SEP 2025

TEXAS
The University of Texas at Austin

ENHANCING COMPETITIVE-LEVEL CODE GENERATION BY UTILIZING NATURAL LANGUAGE REASONING

Ph.D. Thesis Proposal

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

PRESENTER: JIERUI LI
Ph.D. Student, The University of Texas at Austin, Since 2021

Outline

- Introduction & Related Works
- Completed Works
- Proposed Future Works

Outline

- Introduction & Related Works
  - Reasoning Tasks with Large Language Models
  - Gap between Natural Language Reasoning and Symbolic Language Reasoning
  - Competitive-Level Programming in the Era of LLMs
- Completed Works
- Proposed Future Works

Emergence of Large Language Models(LLMs)

Better Text Understanding


Generate Fluent Text

T5, GPT

Complete NLP tasks by generation

Codex FLAN

Complete Complex Tasks

Reasoning Tasks with Large Language Models

Natural Language Reasoning(NLR): Common Sense Reasoning, Reading Comprehension, Multi-hop Question Answering, Textual Entailment Recognition, ...

Symbolic Reasoning(SR): Math Word Problem Solving, Logical Deduction, Code Generation, Automatic Program Repair, Knowledge-Graph QA, ...

Natural Language Reasoning V.S Symbolic Reasoning

LLMs rely heavily on semantics in tokens and contexts, and struggle more when semantics are inconsistent or when symbolic/counter-commonsense reasoning is needed.

[1]Gendron, Gaël, et al. "Large language models are not strong abstract reasoners." *arXiv preprint arXiv:2305.19555* (2023).

Natural Language Reasoning V.S Symbolic Reasoning

Using Natural Language to explicitly describe the chain-of-thought(CoT) in the context can enhance models' abilities to do both Natural Language and Symbolic Reasoning Tasks.

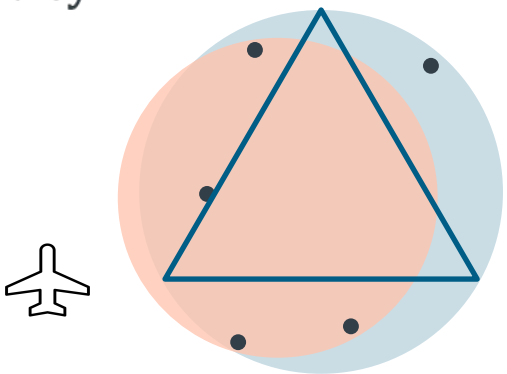
[2] Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." Advances in neural information processing systems 35 (2022): 24824-24837.

Competitive-Level Programming in the Era of LLMs

Some Brain Teaser

[Max Cities To Destroy]

- You must place N cities as points inside a circular territory of radius 2. An enemy can drop a single bomb of radius $\sqrt{3}$, The enemy can see your city placement. If you place your cities to protect as many as possible, what is the maximum number of cities the enemy can destroy?



```
def max_city(n):
    return math.ceil(n/3)
```

Challenges in Competitive Programming Problems

Many times, the chain-of-thought process is not reflected directly in the solution.

Human-written editorials are hard to obtain at scale.

How to better reason CP problems when <problem, solution-program> pairs are all we have?

Enhancing Competitive-level Code Generation by Utilizing Natural Language Reasoning

natural language chain-of-thought → solution (code)

Outline

- Introduction & Related Works
- Completed Works
  - Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs:
    Explaining competitive-level programming solutions using llms (NLRSE 2023)
    Distilling algorithmic reasoning from llms via explaining solution programs (NLRSE 2024)
  - CodeTree: Agent-guided Tree Search for Code Generation with Large Language Models (NAACL 2025)
  - AlgoSimBench: Identifying Algorithmically Similar Problems for Competitive Programming (Under Review)
  - My Other Works@UT: Learning to Reason Deductively (ACL 2022); ContraDoc (NAACL 2024)
- Proposed Future Works

Can LLMs learn the algorithmic reasoning needed for competitive-level programming task?

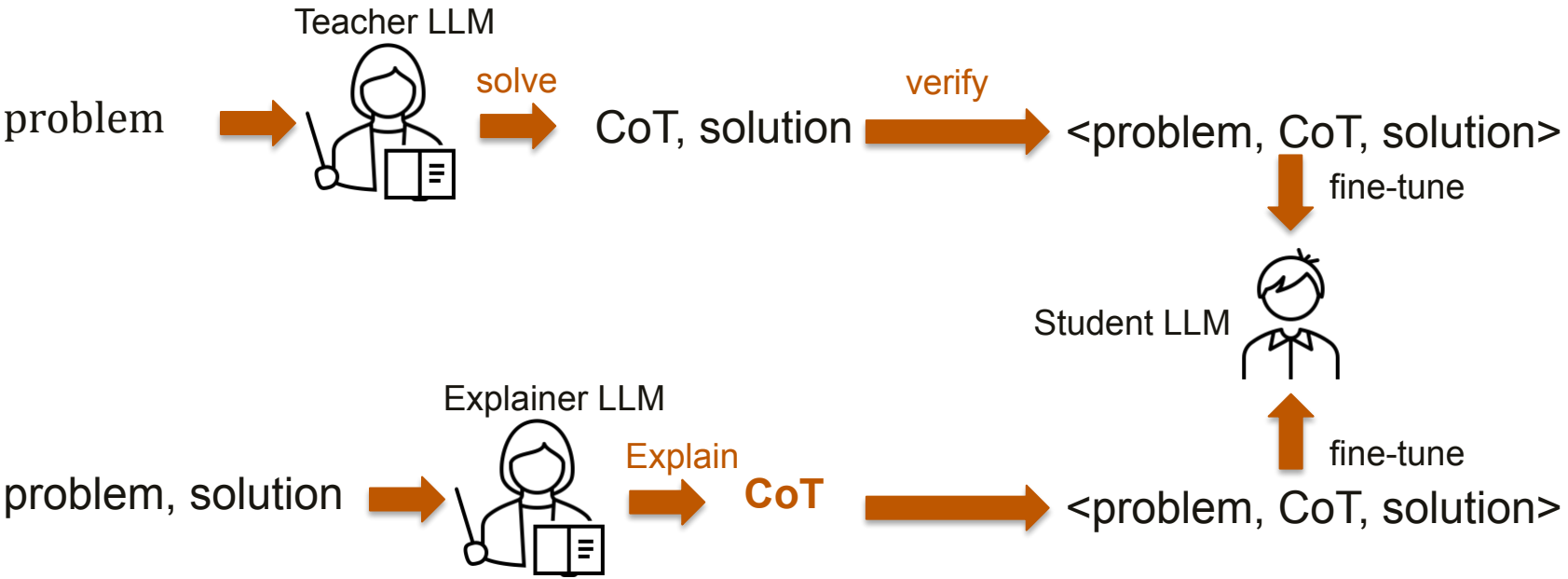
- Editorial: a comprehensive explanation or guide that discusses the problems presented in a programming contest or challenge, which often contains some of the following aspects.
  - Problem restatment
  - Difficulty
  - Prerequisites
  - Quick explanation
  - Explanation
  - Code
  - Pitfalls(what to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

LLMs Reasoning Distillation

GPT-4 performs poorly in solving problems from the training set to yield a set of effective CoTs.

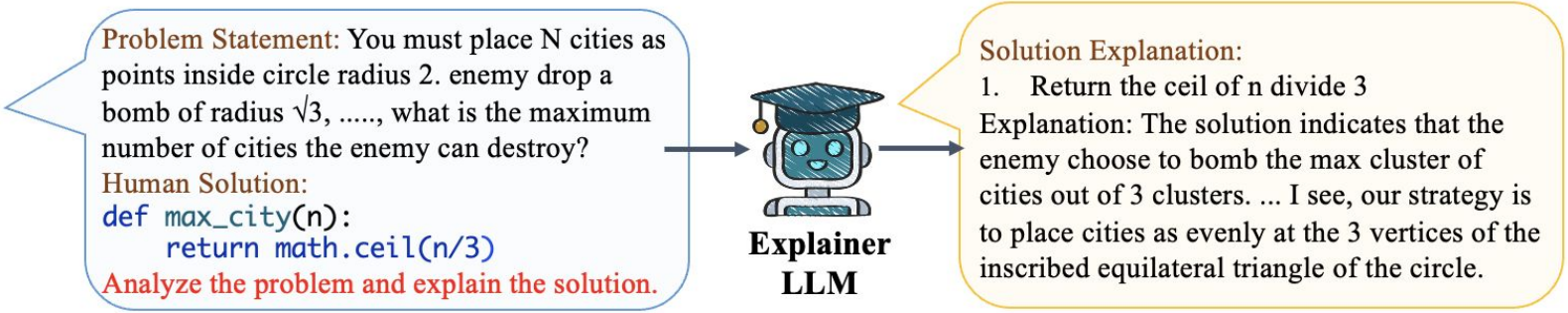
[3]Hsieh, Cheng-Yu, et al. "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes." *arXiv preprint arXiv:2305.02301* (2023).

