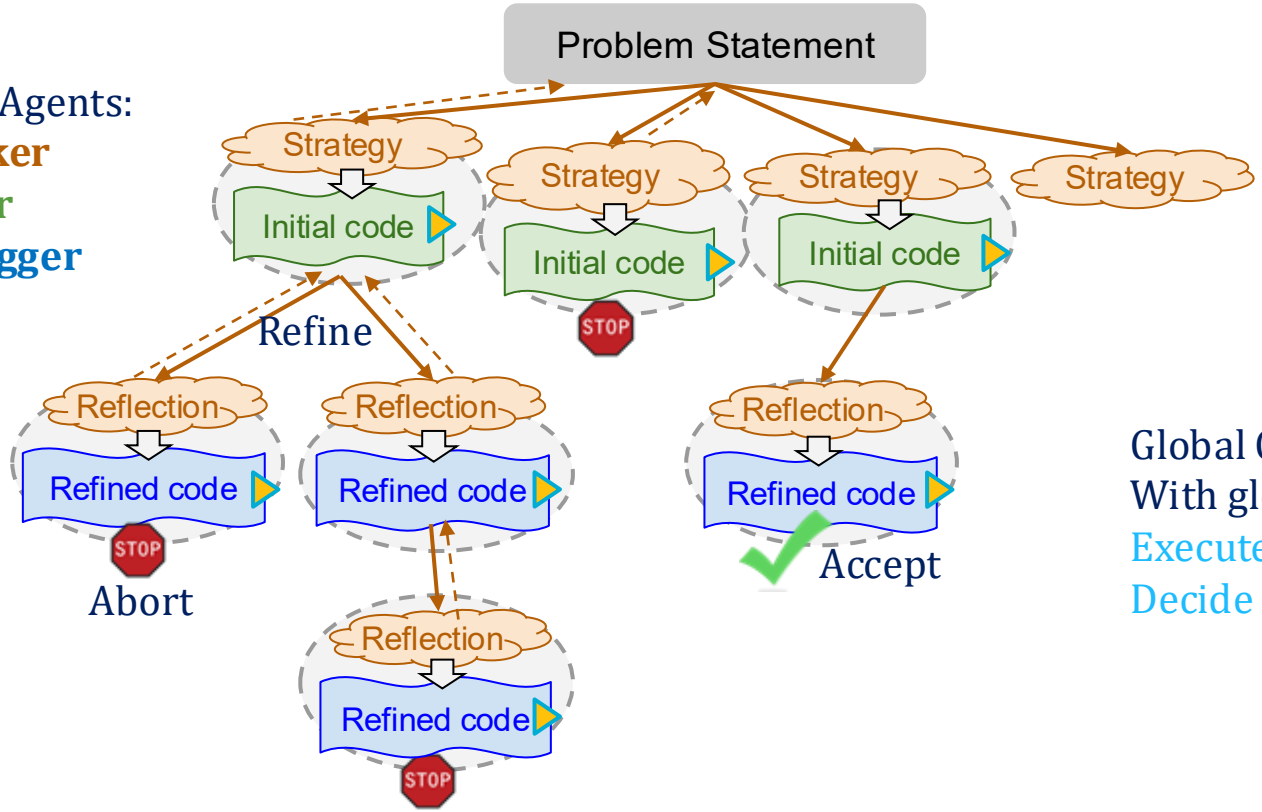
Score: Solutions w/ Execution Feedback

Expand Tree: Next Step, refine or try different strategies

Terminate Search: Judge & Accept the current solution

Method Overview

Local Agents:
Thinker
Solver
Debugger



Global Critic Agent:
With global context
Execute, Score, Critic, Decide Action

Experiments

- **Datasets:**
  - HumanEval(Chen et al., 2021),
  - MBPP(Austin et al., 2021)
  - HumanEval+; MBPPEval+ (Liu et al., 2023)
  - CodeContests(Li et al., 2022)

- **Evaluation Metric:**
  Pass@1 (only 1 program will be run against evaluator); code budget = 20 (at most 20 programs generated per problem)

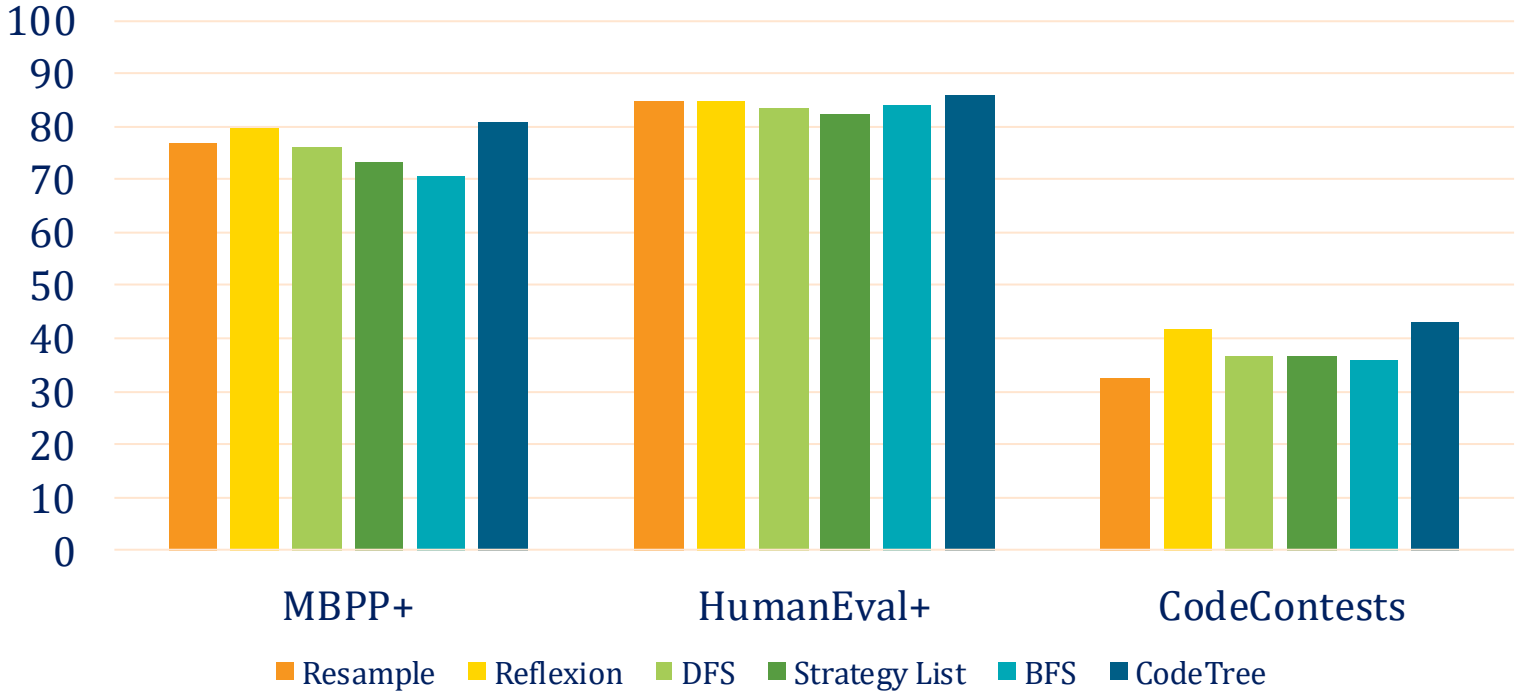
Experiments

- **Baselines**

  Till one solution passes public tests or reaches budget

  - **BFS/DFS**: use BFS/DFS to replace critic agent judge for what nodes to explore and tree expansion
  - **Resample**: Keep resampling
  - **Reflexion**: Keep code-execute-reflect-code (Shinn et al., 2023)
  - **Strategy List**: List budget of strategy, implement one by one
  - **MapCoder**: An agent-based coding framework (Islam et al., 2024)

