Code Generation: LLMs, LLM-agents and Programmatic RL

Karolina Drabent, 24.10.2025

Agenda

- What is Program Synthesis?
- LLMs-based Agents in Code Generation
- Programmatic Reinforcement Learning
- Programmatic RL with LLM Guided Search
- Conclusion

What is Program Synthesis?

Program Synthesis / Code generation - What is it?

The task of generating code based on (human) “intentions”.

	A	B
1	Yoshua Bengio	Y. Bengio
2	Hugo Larochelle	H. Larochelle
3	Tara Sainath	T. Sainath
4	Yann LeCun	Y. LeCun
5	Oriol Vinyals	O. Vinyals
6	Aaron Courville	A. Courville

Figure 1. Flash Fill automatically completes a string transformation task from just a single example.

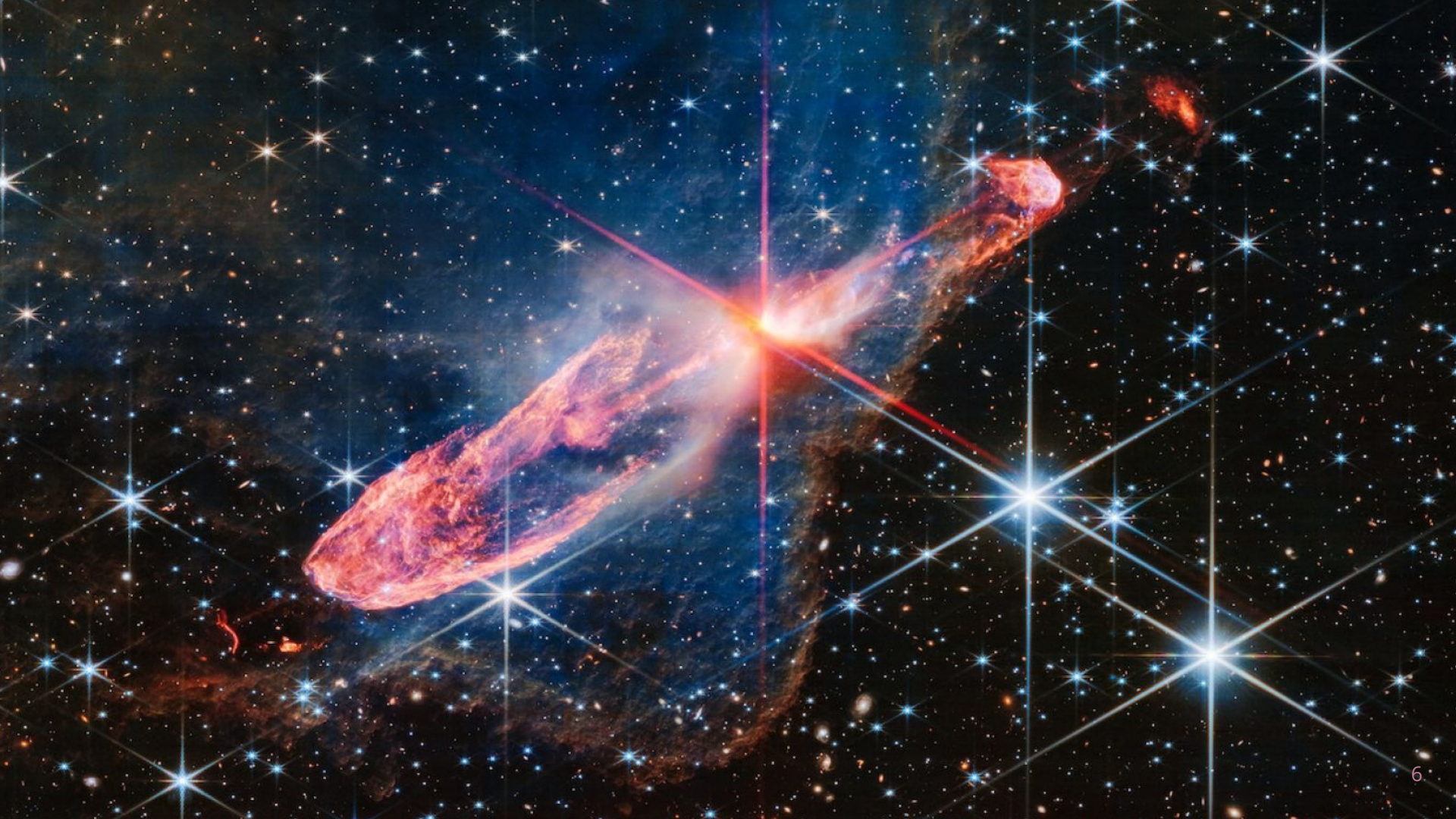
source: https://www.microsoft.com/en-us/research/blog/neural-guided-deductive-search-best-worlds-approach-program-synthesis/