Distill Explaining as Reasoning



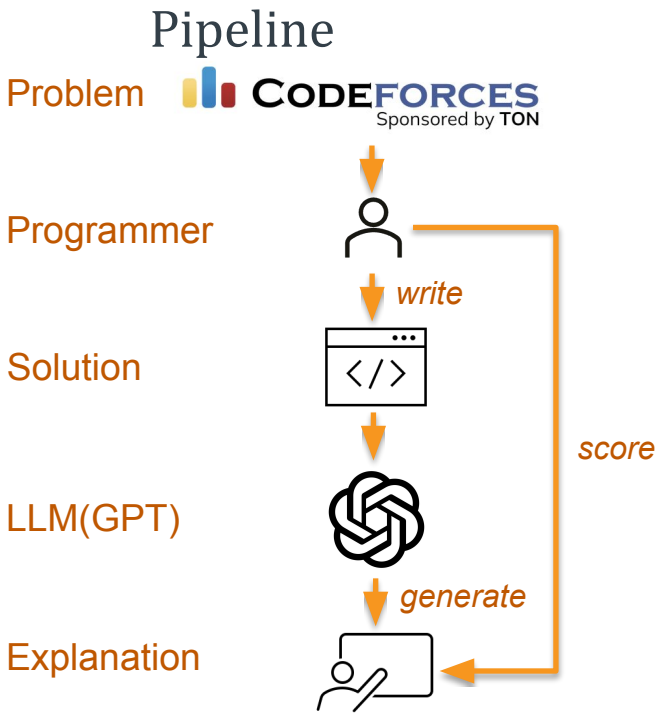
Methodology

<Problem, Human Solution> □ Editorial-style explanation of solution



*A simplified illustration

Human Evaluation of Explanations



Likert Score on Different Aspects:

(1)Brief Problem Summary

(2)Used Algorithm

(3)Step-by-Step Description

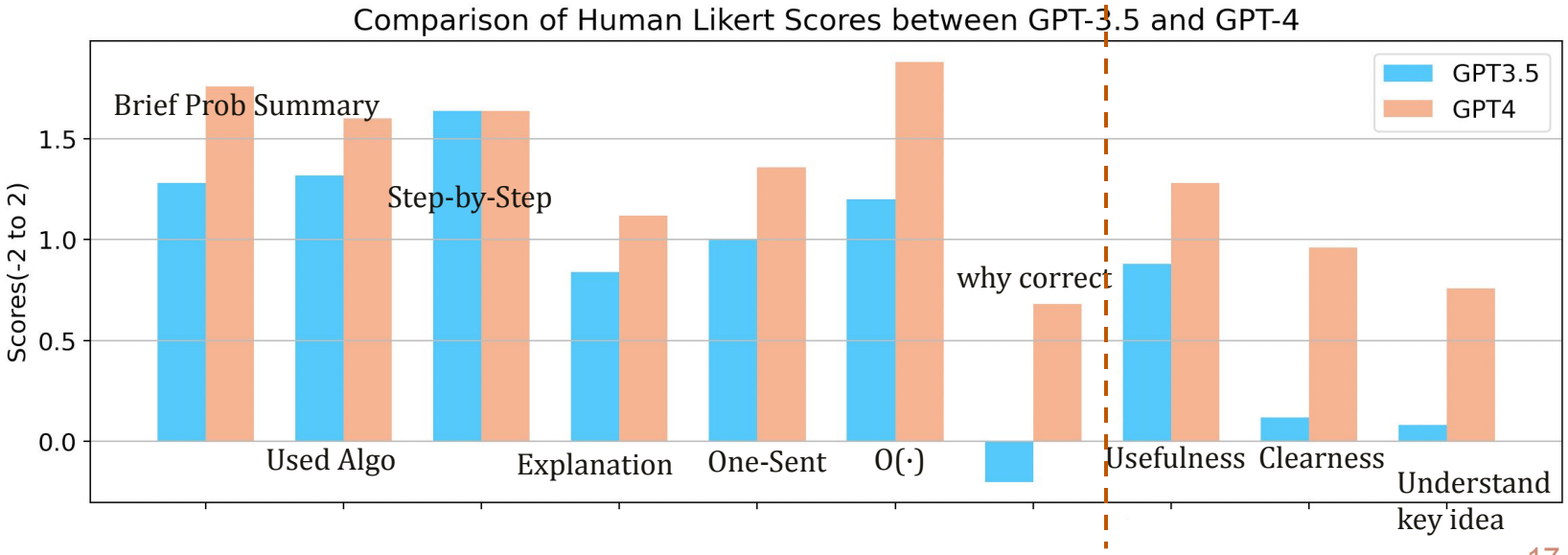
(4)Explanation of the Solution

(5)Time Complexity

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

Human evaluation of LLMs-generated explanations

Likert Scores of 50 problems with rankings from 800 to 2000.



Can LLMs learn from silver(generated) explanations?

Why not directly learn from problem to code?

Learning from<problem, human solution> pair is not effective.

- Practically: AlphaCode fine-tuned on over 12M pairs, but was only to obtain a solve rate of 15.6% when sampling 1000 programs.
- Reasons:
  - Solutions were written given time constraints, readibility is poor.
  - Diverse in coding style, programming languages.
  - Solution alone don't contain the reasoning process preceding problem solving.

Can LLMs learn from silver explanations?

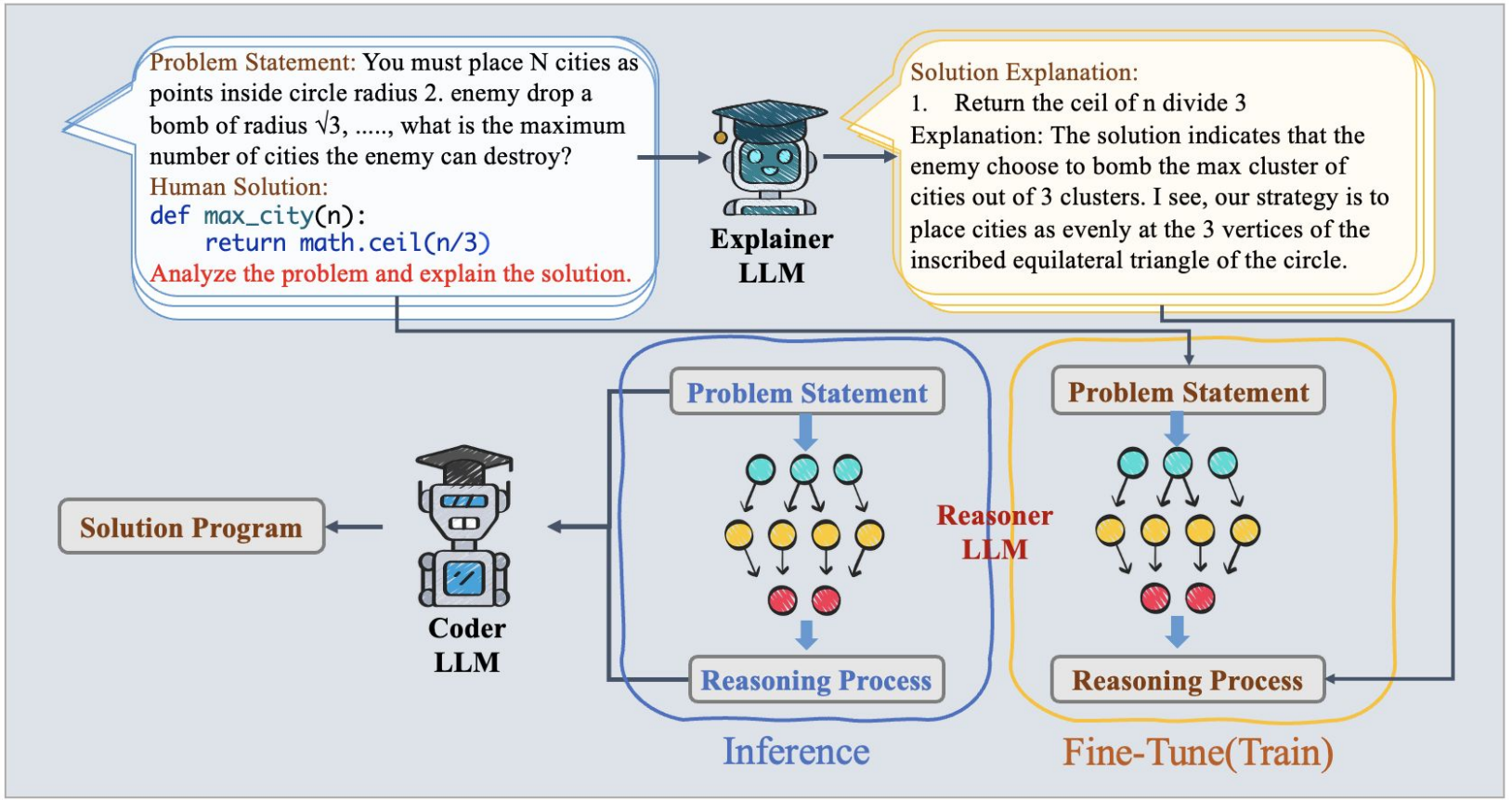
- Natural language explanations, on the other hand, semantic rich, containing the reasoning process preceding implementation, should be more effective for models to learn reasoning abilities.

- **If generated at scale, can silver explanations be used as a source to improve subsequent problem-solving?**

Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs

Instead of learning from the <problem, solution-program> pairs, we propose to learn from hierarchical, detailed, and semantic-rich explanations(Given by Explainer LLM) of them.

Our framework contains a Reasoner LLM to learn the problem-solving reasoning process from automatically annotated editorial-style explanations and a Coder LLM to implement the solution given the verbal solution from the Reasoner.


Problem Statement: You must place N cities as points inside circle radius 2. enemy drop a bomb of radius √3, ..., what is the maximum number of cities the enemy can destroy?
Human Solution:
def max_city(n):
return math.ceil(n/3)
Analyze the problem and explain the solution.
Explainer LLM
Solution Explanation:
1. Return the ceil of n divide 3
Explanation: The solution indicates that the enemy choose to bomb the max cluster of cities out of 3 clusters. I see, our strategy is to place cities as evenly at the 3 vertices of the inscribed equilateral triangle of the circle.
Problem Statement
Reasoning Process
Reasoner LLM
Problem Statement
Reasoning Process
Solution Program
Coder LLM
Inference
Fine-Tune(Train)

Dataset and Evaluation Metrics

Dataset Source: Codeforces problems after 2022, ensuring GPT-3.5/GPT-4 **hasn't seen** the problems.