GPT-4o: Pass@1 budget 20



Other Results & Findings

- Adding budget consistently increase pass@1 for CodeTree but not others
  For BFS, DFS, Strategy List, and Resample, pass@1 stops to grow around budget = 10; Pass@1 for Reflexion stops to grow around budget = 20
  CodeTree keeps growing when budget reaches 30
- Adding budget does not linearly adds usage of tokens for CodeTree?
  CodeTree utilizes tree size control, early termination to avoid looping on problems it cannot solve. It cost fewer tokens than Reflexion on CodeContests
- CodeTree+4o beats O1-preview with fewer tokens on all benchmarks
  Given budget=30 to CodeTree + GPT-4o, it outperforms o1-preview on all benchmarks with −25% token cost, +1.9% pass@1

Contributions

- We introduce CodeTree, an efficient, scalable framework of agent-guided tree search for code generation tasks.
- CodeTree achieves SoTA performances compared to existing agentic code generation methods or frontier closed reasoning model.
- CodeTree can be applied to other framework like Agentless (Xia et al., 2024), increase its performance on repo-level code generation like SWEBench (Jimenez et al., 2024).

Outline

- Introduction: Competitive Programming, CodeGen

- Explaining CP problems
- Distilling Algorithmic Reasoning Via Explaining Solutions
- CodeTree: Agent-guided Tree search for CodeGen
- AlgoSimBench: Identifying Algorithmically Similar CP problems
- InstEmbed: Instruction-Sensitive Code Embeddings

- Future Work & Conclusion

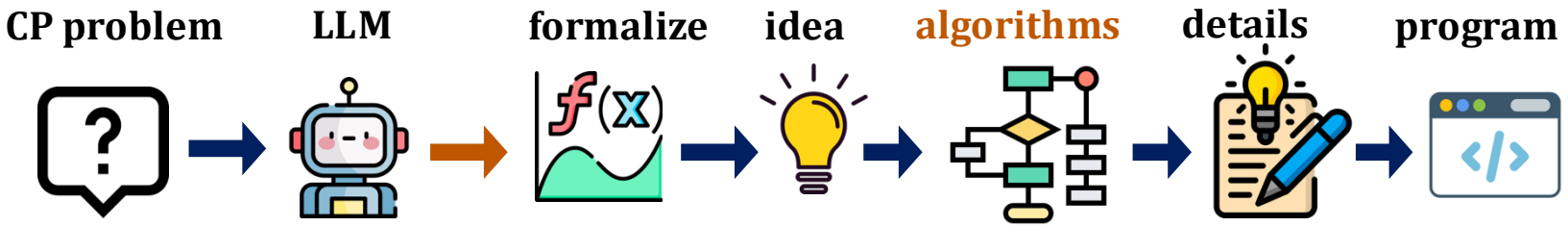
AlgoSimBench: Identifying Algorithmically Similar CP problems

(Li and Mooney, IJCAI 2026)

RQ5: Can LLMs trained to solve problems identify similar problems?

The Myth of Generalizing Algorithmic Reasoning

Identifying which algorithm to use seems to be an upstream task to problem solving.



Can this knowledge of algorithms generalize to another task in the same domain?

Semantic-Adversary Multi-Choice Question to Identify Algorithmically Similar Problems

MCQ: Given a competitive programming problem, your task is to select the one with the most similar algorithm topics, tricks, and ideas to solve it.

Here's the problem: 

Algorithmically similar problem ← Different background story, low textual overlap

4 choices: ABCD

Algorithmically dissimilar distractor ← Similar background story, high textual overlap

Data Curation

- Problem Algorithm Labels:
  Four Competitive-Programming Online Discussion Forums
- Problem Sources: Three practice websites
- Semantic-adversary: Distractors textually similar
  Correct option is textually dissimilar, filtered by Embedding model.
- Human-Verify: filter out False Distractors

402 MCQs with 903 distinct problems, 231 distinct fine-grained algorithm tags. CodeContests only has 40 very general tags.

Method: Attempted Solution Matching(ASM)

Instead of comparing problem choices with the reference problem, we use "Attempted Solution" to match.

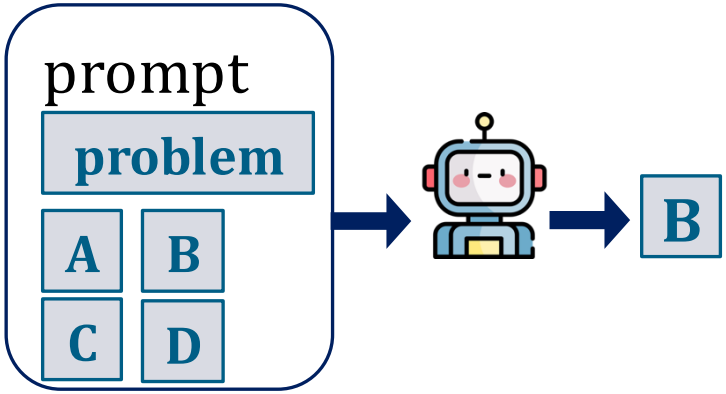
- Generate a first-attempt solution to 4 choices & the problem in the question.
- Replace the problems with the their first-attempt solutions, then let the model select the algorithmically-best-match.

Experimental Settings

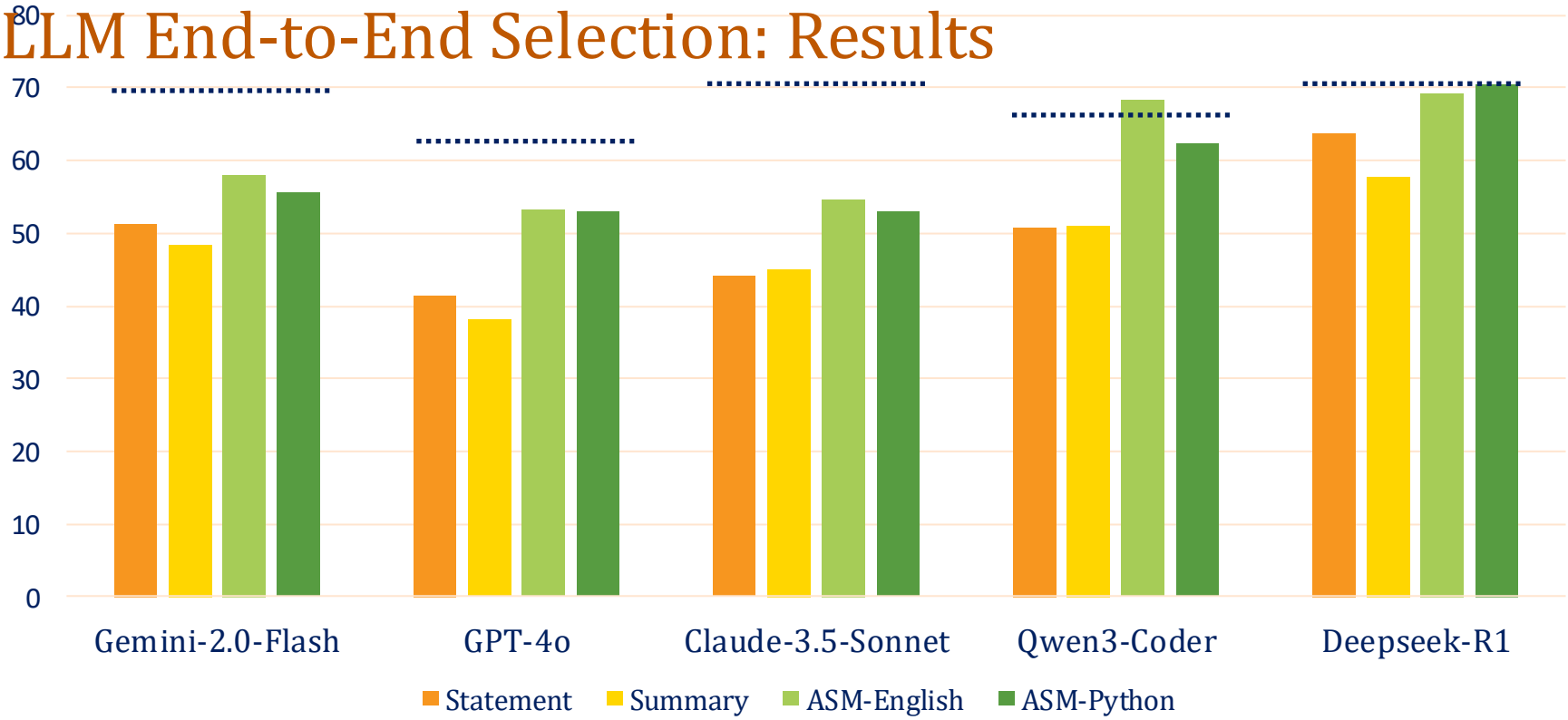
- Dataset: AlgoSimBench 402 MCQs
- Evaluation metric: accuracy of select the correct choice
- Baselines, replace the problem statement with
  - Problem Statement
  - Summary of the Problem
  - Human Oracle Solution (Code)
  - ASM-English: Describe how to solve it in English
  - ASM-Python: Give the Python solution