Program Synthesis / Code generation - What is it?

The task of generating code based on (human) “intentions”.

The problem is that it’s basically a search in a HUUUUUGE space

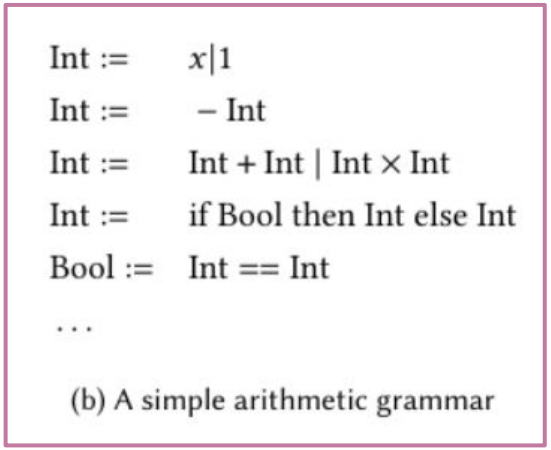
source: https://pursuit.unimelb.edu.au/articles/once-just-a-speck-of-light-now-revealed-as-the-biggest-known-galaxy-in-the-early-universe



Domain Specific Language - DSL

Makes the search space smaller

E.g., language for list processing or excel formulas



$$
\begin{array}{ll}
\text{Int} := & x \mid 1 \\
\text{Int} := & -\,\text{Int} \\
\text{Int} := & \text{Int} + \text{Int} \mid \text{Int} \times \text{Int} \\
\text{Int} := & \text{if Bool then Int else Int} \\
\text{Bool} := & \text{Int} == \text{Int} \\
\ldots &
\end{array}
$$

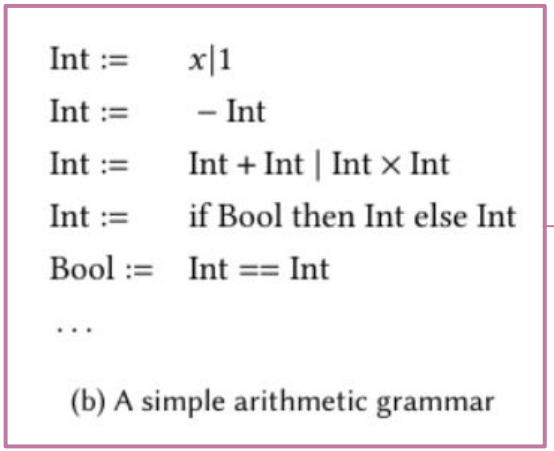
(b) A simple arithmetic grammar

source: Modelling Program Spaces in Program Synthesis with Constraints

Domain Specific Language - DSL

Makes the search space smaller

E.g., language for list processing or excel formulas



Int := $x|1$
Int := $-$ Int
Int := Int + Int | Int × Int
Int := if Bool then Int else Int
Bool := Int == Int
...