We collect 246 problems with release date 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: Intruct 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 is **GPT-turbo-3.5.**
We also experimented with DeepSeek-Coder-6.7b (details in paper)

Editorial-Style Chain-of-thought (Explanation)

- Problem Restatement
- Conceptual Evolution
- Key to Solution
- Solution Description
- Step-by-Step Solution Explanation
- Common Pitfalls

Experimental Results

	solve@1	solve@5	solve@10
0-shot Coder	1.1%	2.7%	3.3%
0-shot Coder w/CoT	1.1%	2.7%	3.6%
Finetuned Coder	0.5%	0.8%	3.6%
0-shot Reasoner + Coder	1.2%	2.5%	3.3%
w/ Finetuned Reasoner (Full Exp)	1.1%	3.2%	4.9%
w/ Finetuned Reasoner (Best Exp)	1.1%	**3.7%**	**6.1%**

Full Exp: With every aspect in the editorial.
Best Exp: With single aspect in the editorial that can yield best performance(Step-by-Step)

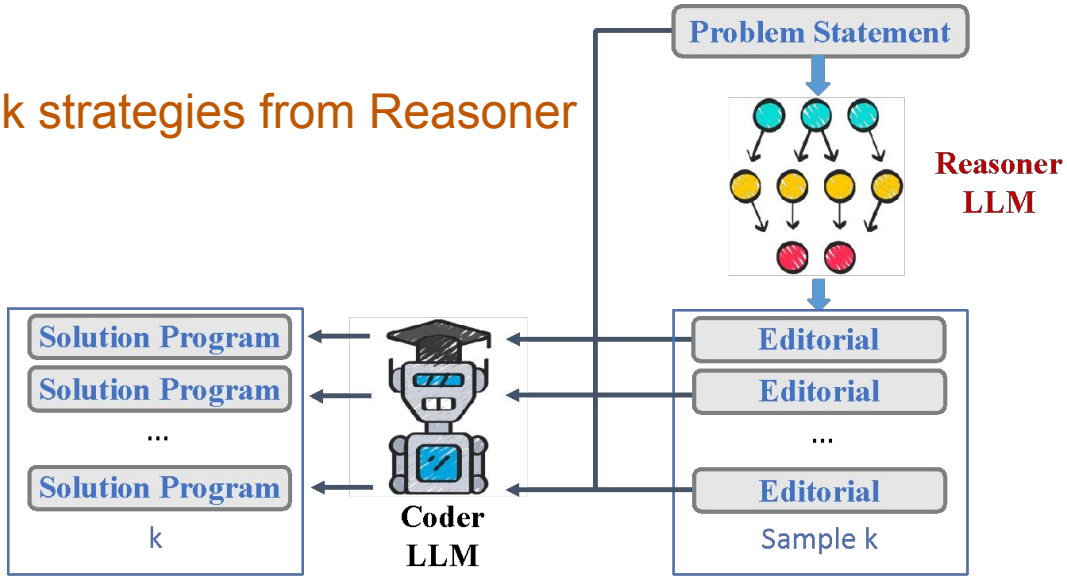
Does learning from explanations help avoid brute-force solution?

	Accepted	Wrong Answer	Time Limit Exceeded	Other
0-shot Coder	24.0%	18.3%	56.7%	1.0%
0-shot Coder w/CoT	23.4%	29.2%	42.5%	5.0%
0-shot Reasoner+Coder	24.3%	19.3%	52.9%	3.4%
w/ Full Explanation	42.1%	22.8%	29.8%	5.3%
w/ Step-by-Step Description	50.0%	19.2%	25.6%	5.8%
w/ Key Idea	35.5%	29.2%	32.3%	3.1%

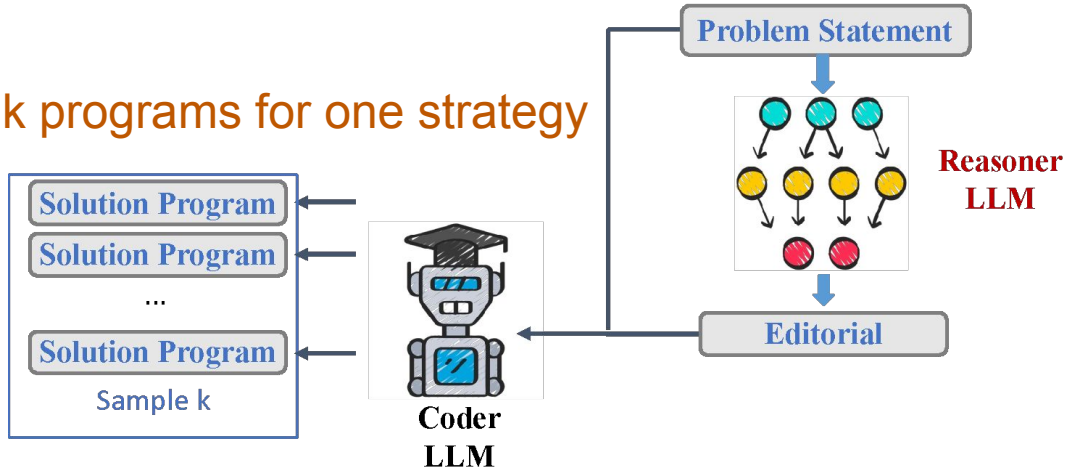
Final judgments from Codeforces of programs that pass public tests, cases rejected due to ineffeciency are largely reduced. (Other=Run time error/memory limit exceeded etc.)

RQ: Should we have diverse solving strategies or should we have different implementations of the most promising strategy?

Sample from Reasoner/Coder



Sample from Reasoner/Coder



Sample from Reasoner/Coder

Num_Sample from Reasoner	Num_Sample/edit from Coder	Solve@10
1	10	2.8%
2	5	3.7%
5	2	4.1%
10	1	4.9%

All experiments are with Fine-tuned Reasoner(Full Explanation), the temperature for sampling is 0.5

RQ: Should we have diverse solving strategies or should we have different implementations of the most promising strategy?

Ans: Sampling different chain-of-thoughts is more helpful than implementing different programs under the same chain-of-thoughts.

Conclusion from Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs

To tackle the challenging algorithmic reasoning task:

- Explaining solutions along with problems leverages LLMs' abilities in code comprehension, enable reasoning process distilling for problems beyond models' capacities.

- **Learning from** hierarchical, detailed, and semantic-rich **explanations** of <problem, solution-program> pairs is more effective than learning from them directly and can substantially improve LLMs' performance in solving these problems.

- Sampling more strategies allows distinct reasoning paths, which can improves solve@k when k>1

Conclusion from Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs

To tackle the challenging algorithmic reasoning task:

- Explaining solutions along with problems leverages LLMs' abilities in code comprehension, enable reasoning process distilling for problems beyond models' capacities.

- **Learning from** hierarchical, detailed, and semantic-rich **explanations** of <problem, solution-program> pairs is more effective than learning from them directly and can substantially improve LLMs' performance in solving these problems.

- Sampling more strategies allows distinct reasoning paths, which can improves solve@k when k>1

Conclusion from Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs

To tackle the challenging algorithmic reasoning task:

- Explaining solutions along with problems leverages LLMs' abilities in code comprehension, enable reasoning process distilling for problems beyond models' capacities.

- **Learning from** hierarchical, detailed, and semantic-rich **explanations** of <problem, solution-program> pairs is more effective than learning from them directly and can substantially improve LLMs' performance in solving these problems.

- Sampling more strategies allows distinct reasoning paths, which can improves solve@k when k>1

Outline

- Introduction & Related Works
- Completed Works
  - Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs (2 workshop papers @ NLRSE 2023, NLRSE 2024)
  - CodeTree: Agent-guided Tree Search for Code Generation with Large Language Models (NAACL 2025)
  - AlgoSimBench: Identifying Algorithmically Similar Problems for Competitive Programming (Under Review)
  - My Other Works@UT: Learning to Reason Deductively (ACL 2022); ContraDoc (NAACL 2024)
- Proposed Future Works

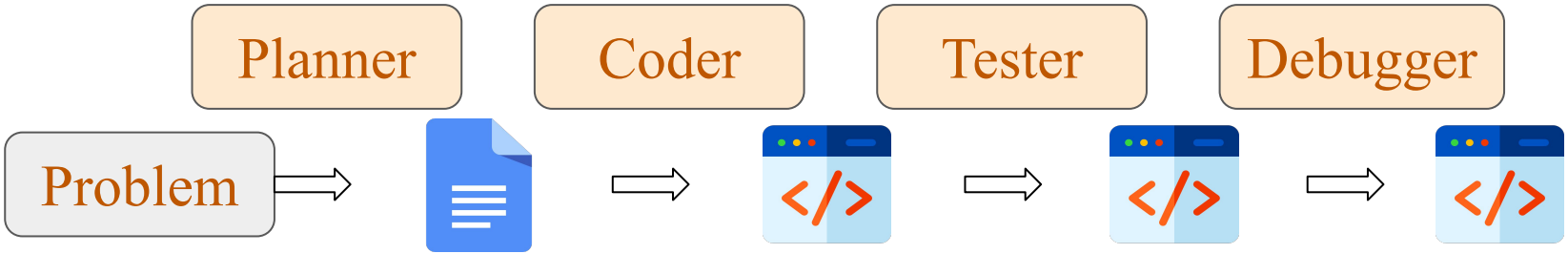
Findings & Questions

- LLMs are good at explaining solutions.
- Separating the strategy-exploration & code implementation helps structuring the reasoning process
- Exploring diverse problem-solving strategies performs better than exploring diverse implementations of similar strategy.

- Can LLMs self-explain its generated solutions?
- May we apply this to a zero-shot setting?