Experimental Settings

- LLM End-to-End Selection



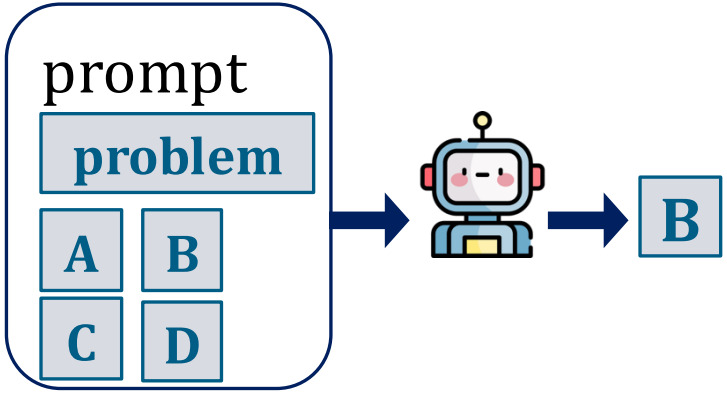
LLM End-to-End Selection: Results



Evaluate models' performances (%) of MCQ are correct.
..... Solution*: Use human-written correct solution for options and given problem.

Experimental Settings

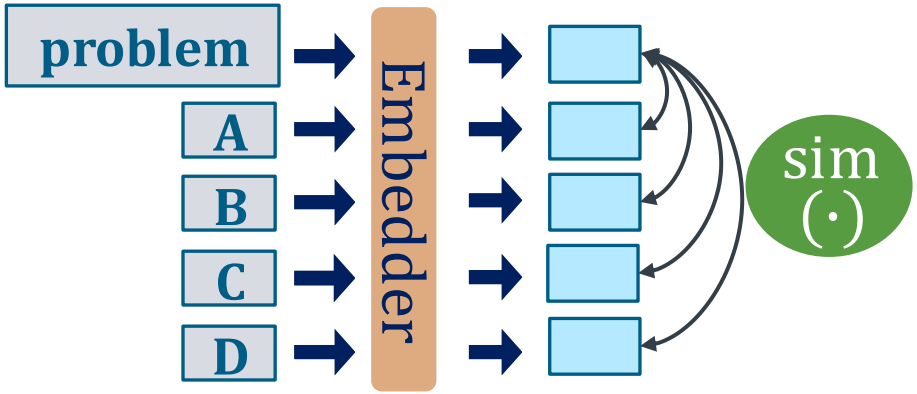
- LLM End-2-End Selection



Not scalable

- ❖ Can only handle AlgoSimBench with limited number of choices
- ❖ Slow inference

- Retrieval-based Selection



Scalable to large corpus

- ❖ Each problem in the context only process once
- ❖ Unbounded by num of choices

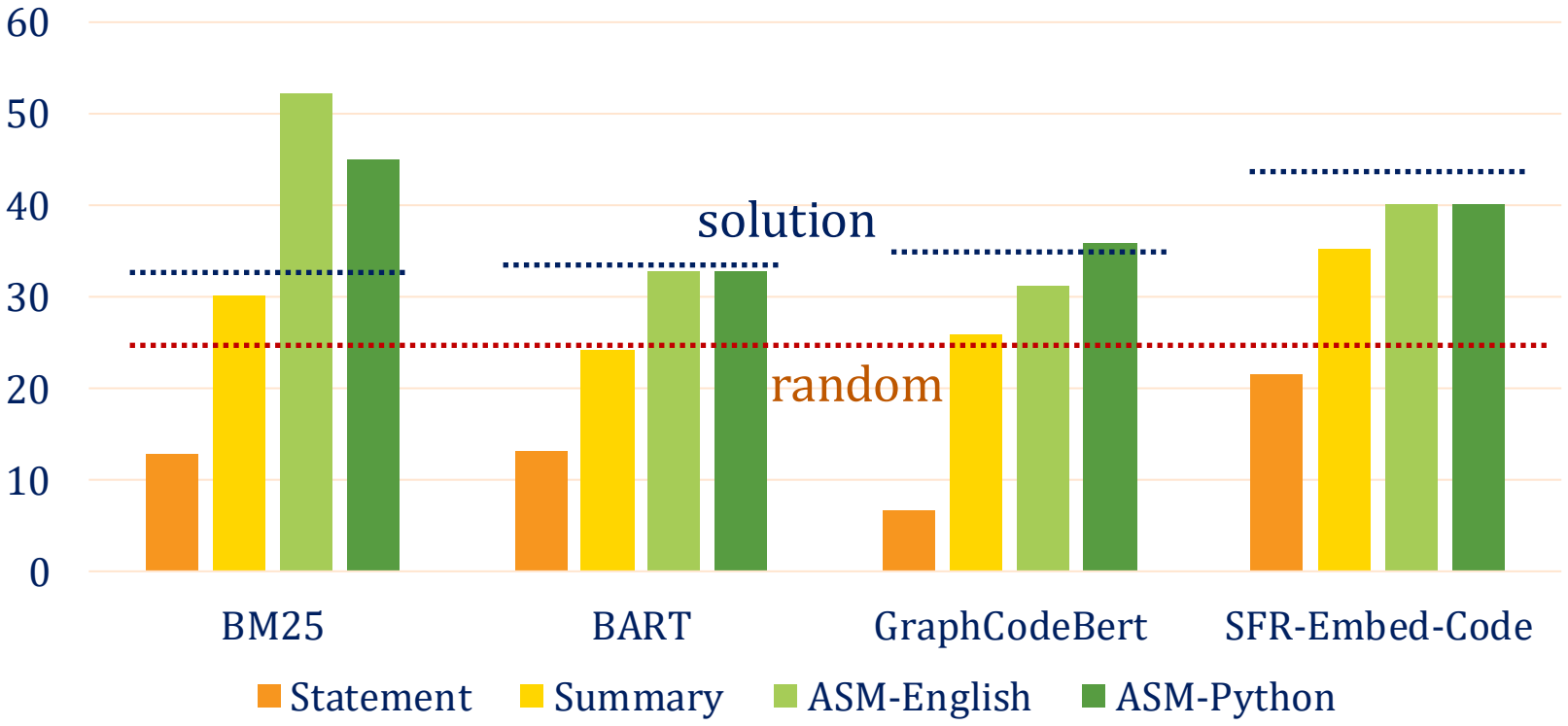
Retrieval-based Selection

Experimental Setting: Deepseek-R1 to generate summary, ASM-English, ASM-Python

Text Embedding Models:

Model	Method Type	Parameters
BM25 [Trotman et al., 2014]	Keyword-based ranking	b=0.75, k1=1.2
BART [Lewis et al., 2020]	Dense text embedding model	# 139M
GraphCodeBert [Guo et al., 2021]	Dense code embedding model	# 125M
SFR-Embed-Code [Liu et al., 2024]	Dense code mmbedding model	# 400M

Retrieval-based Selection



Contributions:

- We introduced AlgoSimBench, a novel benchmark designed to evaluate models' ability to reason about algorithmic similarity between competitive programming problems.
- AlgoSimBench can serve as a benchmark to evaluate LLMs (LLM End2End-Selection), or retrieval methods (Retrieval-based Selection)
- We propose Attempted Solution Matching, leveraging LLM-generated first-attempt solutions, achieving promising results in both settings.

Outline

- Introduction: Competitive Programming, CodeGen

- Explaining CP problems
- Distilling Algorithmic Reasoning Via Explaining Solutions
- CodeTree: Agent-guided Tree search for CodeGen
- AlgoSimBench: Identifying Algorithmically Similar CP problems
- InstEmbed: Instruction-Sensitive Code Embeddings

- Conclusion & Future Work

InstEmbed: Instruction-Sensitive Code-Embeddings via Task-Contrastive Training

(Li and Mooney, 2026, under review)

Can embedding models follow the instruction to retrieve requested target?

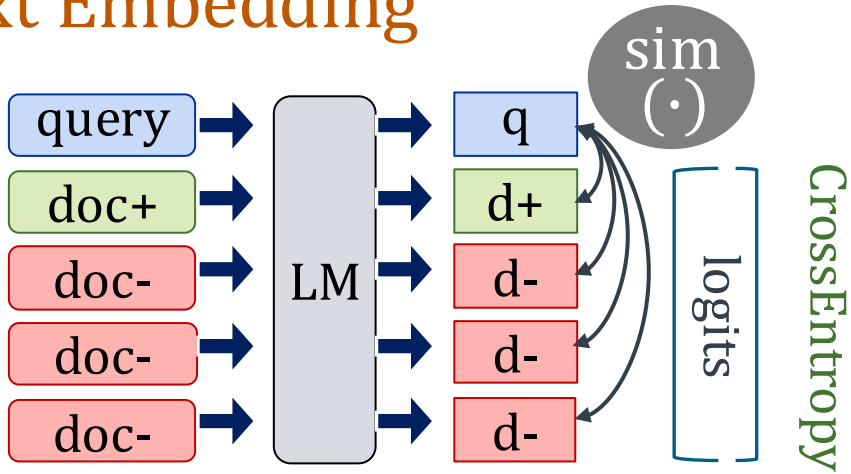
Background: Retrieval & Text Embedding

Task: Find the code snippet in Python with the same functionality.

Query:  code of Fibonacci

Document:  code of Fibonacci

Corpus: A set of documents



In-batch Negatives: documents in the training batch

Hard Negatives: relevant document but not the target

SimCSE: Simple Contrastive Learning of Sentence Embeddings (Gao et al., 2021)

Information Noise-Contrastive Estimation (InfoNCE[1])

$$\mathcal{L} = -log \frac{\exp(cos(q, d^+)/\tau)}{\sum_{d \in \mathcal{D}} \exp(cos(q, d)/\tau)}$$

$$\mathcal{D} = \{d^+\} \cup \mathcal{D}^-_{batch} \cup \mathcal{D}^-_{semantic}$$

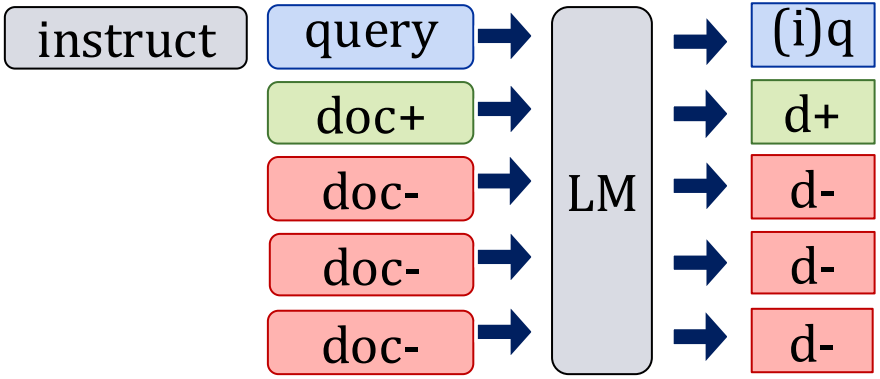
positive

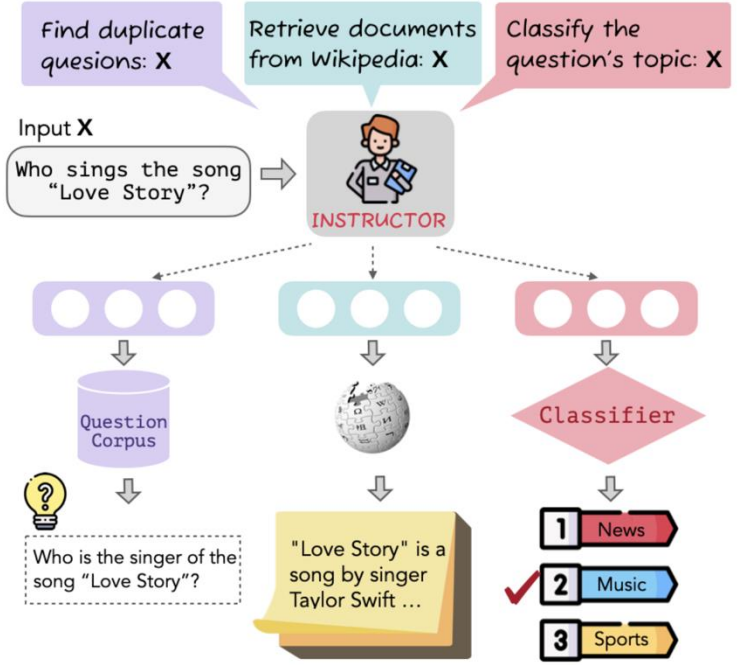
in-batch negative

mined hard negatives

[1]Representation Learning with Contrastive Predictive Coding (Oord et al., 2018)