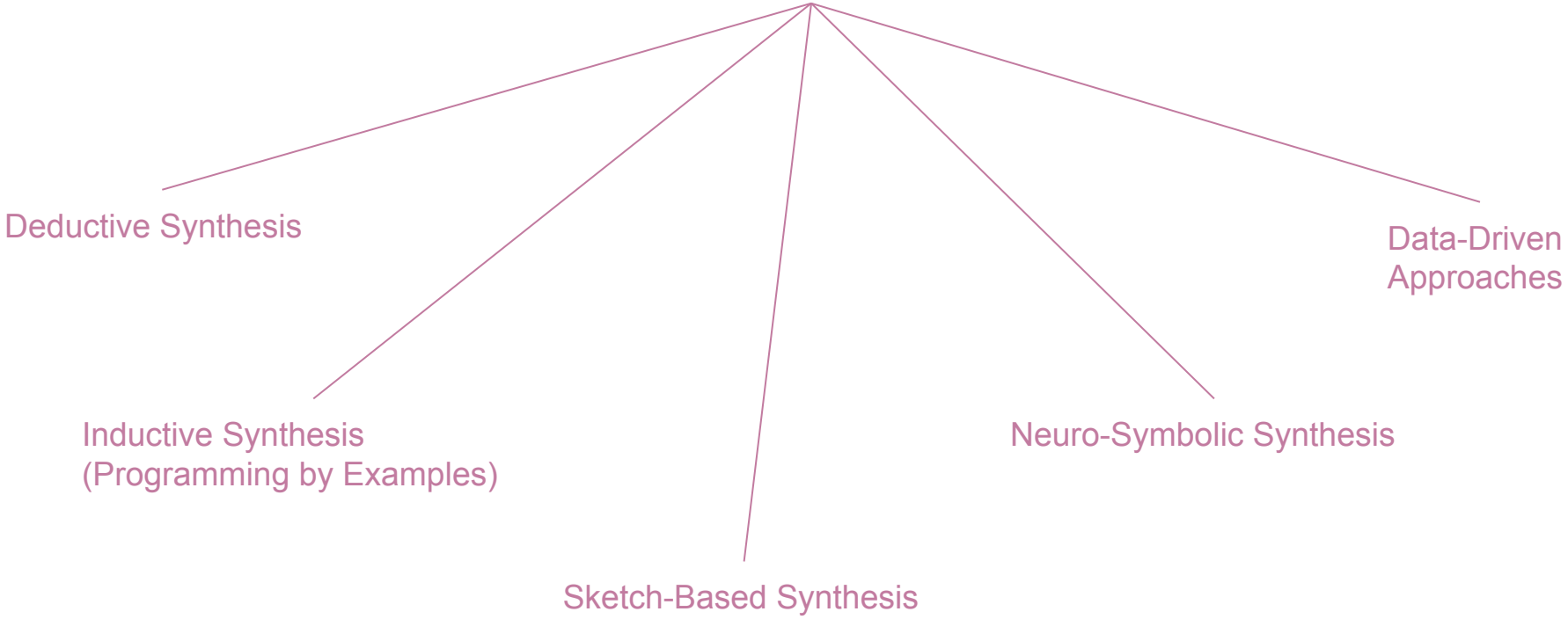
(b) A simple arithmetic grammar

example expressions:

- x
- if x==1 then 1+1 else x
- x*x*x*x*x*x

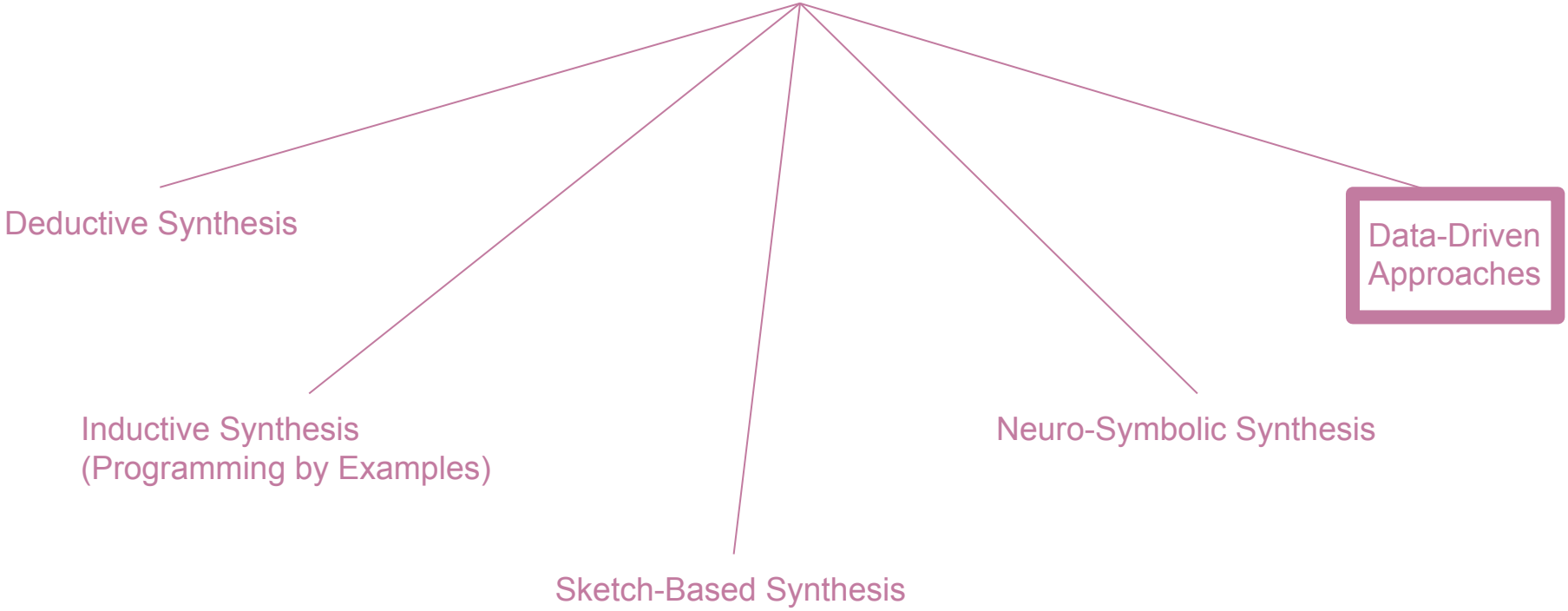
source: Modelling Program Spaces in Program Synthesis with Constraints

Program Synthesis



source: slightly modified “From Provable Correctness to Probabilistic Generation: A Comparative Review of Program Synthesis Paradigms”, 2025, Arnania, Kobaladze, Sanikize

Program Synthesis



source: slightly modified “From Provable Correctness to Probabilistic Generation: A Comparative Review of Program Synthesis Paradigms”, 2025, Arnania, Kobaladze, Sanikize