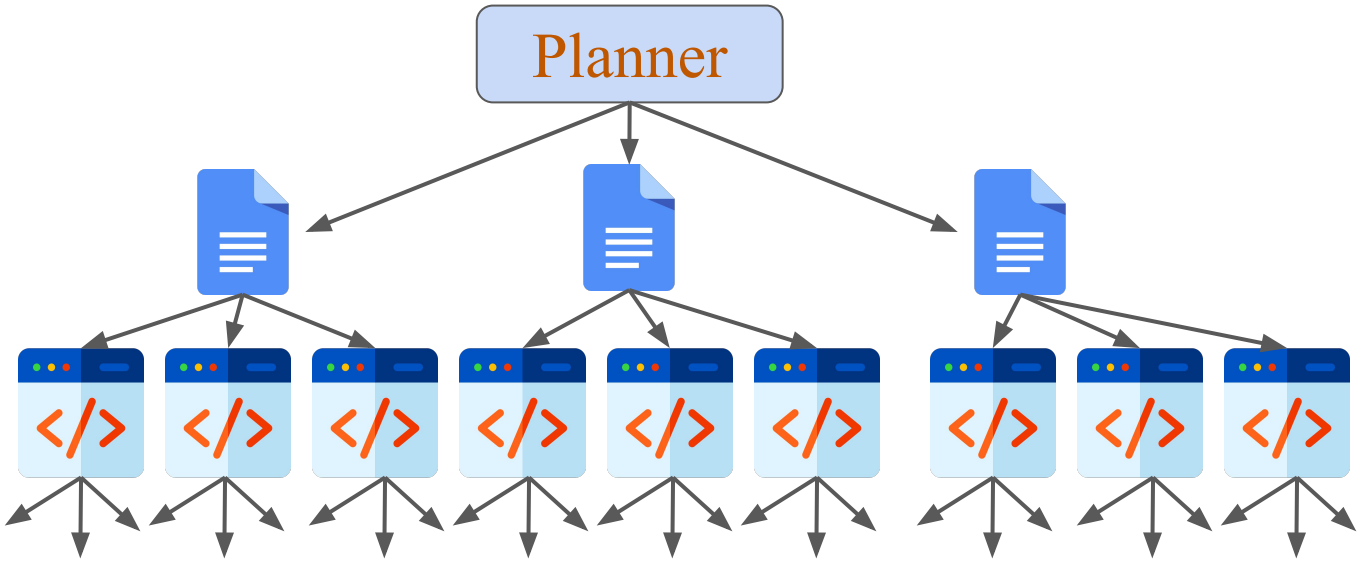
- How do LLMs efficiently explore different strategies in the search space?

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

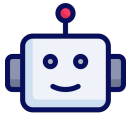
Carefully set the width & depth according to task

Limitations of Previous Works

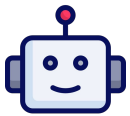
- Agent pipeline is fixed process
- Tree exploration relies on width&depth parameters
- Exit condition is simple (e.g., pass visible test cases)

Motivation

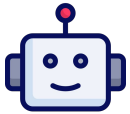
- Agent pipeline is flexible
- Tree Expanding is dynamic
- Exit Condition (Verified)



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?

Method Overview

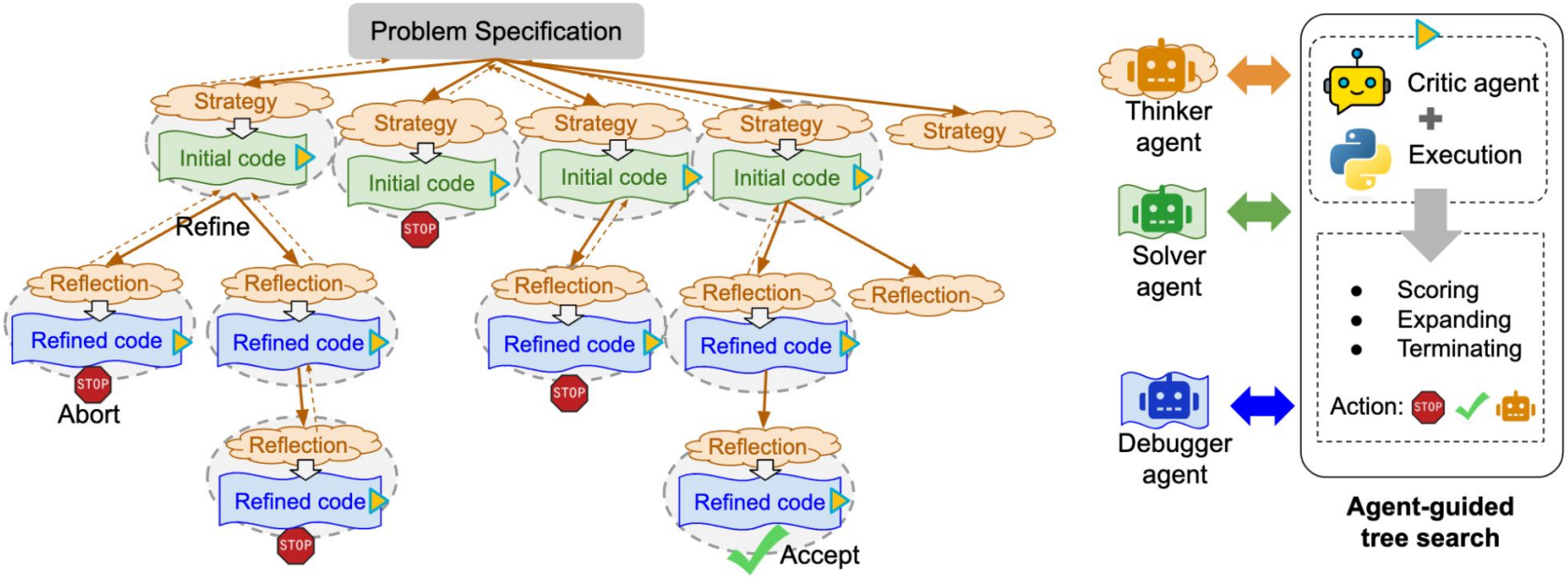


Figure 1: CodeTree creates a unified search space for exploration throughout the multi-stage code generation process: strategy generation by a "Thinker" agent, initial code generation by a "Solver" agent, and code improvement by a "Debugger" agent. To effectively perform exploration within the tree structure, we incorporate both environmental execution-based feedback as well as AI-generated feedback (generated by a "Critic" LLM agent).

Agents (Zero-shot Instruction)

- **Local Functional Agents**:
  - Thinker: Generate Strategies for Solving or Error Reflection
  - Solver: Implement Code from Strategy
  - Debugger: Refine Code based on Reflection

- **Global Critic Agent**:
  - Tree Expansion, number of children
  - Node-Wise Decision Making on Next Step
  - Evaluate Solutions w/ Execution Feedback from Environment

Prompts

Thinker:
Your goal is to think of multiple strategies in English on how to [approach and solve this problem)]/[improve this solution]. You should decide how many and what strategies are feasible and list and number them line by line. …
[Problem]: <problem description>
[Solution]: <previous solution>

Solver:
Your goal is to implement the solution for a programming problem based on the instruction from user.
[Problem]: <problem description>
[Instruction]: <strategy>

Debugger:
Your goal is to improve the following solution for a programming problem based on its execution feedback on test cases, including evaluation/reflection for the solution and an instruction from user. …
[Problem]: <problem description>
[Solution]: <previous solution>
[Feedback]: <execution feedback>, <Critic Agent feedback>
[Instruction]: <reflection>

Critic Agent Scoring & Evaluation:
Your task is to evaluate a strategy and corresponding implementation for solving a programming problem. The solution failed on test cases.
You should **score** from 1 to 5 on how good the execution outputs are matching the expected ones. …
—----------
You should **score** from 1 to 5 on how well do the solution implement the strategy and solve the task?
Evaluate if one should keep refining this solution or try other strategies.
[problem] [solution] [feedback]

Critic Agent Solution Verification:
You are given a programming task along with a user's solution that passed all visible tests. Your job is to verify whether this solution will pass the hidden test cases. Answer True if it's an acceptable solution, Answer False if it's not. Your answer should be a single word **True/False**. …
[problem][solution][feedback]

Experiments

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

- **Evaluation Metric:**

  Pass@1 (only 1 program will be run against evaluator);
  code budget = 20 (allow to gen at most 20 programs per prob)

Experiments

Baselines & Methods

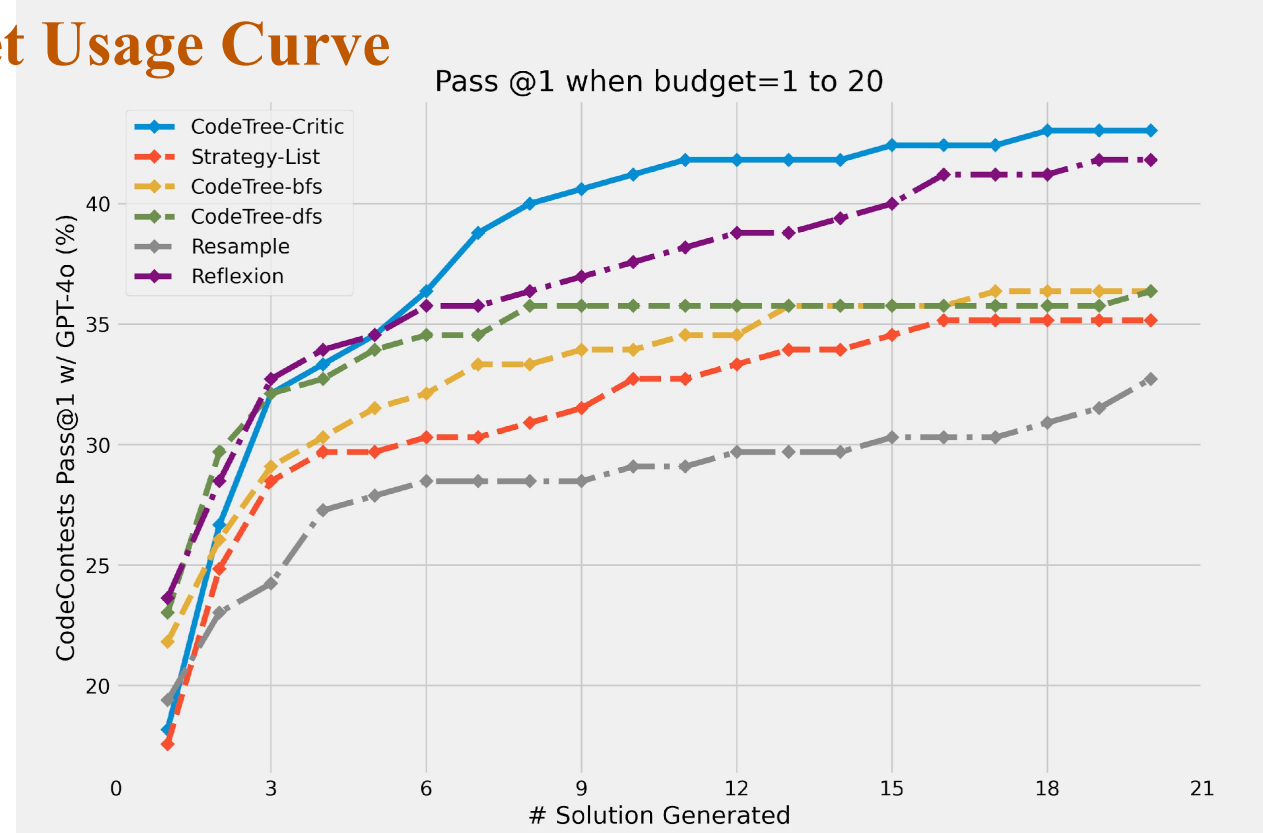
- **CodeTree-BFS/DFS**: use BFS/DFS to replace agent judge for what nodes to explore and tree expansion
- **Resample**: Keep resampling till reaching budget.
- **Reflexion**: Keep code-execute-reflect-code till reaching budget (Shinn et al., 2023)
- **Strategy List**: List budget of strategy, implement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