Background: Task-prefixed Multi-task Text Embedding

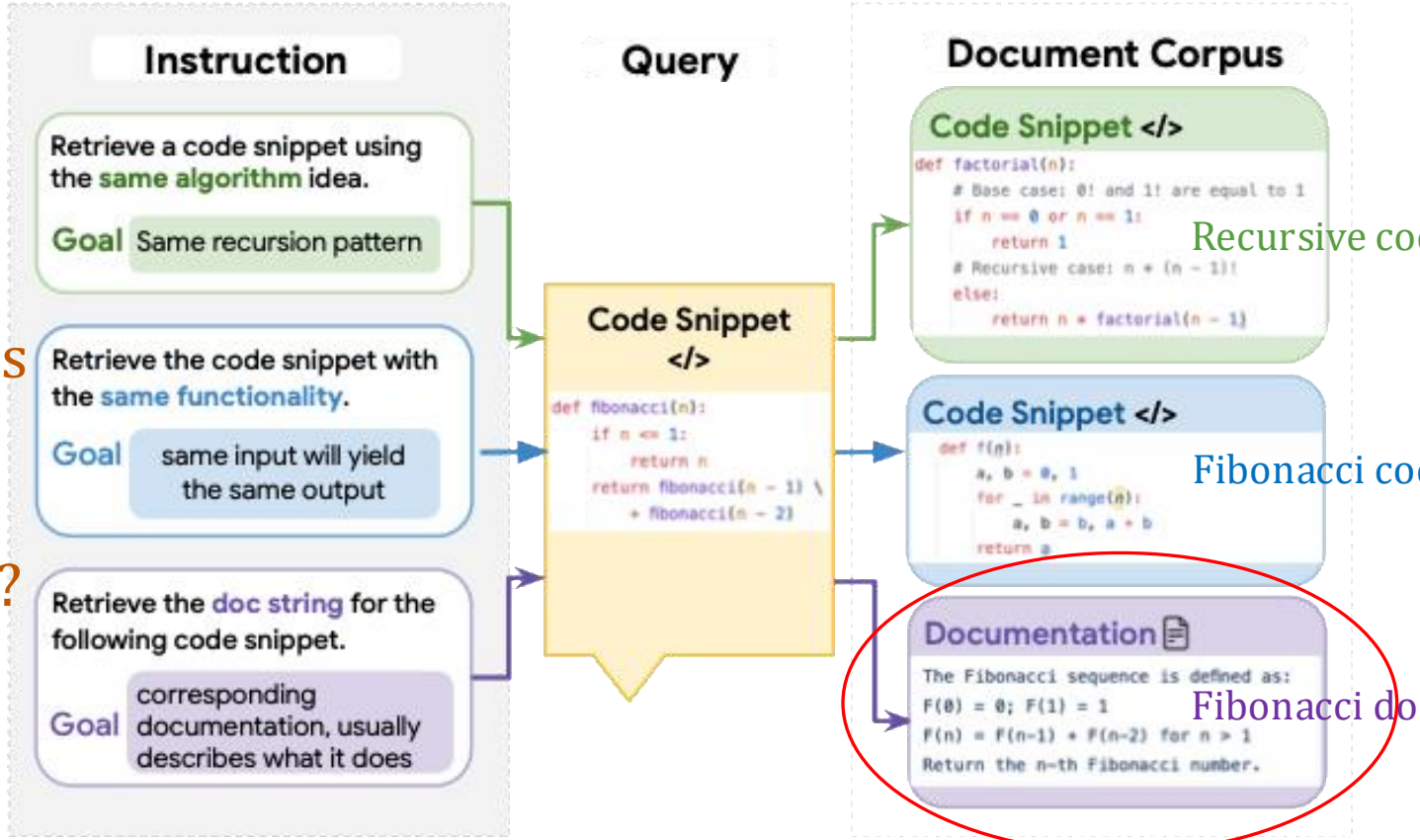




Do they exhibit the capacities to follow instructions, even simple ones?

One Embedder, Any Task: Instruction-Finetuned Text Embeddings (2022)

Why Instrution-sensitivity is crucial for code embedding?



[1]Towards A Generalist Code Embedding Model Based On Massive Data Synthesis (Li et al., 2025)
[2]Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models (Zhang et al 2025)

Information Noise-Contrastive Estimation

$$\mathcal{L} = -log \frac{\exp(cos(q, d^+)/\tau)}{\sum_{d \in \mathcal{D}} \exp(cos(q, d)/\tau)}$$

target documents to the same query but under different tasks (instructions)

$$\mathcal{D} = \{d^+\} \cup \mathcal{D}^-_{batch} \cup \mathcal{D}^-_{semantic} \cup \mathcal{D}^-_{task}$$

- d^+: positive
- $\mathcal{D}^-_{batch}$: in-batch negative
- $\mathcal{D}^-_{semantic}$: mined hard negatives
- $\mathcal{D}^-_{task}$: task-contrastive negative

Task-Contrastive Data Construction

Multi Modality Target Retrieval

Query-Query Retrieval

Query \ Document	positive	negative-1	negative-2
q	d-1	d-2	d-3
q	d-2	d-1	d-3
q	d-3	d-1	d-2

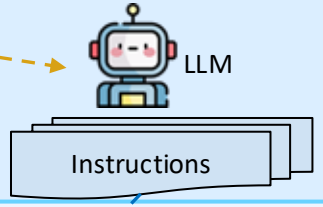
Instruction Parameterization

Target Object What's the modality of the target document?

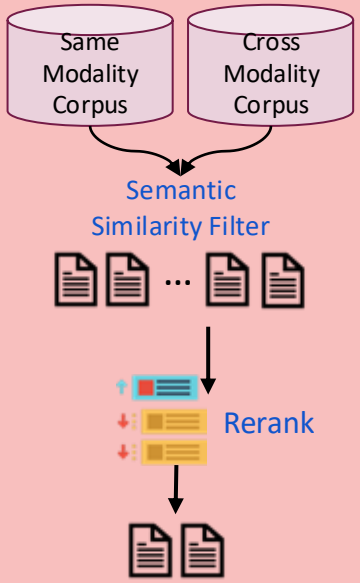
Define Similarity Specify underlying relationship required between query & target. same programming language; same framework; same functionality…

Focus Guide Instruct on where to focus, context to ignore.

Instruction Generation



Hard Negative Mining



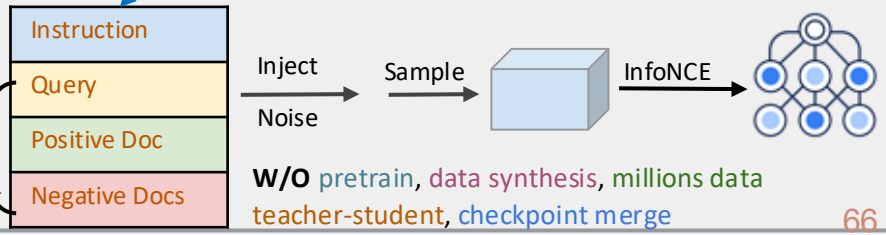
Data Filtering

Data Cleaning & Normalization

Lexical Quality Check

task, q, d+, d- → LLM

Instruction Sensitive Noisy Training



W/O pretrain, data synthesis, millions data teacher-student, checkpoint merge

Evaluation Benchmark

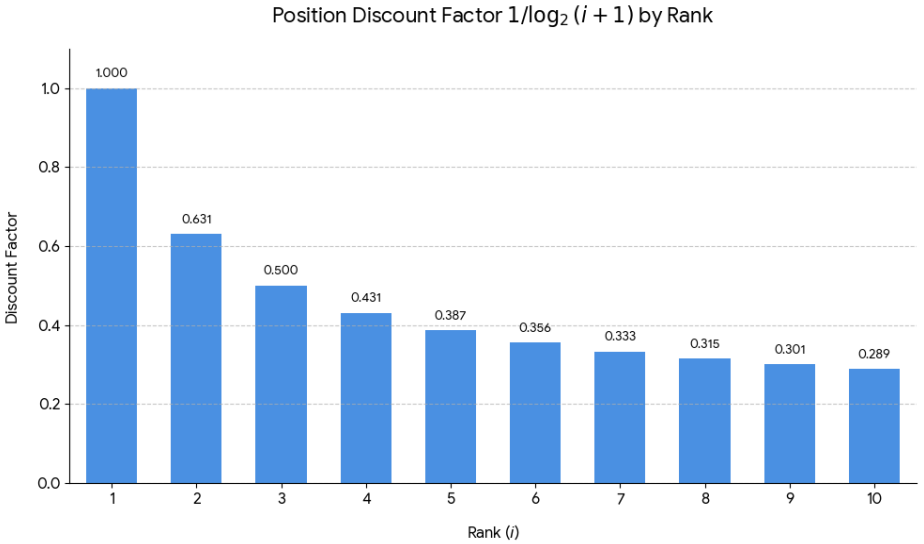
- MTEB-Code (12 tasks + optional tasks)
  MTEB: Massive Text Embedding Benchmark (Muennighoff et al., 2023)
  - CodeEditSearch: Github commit msg => diff patch
  - CodeSearchNet: Implement instruction => code
  - CodeSearchNet-CodeCompletion: first half of code => second half
  - CodeTranslate-Contest: python code => java code
  - StackOverflowQA: question => answer (from StackOverflow)
  - WikiSQL: natural language query => SQL query