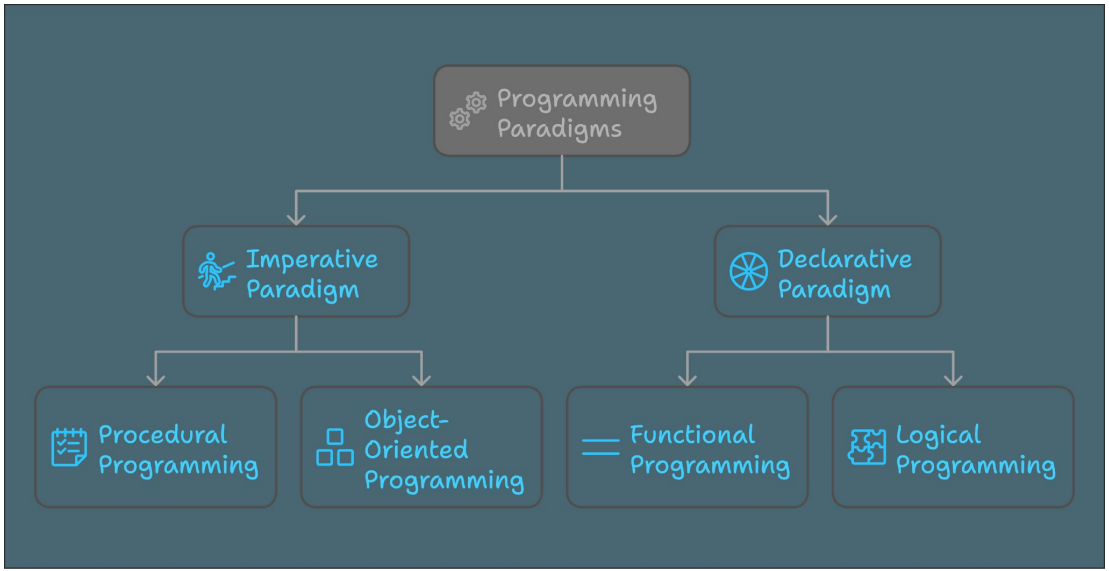
LLM-based Agents in Code Generation

Advantages of LLM

- Learn syntactic rules of multiple programming languages

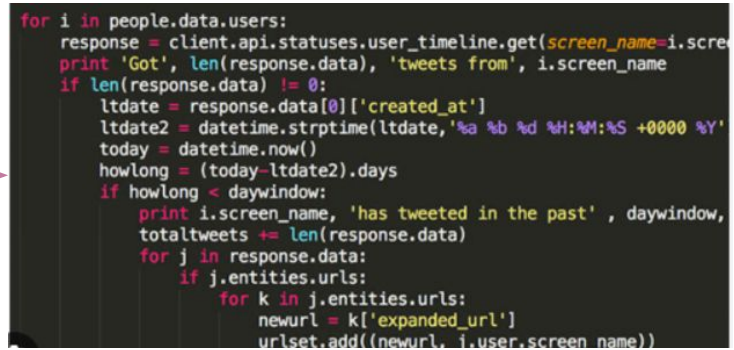
Advantages of LLM

- Learn syntactic rules of multiple programming languages
- Master common programming paradigms



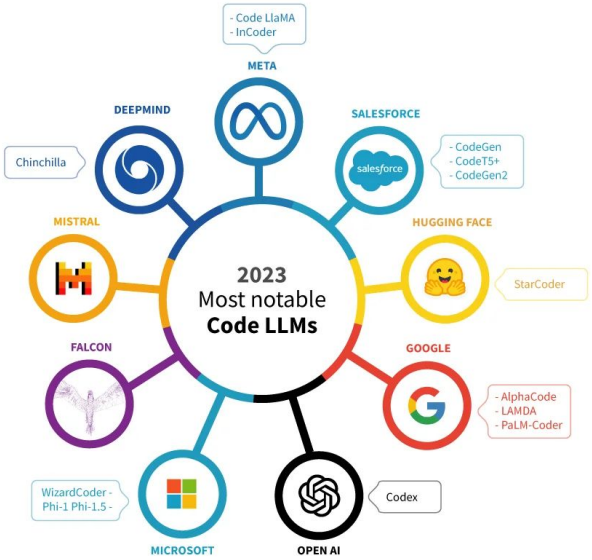
Advantages of LLM

- Learn syntactic rules of multiple programming languages
- Master common programming paradigms
- Understand mappings between natural language and code logic



```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, i.user.screen_name))
```

LLM Coders



Code LlaMA[1], InCoder, LLama 2 [2]) Code Llama is a specialized model for code-related tasks, offering strong code generation and instruction-following capabilities.
StarCoder [4] (the most advanced) (SantaCoder, OctoCoder, CodeParrot)