Experimental Results

Model	Method	HumanEval	HumanE+	MBPP	MBPP+	Codecontests	Avg.
GPT-4o-mini	Direct	86.6%	78.7%	87.8%	73.3%	10.3%	67.3%
	CoT	84.8%	78.0%	89.2%	74.3%	12.7%	67.8%
	Reflexion	92.1%	83.5%	96.6%	**78.6%**	21.8%	74.5%
	MapCoder	91.5%	78.0%	90.0%	-	-	-
	Resample	89.0%	80.5%	94.3%	76.8%	18.2%	71.8%
	CodeTree-BFS	93.3%	82.1%	91.5%	72.3%	20.6%	72.0%
	CodeTree-DFS	92.7%	81.1%	87.6%	71.4%	20.6%	70.7%
	Strategy List	90.2%	80.5%	90.5%	69.6%	22.4%	70.6%
	CodeTree	**94.5%**	**84.8%**	**96.8%**	77.0%	**26.4%**	**75.9%**
GPT-4o	Direct	88.4%	81.7%	92.3%	75.9%	20.6%	71.8%
	CoT	92.1%	84.1%	93.7%	77.2%	24.8%	74.4%
	Reflexion	94.5%	84.8%	97.9%	79.6%	41.8%	79.7%
	MapCoder	92.7%	81.7%	90.9%	-	-	-
	Resample	93.9%	84.8%	96.2%	77.0%	32.7%	76.9%
	CodeTree-BFS	94.5%	84.1%	93.9%	70.7%	35.8%	75.8%
	CodeTree-DFS	**95.1%**	83.5%	91.5%	76.2%	36.4%	76.5%
	Strategy List	**95.1%**	82.3%	92.6%	73.3%	36.4%	75.9%
	CodeTree	94.5%	**86.0%**	**98.7%**	**80.7%**	**43.0%**	**80.6%**

Experimental Results

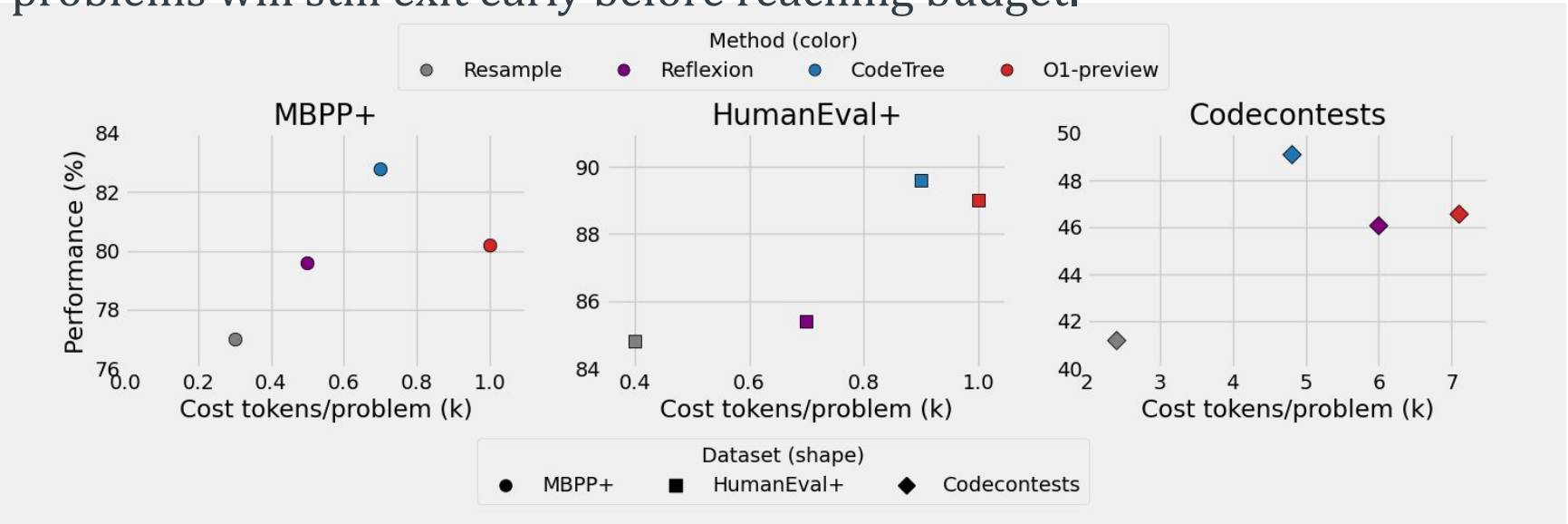
- **Budget Usage Curve**



Completed Work: CodeTree

Efficiency

Adding budget doesn't increase the average inference time linearly, simple problems will still exit early before reaching budget.



(code budget=30 for Resample, Reflexion & CodeTree under GPT-4o; off-the-shelf O1-preview)

Conclusion from CodeTree

To solve algorithmic programming challenges in a 0-shot setting:

- **Separating Natural Language Reasoning** with **Code Implementation** into 2 stages allows agents to explore distinct thoughts towards problem-solving.
- Exploring the correct solution search space can be exhausive, and agent-guided tree seach can make it efficient.

Significant Progress has been made to Solving CP. Can LLMs trained to solve problems identify similar problems?

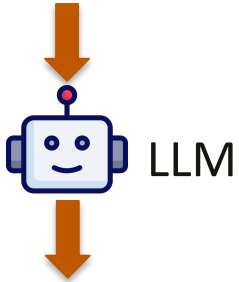
Outline

- Introduction & Related Works
  - Emergence of Large Language Models
  - Gap between Natural Language Reasoning and Symbolic Language Reasoning
  - Competitive-Level Programming in the Era of LLMs
- Completed Works
  - Learning Algorithmic Reasoning with LLMs from Explaining Solution Programs (2 workshop papers @ NLRSE 2023, NLRSE 2024)
  - CodeTree: Agent-guided Tree Search for Code Generation with Large Language Models (NAACL 2025)
  - AlgoSimBench: Identifying Algorithmically Similar Problems for Competitive Programming (Under Review)
  - My Other Works@UT: Learning to Reason Deductively (ACL 2022); ContraDoc (NAACL 2024)
- Proposed Future Works
  - Instruction-aware Code Embedding
  - Learning Embeddings from Downstream Code Generation Feedback

Completed Work: AlgoSimBench

The Myth of Generalizing Algorithmic Reasoning

Problem



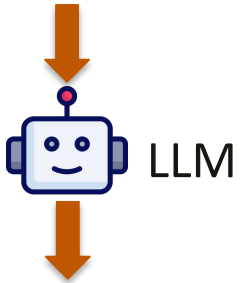
(Chain-of-thought) Solution

A Top-down Point of View

- Analyze the problem and identify the problem type
- Identify the solving strategies and algorithms to use
- Work out the details on the algorithm
- Implement the code => debug

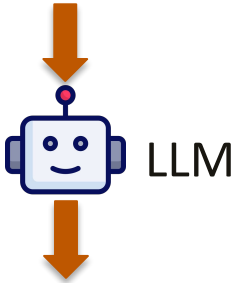
Can LLMs trained to solve problems identify similar problems?