CoIR: A Comprehensive Benchmark for Code Information Retrieval Models. (Li et al., 2024)

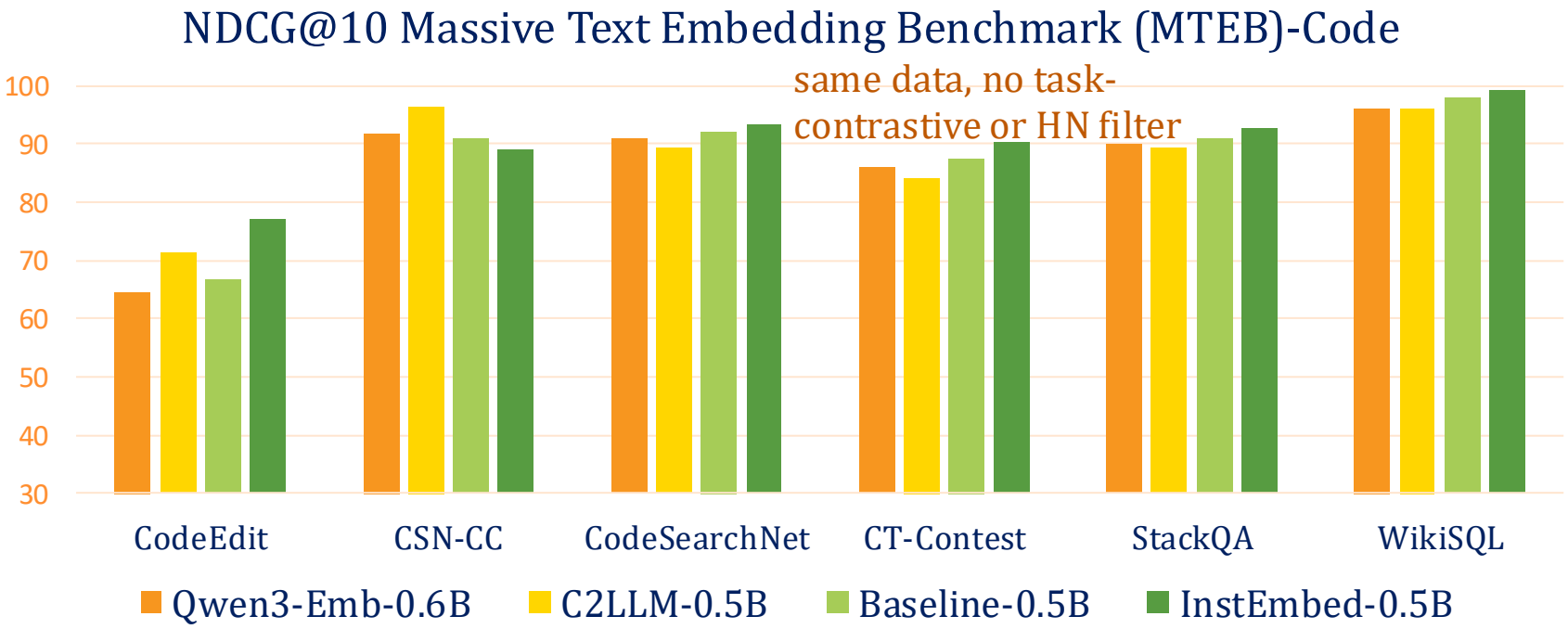
Evaluation Metrics

- Normalized Discounted Cumulative Gain at 10 (NDCG@10):

A metric used to measure the ranking quality of a search engine or recommendation system, evaluated across the top 10 returned results.

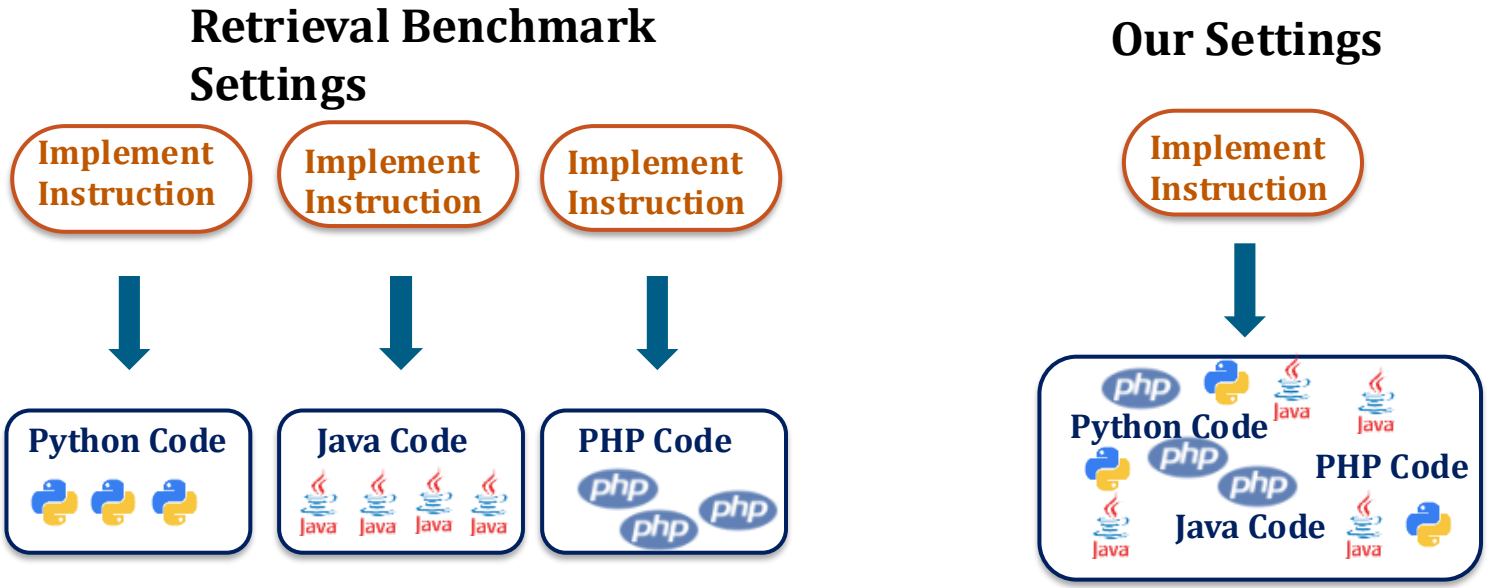


Code Retrieval Performance



C2LLM Technical Report (2025); Qwen3-Embedding Technical Report (2025);