	Benchmark (Metric)	Claude-3.5-Sonnet-1022	GPT-4o 0513	DeepSeek V3	OpenAI o1-mini	OpenAI o1-1217	DeepSeek R1
	Architecture	-	-	MoE	-	-	MoE
	# Activated Params	-	-	37B	-	-	37B
	# Total Params	-	-	671B	-	-	671B
English	MMLU (Pass@1)	88.3	87.2	88.5	85.2	**91.8**	90.8
	MMLU-Redux (EM)	88.9	88.0	89.1	86.7	-	**92.9**
	MMLU-Pro (EM)	78.0	72.6	75.9	80.3	-	**84.0**
	DROP (3-shot F1)	88.3	83.7	91.6	83.9	90.2	**92.2**
	IF-Eval (Prompt Strict)	**86.5**	84.3	86.1	84.8	-	83.3
	GPQA Diamond (Pass@1)	65.0	49.9	59.1	60.0	**75.7**	71.5
	SimpleQA (Correct)	28.4	38.2	24.9	7.0	**47.0**	30.1
	FRAMES (Acc.)	72.5	80.5	73.3	76.9	-	**82.5**
	AlpacaEval2.0 (LC-winrate)	52.0	51.1	70.0	57.8	-	**87.6**
	ArenaHard (GPT-4-1106)	85.2	80.4	85.5	92.0	-	**92.3**
Code	LiveCodeBench (Pass@1-COT)	38.9	32.9	36.2	53.8	63.4	**65.9**
	Codeforces (Percentile)	20.3	23.6	58.7	93.4	**96.6**	96.3
	Codeforces (Rating)	717	759	1134	1820	**2061**	2029
	SWE Verified (Resolved)	**50.8**	38.8	42.0	41.6	48.9	49.2
	Aider-Polyglot (Acc.)	45.3	16.0	49.6	32.9	**61.7**	53.3
Math	AIME 2024 (Pass@1)	16.0	9.3	39.2	63.6	79.2	**79.8**
	MATH-500 (Pass@1)	78.3	74.6	90.2	90.0	96.4	**97.3**
	CNMO 2024 (Pass@1)	13.1	10.8	43.2	67.6	-	**78.8**
Chinese	CLUEWSC (EM)	85.4	87.9	90.9	89.9	-	**92.8**
	C-Eval (EM)	76.7	76.0	86.5	68.9	-	**91.8**
	C-SimpleQA (Correct)	55.4	58.7	**68.0**	40.3	-	63.7

Table 4 | Comparison between DeepSeek-R1 and other representative models.

source: DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning DeepSeek-AI, and https://www.krasamo.com/code-llm/