Problem



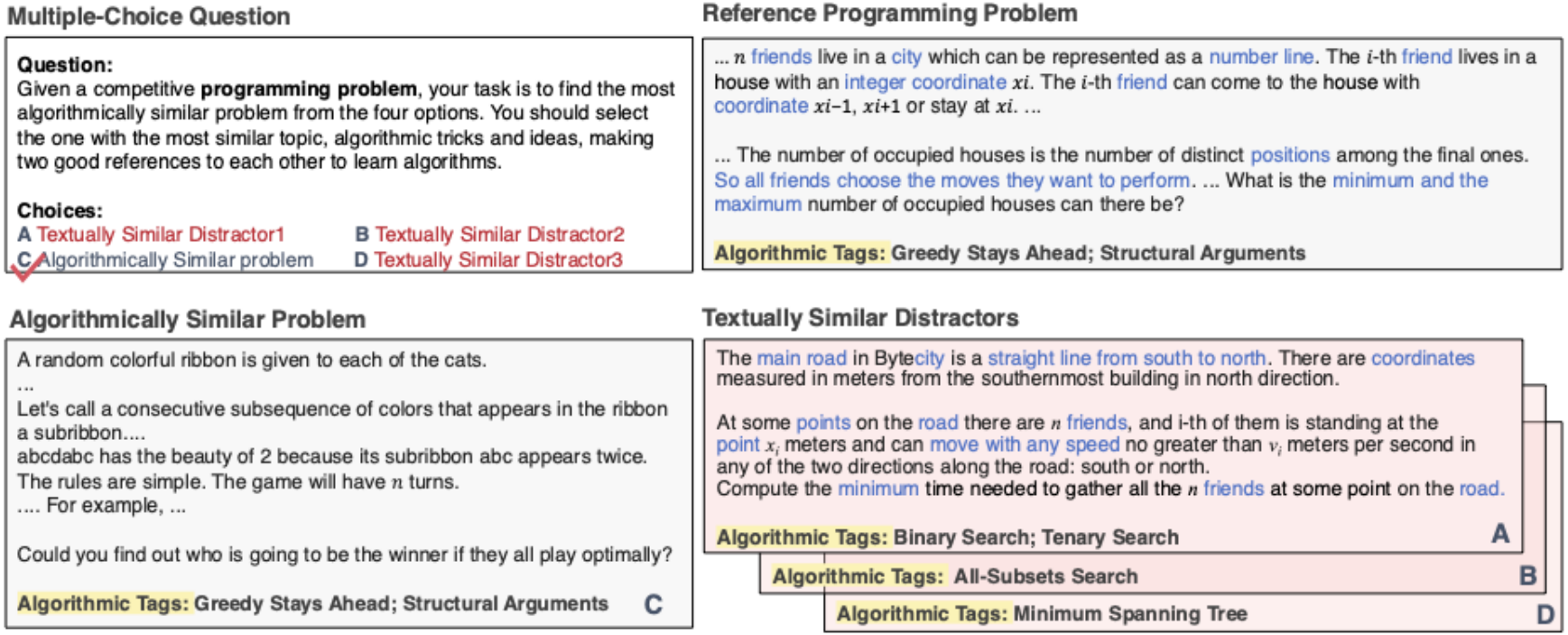
(Chain-of-thought)
Solution

Problem, Problem, ..., Problem
Ref_Prob



the __ problem requires the same algorithm as the ref_prob

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

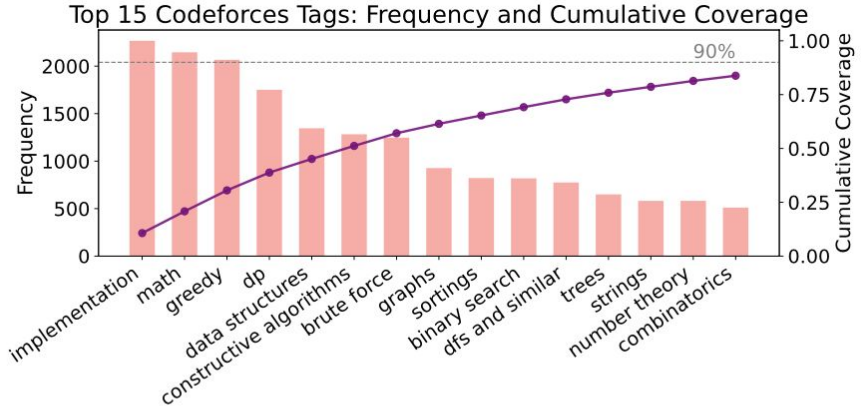


Data Curation

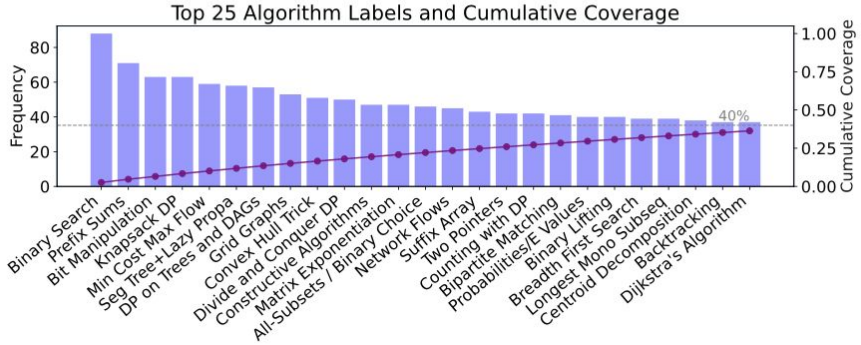
- Problem Algorithm Labels:
  Four Competitive-Programming Online Communities
- Problem Sources: Codeforces, AtCoder, CodeChef
- Semantic-adversary: Distractors as semantically similar as possible, Correct option as semantically dissimilar as possible
- Human-Verify: filter out False Distractors

402 MCQs with 903 distinct problems, 231 distinct fine-grained algorithm tags.

Fine-grained Algorithm Tags



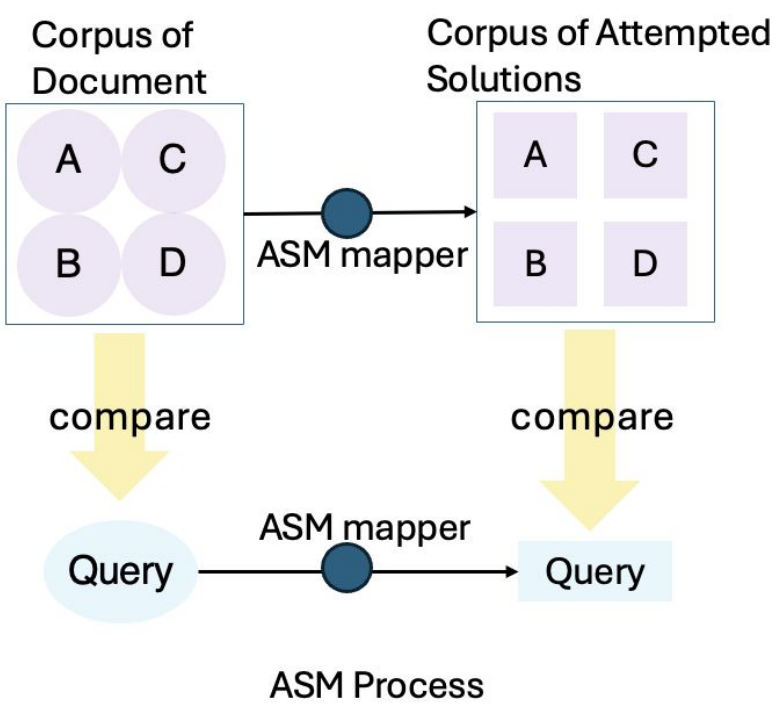
(a) Top 15 Codeforces broad algorithm tags.



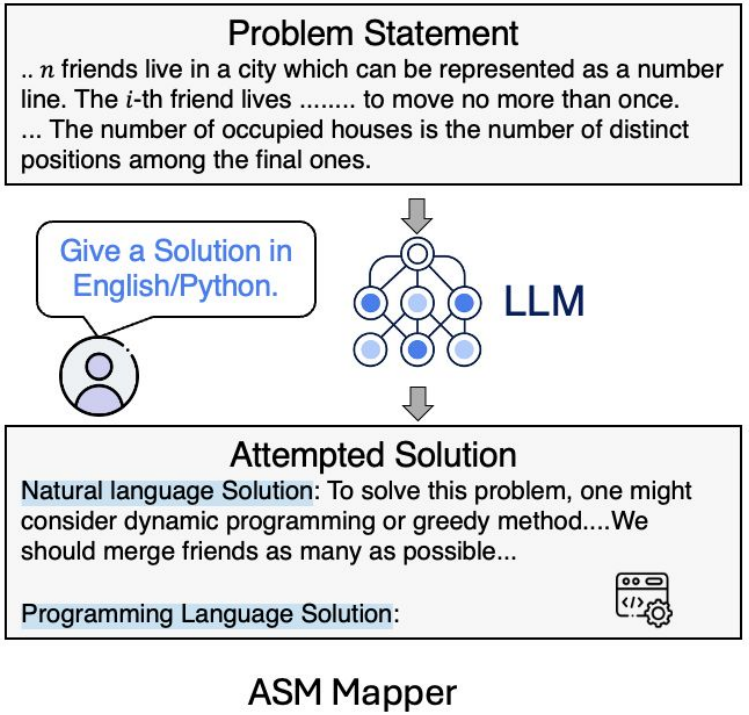
(b) Top 25 fine-grained labels in our dataset.

Instead of general tags like dynamic programming, we use fine-grained algorithms tags that can represent the solution better.

Attempted Solution Matching



ASM Process



ASM Mapper

Experimental Settings

- Dataset: AlgoSimBench 402 MCQs
- Evaluation metric: cross-dataset accuracy
- Experimental Settings:
  - LLM End2End Selection: {problem}{options} =LLM=> {CoT}{option}
  - Retrieval Setting: Sim(ref_p, correction_option) > Sim(ref_p, option_i)
- Methods:
  - Problem Statement
  - Summary
  - Oracle Solution
  - ASM-NL(Natural Language); ASM-PL (Programming Language)

LLM End-to-End Selection

Model	Statement	Summary	ASM-NL	ASM-PL	Solution*
GPT-4o-mini	35.5	35.8	**43.8 (↑ 8.3)**	42.5 (↑ 7.0)	54.4
GPT-4o	41.5	38.1	**53.2 (↑ 11.7)**	53.0 (↑ 11.5)	63.4
o3-mini-medium	65.9	63.4	74.4 (↑ 8.5)	**75.1 (↑ 9.2)**	72.6
Deepseek-R1	63.7	57.7	69.2 (↑ 5.5)	**70.4 (↑ 6.7)**	70.6
Deepseek-V3	55.2	53.2	**64.9 (↑ 9.7)**	62.7 (↑ 7.5)	66.7
Claude-3.5-Sonnet	44.2	45.0	**54.7 (↑ 10.5)**	53.0 (↑ 8.8)	70.5
Gemini 2.0 Flash	51.2	48.5	**57.9 (↑ 6.7)**	55.5 (↑ 4.3)	69.7
Avg	50.4	48.8	59.5 (↑ 9.1)	58.5 (↑ 8.1)	66.5

Evaluate models' performances (%) of MCQ are correct.
Solution*: An oracle setting where each human-written correct solution is assumed to be available any time.