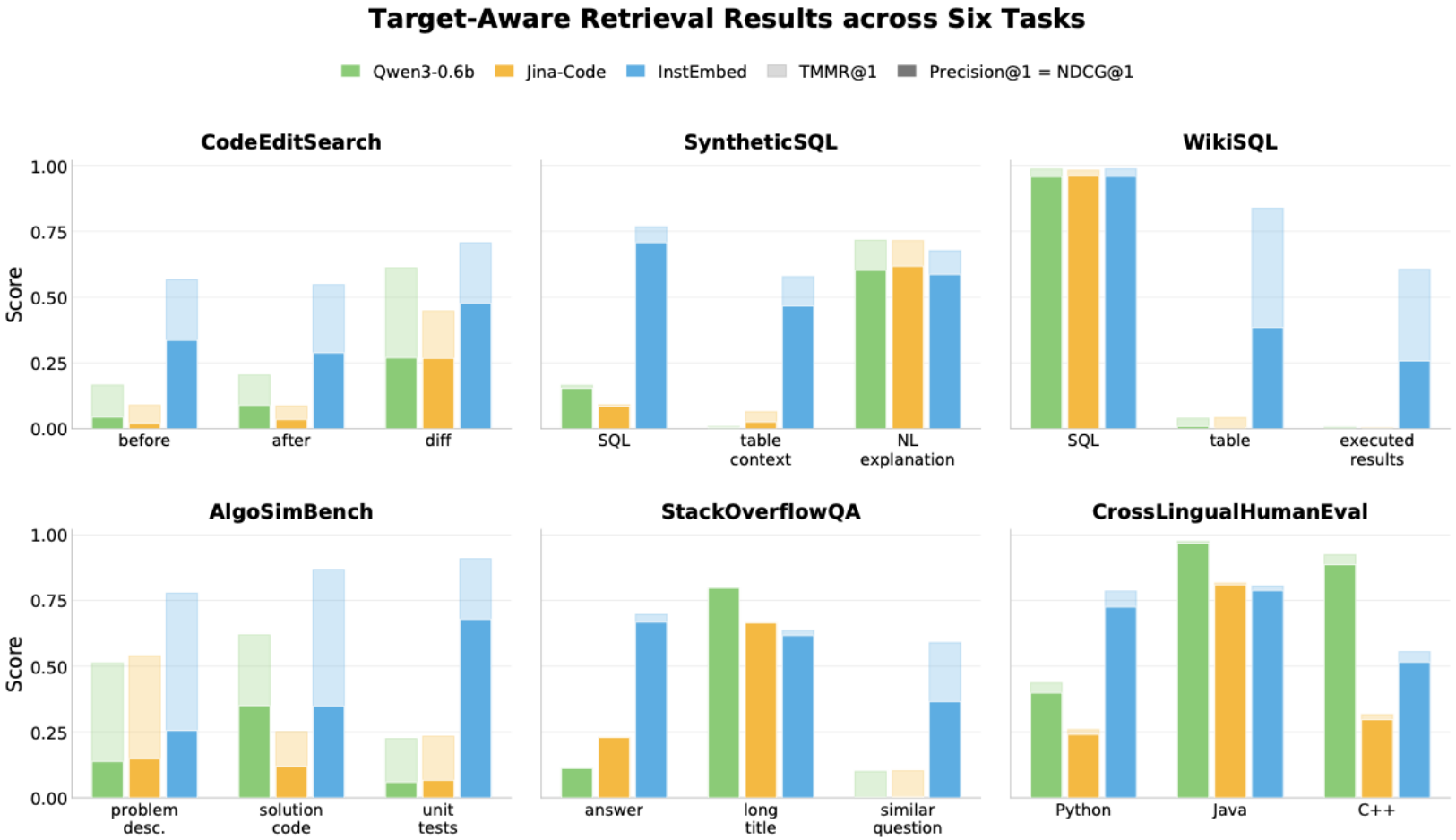
Target-aware Diagnostic Test



Target-aware Diagnostic Test: Evaluation

- Task Setting:
  Mixing corpus with different types of target, each target type is a modality. We evaluate the ability to retrieve the correct document and to retrieve the correct modality of target.

- Precision@1: how many of top 1 matched the correct document
- Top-1 Target Modality Matching Rate (TMMR@1), which calculates the percentage of queries where the top-retrieved document correctly belongs to the modality requested by the instruction.


Target-Aware Retrieval Results across Six Tasks
Qwen3-0.6b
Jina-Code
InstEmbed
TMMR@1
Precision@1 = NDCG@1
CodeEditSearch
SyntheticSQL
WikiSQL
AlgoSimBench
StackOverflowQA
CrossLingualHumanEval
Score
1.00
0.75
0.50
0.25
0.00
before
after
diff
SQL
table context
NL explanation
SQL
table
executed results
problem desc.
solution code
unit tests
answer
long title
similar question
Python
Java
C++

Contributions

- We found that several SoTA text and code embedding models are poor at following the requirements in the instruction prefixed to query.
- We propose InstEmbed, a code embedding model that identifis the target, focus, and definition in instructions to retrieve the target document, being instruction-sensitive.
- InstEmbed performs on par with SoTA open models of similar sizes and can retrieve different targets for the same query as instructed. It is potentially applicable to modern code search where retriever faces complex environment and corpus.

Outline

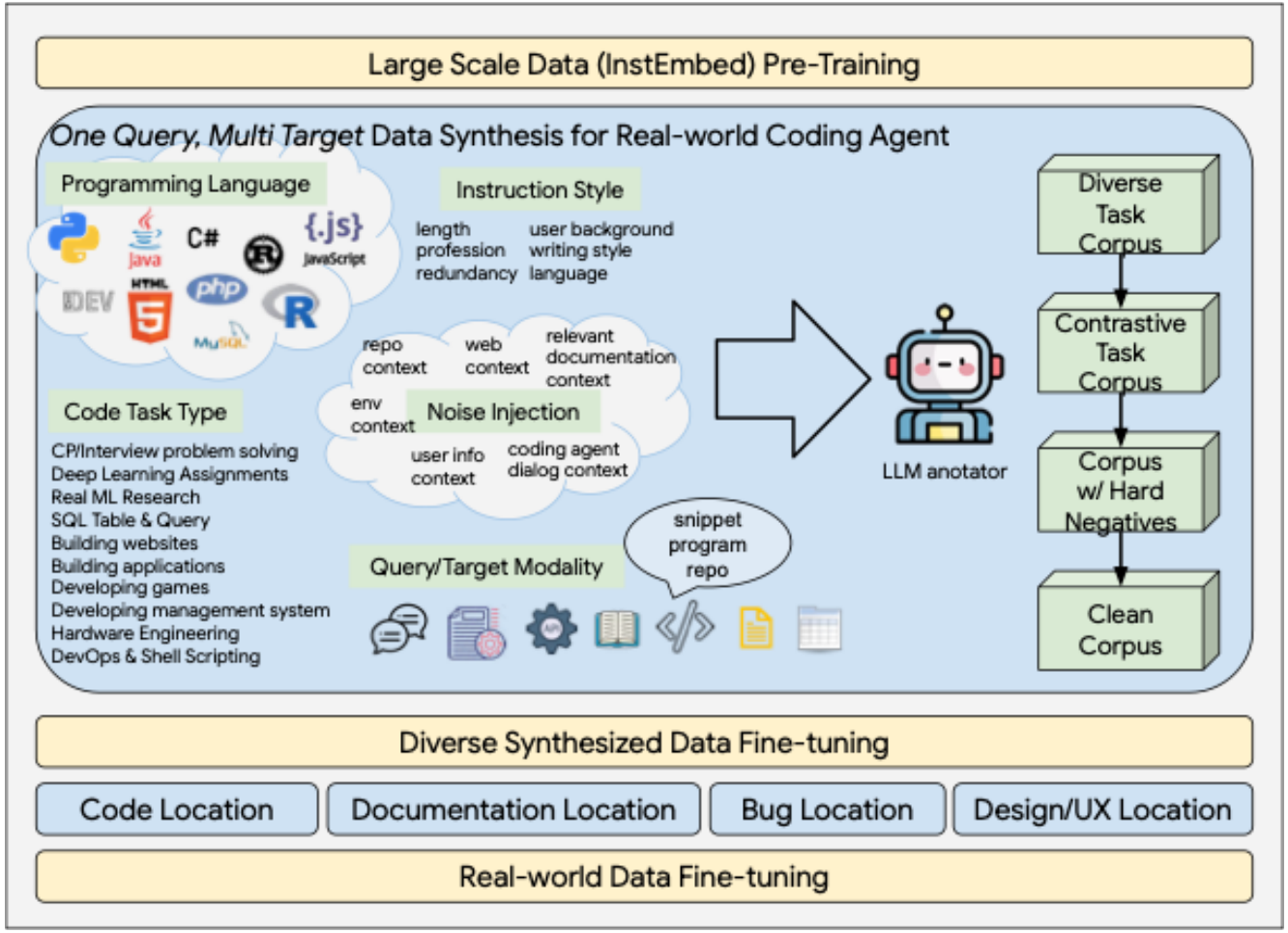
- Introduction: Competitive Programming, CodeGen