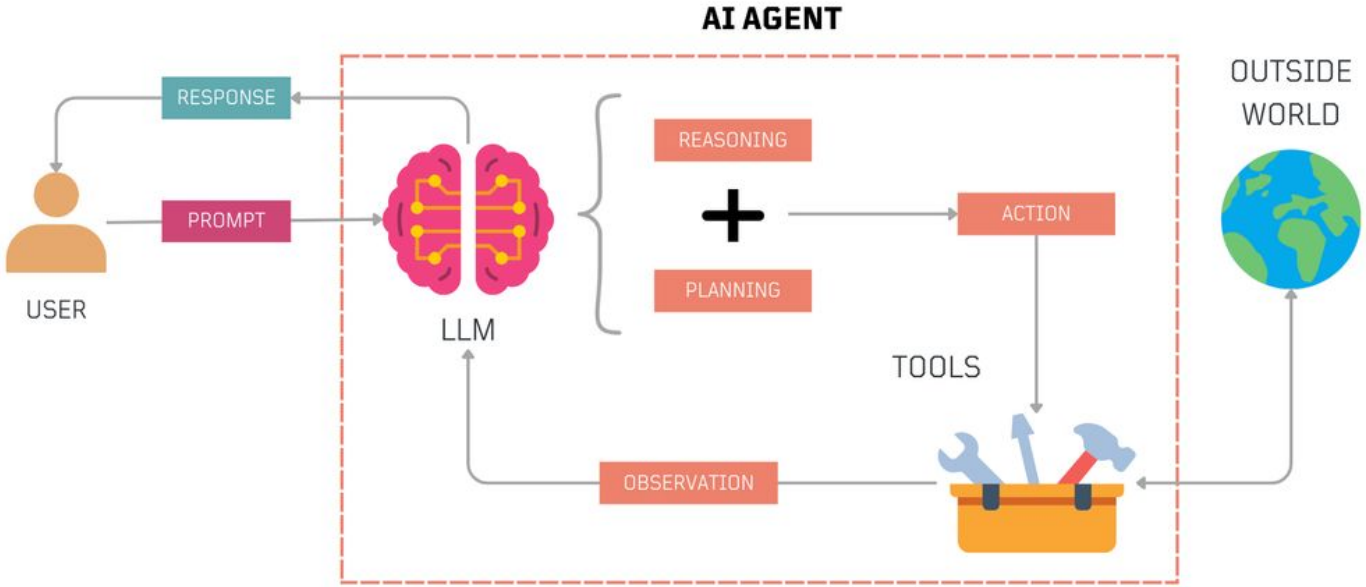
LLM Coders - Limitations

- Single-turn, passive response process
- Lack active planning, state tracking, and continuous interaction with external environments
- Struggle with complex, ambiguous, or multi-step collaborative development tasks

Solution?

LLM-based Agents!

LLM agents can enhance LLMs by for example tool usage, multiple rounds of thinking and generation etc.



source: https://blog.openthreatresearch.com/rise-of-the-planet-of-the-agents/

Shift in Code Generation due to LLM Agents

BEFORE

NOW

Shift in Code Generation due to LLM Agents

BEFORE

Passive

NOW

Shift in Code Generation due to LLM Agents

BEFORE

Passive

NOW

Actively manages and execute workflows

Shift in Code Generation due to LLM Agents

BEFORE

Passive

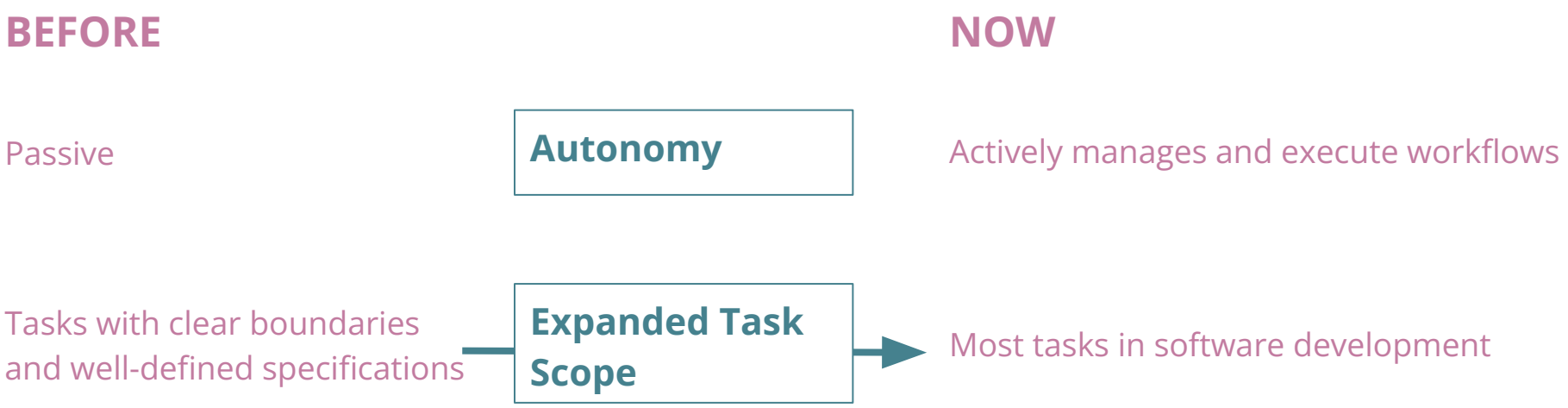
Tasks with clear boundaries
and well-defined specifications