Retrieval-based Selection

	Summary			ASM-NL			ASM-PL		
	BM25	BART	GCB	BM25	BART	GCB	BM25	BART	GCB
GPT-4o-mini	25.6	23.4	20.6	**35.3**	29.1	22.4	34.8	26.1	32.8
GPT-4o	25.8	24.6	26.1	**42.5**	30.1	32.1	35.6	28.8	29.1
o3-mini-medium	39.3	32.6	31.3	**49.0**	35.6	38.3	48.8	28.4	39.3
Deepseek-V3	29.9	26.1	22.4	**46.0**	33.1	32.1	45.5	32.3	35.1
Deepseek-R1	30.1	24.1	25.9	**52.2**	32.8	31.1	45.0	32.8	35.9
Gemini-2.0-Flash	27.1	24.4	18.6	**41.0**	34.3	26.6	38.0	36.8	35.6
Claude-Sonnet-3.5	29.4	25.1	25.4	**39.6**	28.1	28.6	35.0	29.1	29.9
Avg	29.6	25.8	24.3	**43.7**	31.9	30.2	40.4	30.6	34.0

Accuracy(%) comparison across summary, ASM generated by different LLMs.
GCB: graph-code-bert

Conclusion from AlgoSimBench:

- The algorithmic reasoning abilities learnt from problem solving does not automatically generalize to similar domain: identifying algorithmically similar problems
- To explicitly enforce LLMs to solve the problem first will largely improve LLMs' performances in identifying algorithmically similar problems
- BM25 which focuses more on keywords performs better than dense retrieval methods

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

Outline

- Introduction & Related Works
- Completed Works
- Proposed Future Works
  - Instruction-aware Code Embedding
  - Learning Embeddings from Downstream Code Generation Feedback

Proposed Future Works
Instruction-aware Code Embedding

Identifying Algorithmically Similar problems can be an important step towards generating diverse strategies for one problem as it can flag references in a corpus of problems.

To tackle the challenge presented in AlgoSimBench, we aim at improving the current Code Embedding models to highlight features related to algorithms and problem-solving.

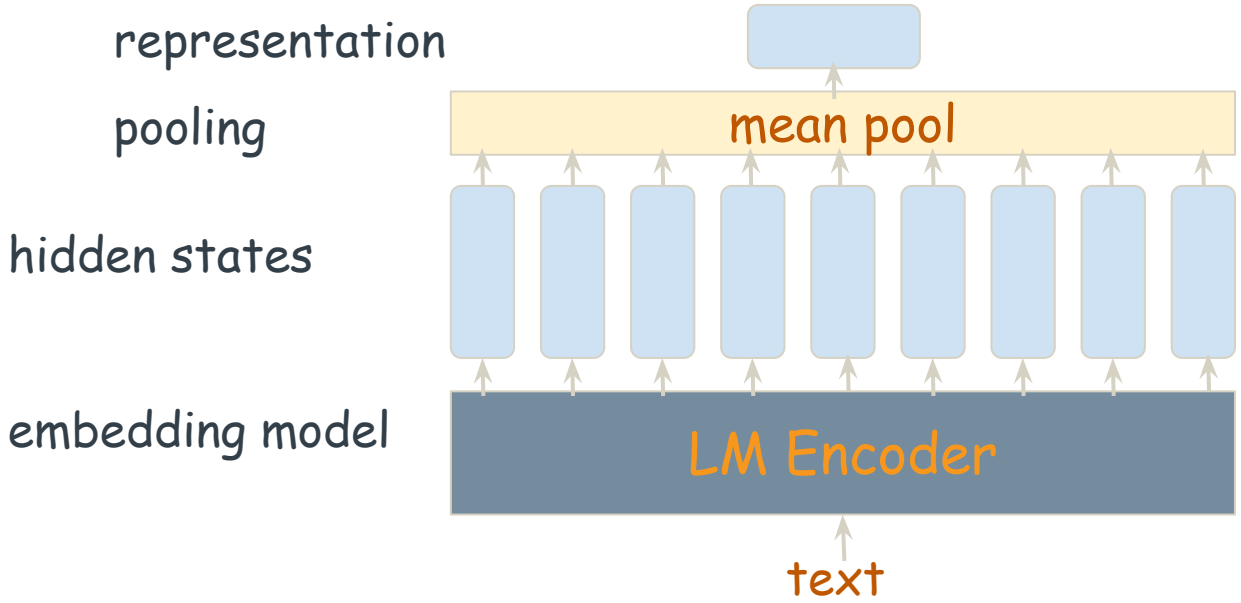
Instruction-aware Code Embedding Preliminaries

Text Piece Embedding: Given each piece of query or document, the embedding model represents it as a single vector.

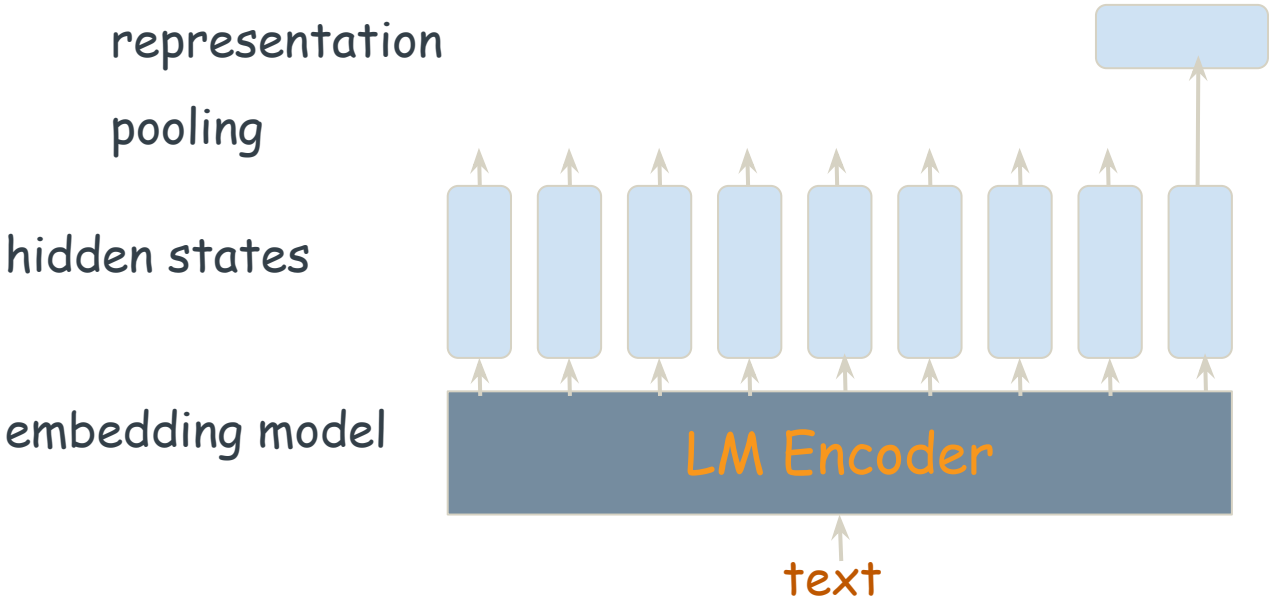
Top-k Candidate Finding: With cosine-similarity or doc-product similarity, find top-k closest vectors to the query vector.

Reranker: Interaction between query and each candidate document.

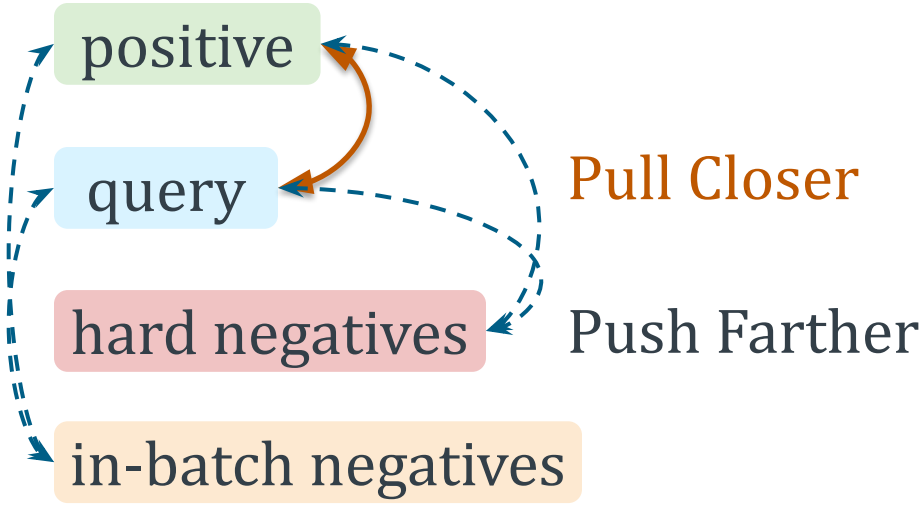
Mean Pooling



Last-token Pooling



Contrastive Training for Embedding Models



- Mining Hard Negatives[5]
- Causal LMs[6]
- Instruct-append[6]

[5]Suresh, Tarun, et al. "CoRNStack: High-quality contrastive data for better code retrieval and reranking." *arXiv preprint arXiv:2412.01007* (2024).
[6]Zhang, Yanzhao, et al. "Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models." *arXiv preprint arXiv:2506.05176* (2025).