- Explaining CP problems
- Distilling Algorithmic Reasoning Via Explaining Solutions
- CodeTree: Agent-guided Tree search for CodeGen
- AlgoSimBench: Identifying Algorithmically Similar CP problems
- InstEmbed: Instruction-Sensitive Code Embeddings

- Future Work & Conclusion

Future Work:

- Extend InstEmbed to Real-world code search application by synthesizing thousands of retrieval tasks.
- Multi-Facet Embeddings via Instruction-Transformed Embedding for Code Search
- Training Code Embeddings from Downstream Code Generation Feedback



Learning Real-world code search application.

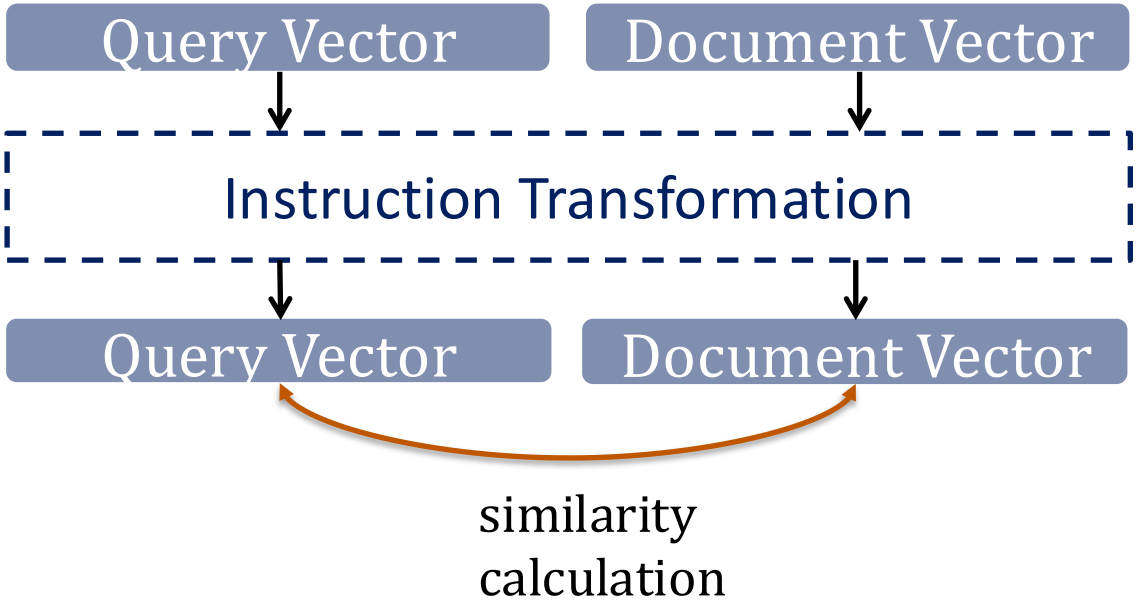
Future Work: Multi-Facet Embeddings via Instruction as Transformation

You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!

Ray Mooney

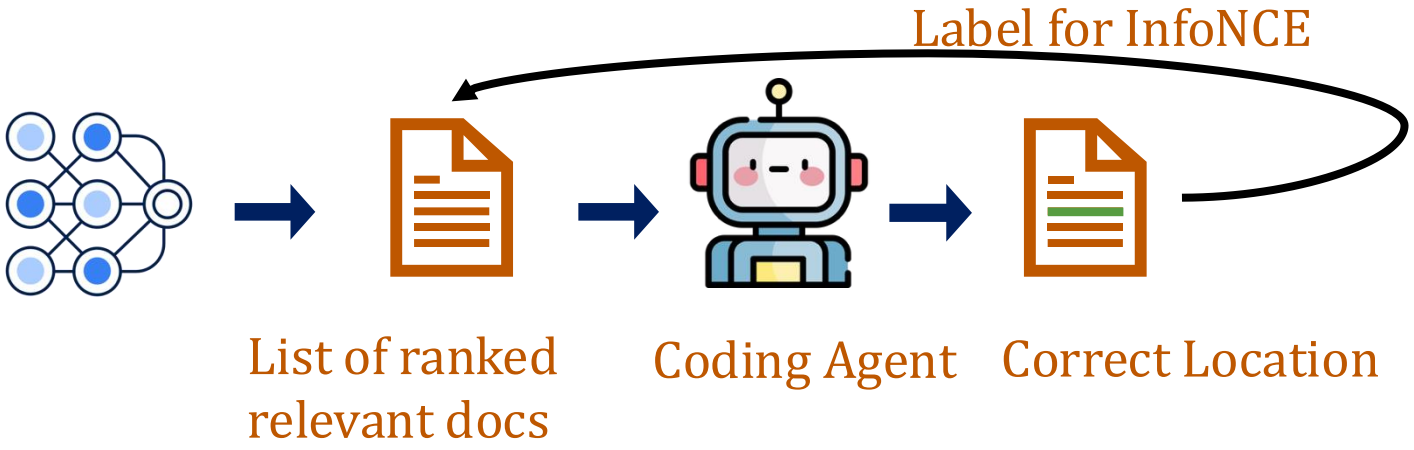
What about one vector + one transformation?

Future Work: Multi-Facet Embeddings via Instruction as Transformation



Future Work: Training Code Embeddings from Downstream Code Generation Feedback

- Train an embedding model with a code generation model to give feedback.



Conclusion

This defense introduces works on investigating, enhancing code generation for competitive programming by utilizing natural language as a bridge for reasoning.

Investigate: Explaining CP solutions

Enhance: Distilling Explanations; CodeTree

Empower the System: AlgoSimBench; InstEmbed

Conclusion

We present findings and insights in code explanation, generation, and representation, making progress in reasoning distillation, agentic framework for code generation, algorithmic similarity identifying, instruction-sensitive code embedding.

We believe this progress can support future research in this domain, and eventually lead to an automatic, intelligent, and controllable software engineering system to assist human engineers.

Other Works when @ UT

- ContraDoc: Understanding Self-Contradictions in Documents with Large Language Models (NAACL 2024)
  Jierui Li, Vipul Raheja and Dhruv Kumar

- Learning to Reason Deductively: Math Word Problem Solving as Complex Relation Extraction (ACL 2022)
  Zhanming Jie, **Jierui Li** and Wei Lu

Acknowledgement

other labmates, faculties, research peers...

Question & Discussion