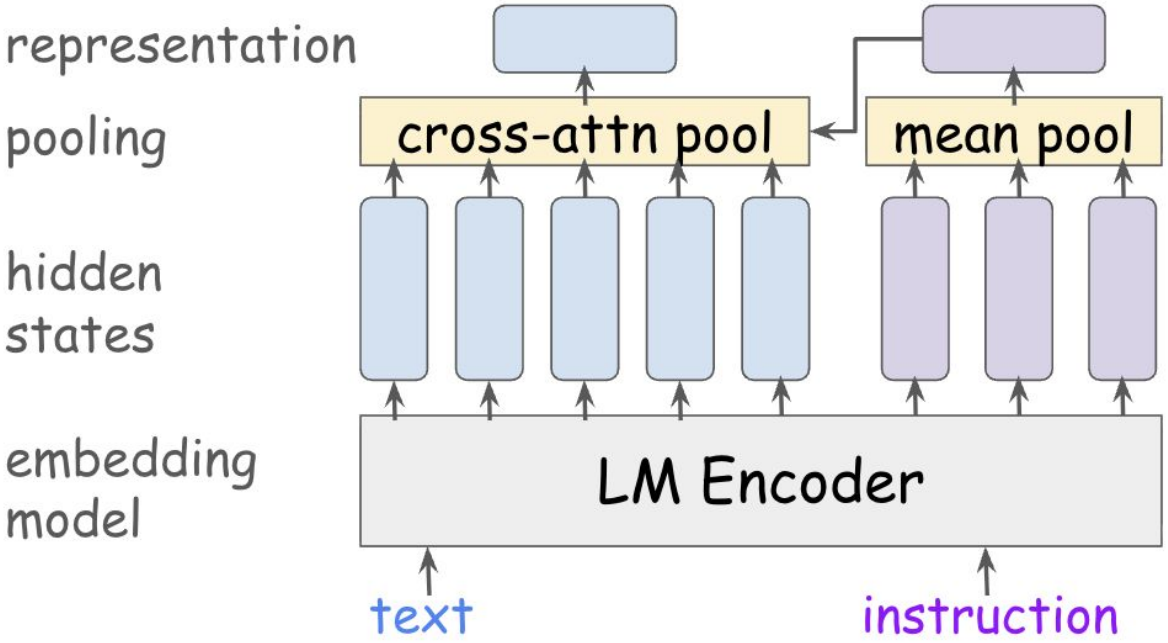
Limitations of Current Methods

Averaged Information in Text : Pooling aggregating token-level represen- tations into fixed-length embeddings, without specific focus of any part

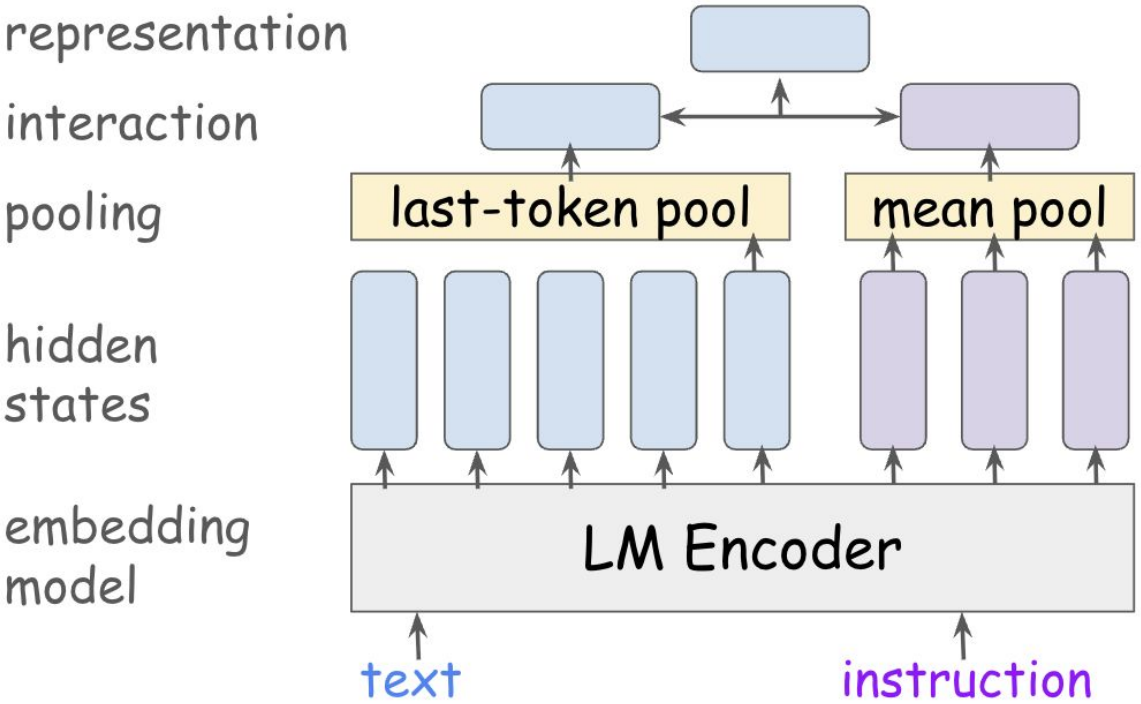
Task-Agnostic: Code2NL, NL2Code, Code2Code tasks are all feeded as <postive, anchor, negative> triplets, performance relies on “defining hard negatives”[5]

Intention-Unaware: One might intend to retrieve code pieces with “functionality” or “algorithm” or “programming language”.

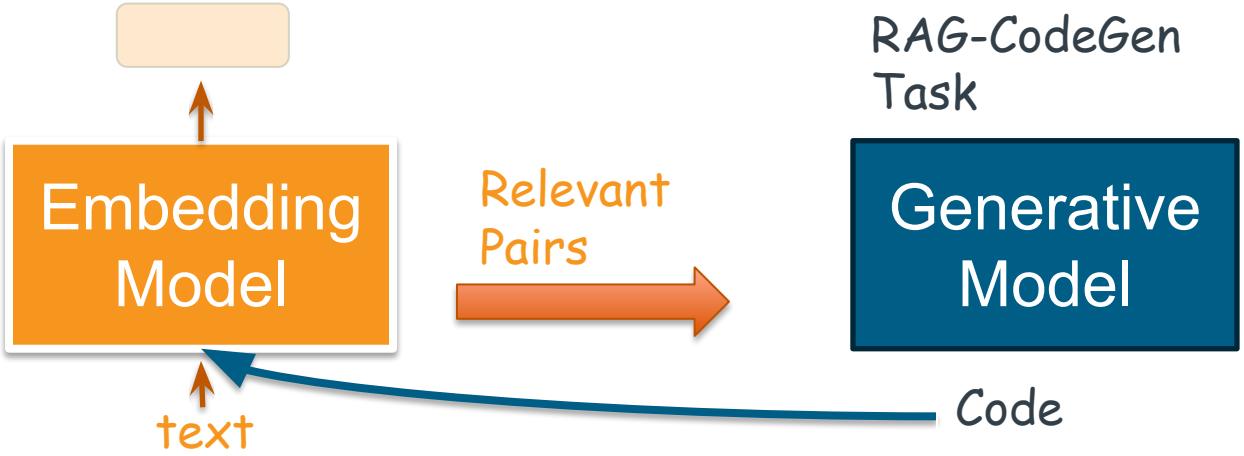
Instruction-attended Text Embedding



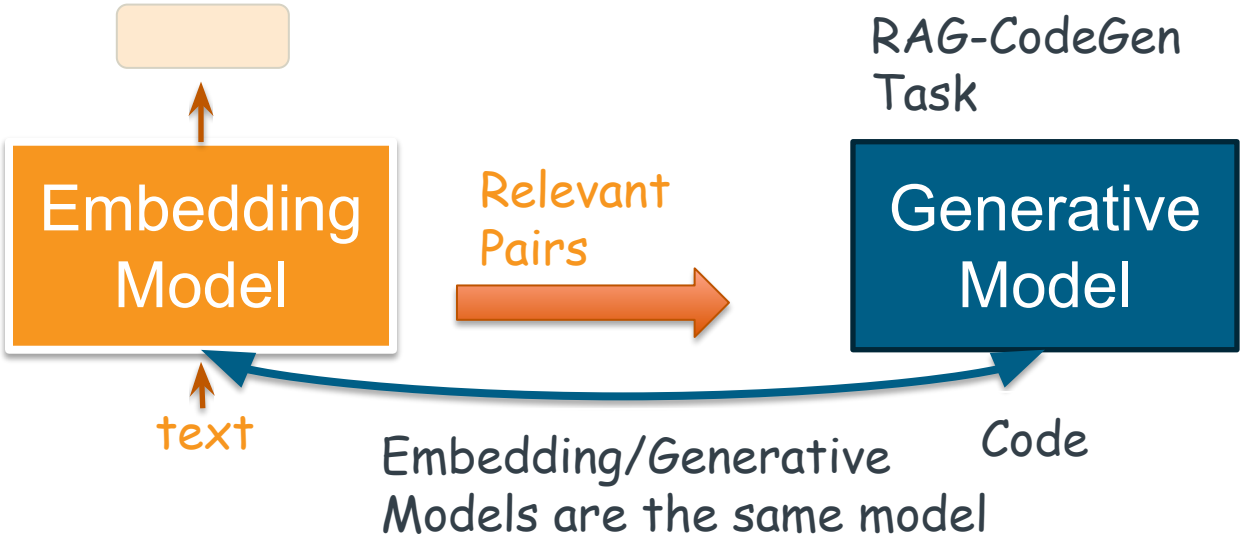
Interactive Text Embedding



Learning Embeddings from Downstream Code Generation Feedback



Learning Embeddings from Downstream Code Generation Feedback



Evaluation

Datasets:

- Embedding/Retrieve Benchmarks: MTEB[7], AlgoSimBench, CodeSearchNet[8]
- CodeRAG: CodeRAGBench, RPO
- CP CodeGen: LiveCodeBench, CodeContests

[7]Muennighoff, Niklas, et al. "Mteb: Massive text embedding benchmark." *arXiv preprint arXiv:2210.07316* (2022).
[8] Husain, Hamel, et al. "Codesearchnet challenge: Evaluating the state of semantic code search." arXiv preprint arXiv:1909.09436 (2019).
[9] Shi-Qi Yan and Zhen-Hua Ling. Rpo: Retrieval preference optimization for robust retrieval-augmented generation, 2025.

Question & Discussion