Autonomy

NOW

Actively manages and execute workflows

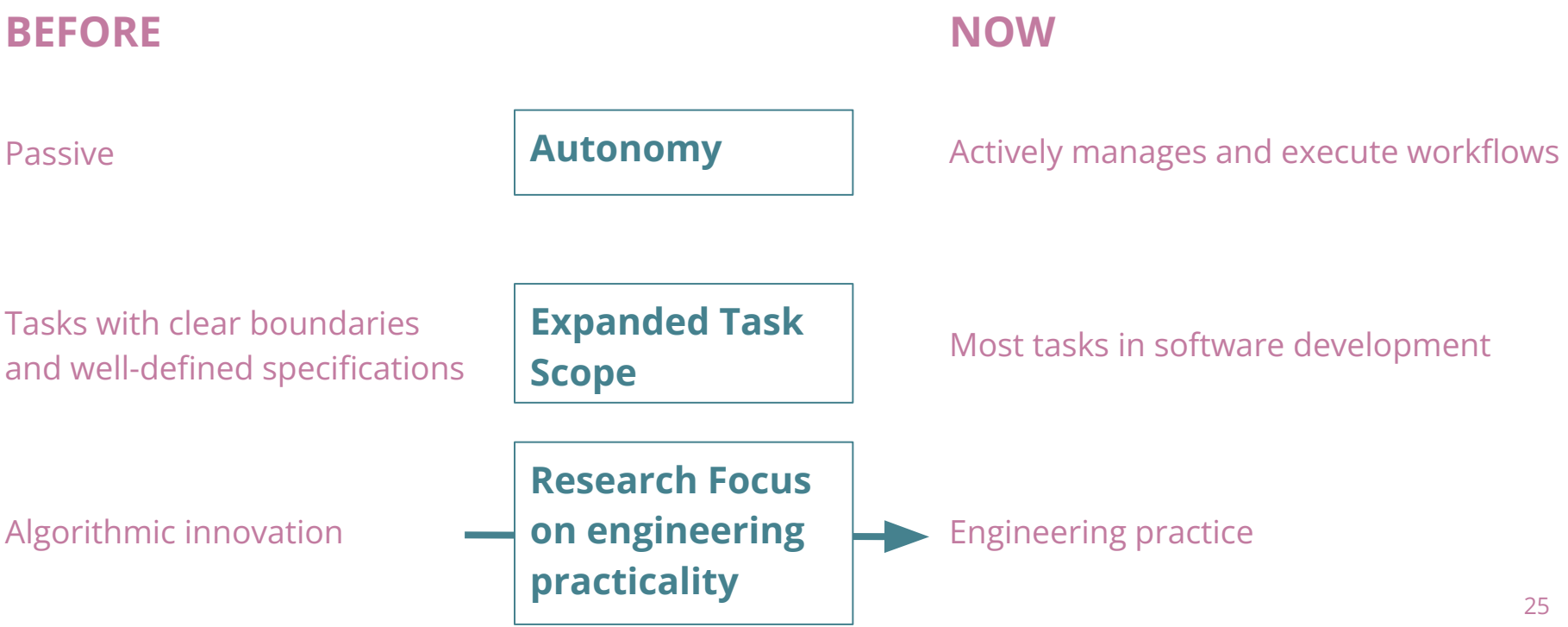
Shift in Code Generation due to LLM Agents



Shift in Code Generation due to LLM Agents

BEFORE		NOW
Passive	**Autonomy**	Actively manages and execute workflows
Tasks with clear boundaries and well-defined specifications	**Expanded Task Scope**	Most tasks in software development
Algorithmic innovation		

Shift in Code Generation due to LLM Agents



LLM Agent - Components

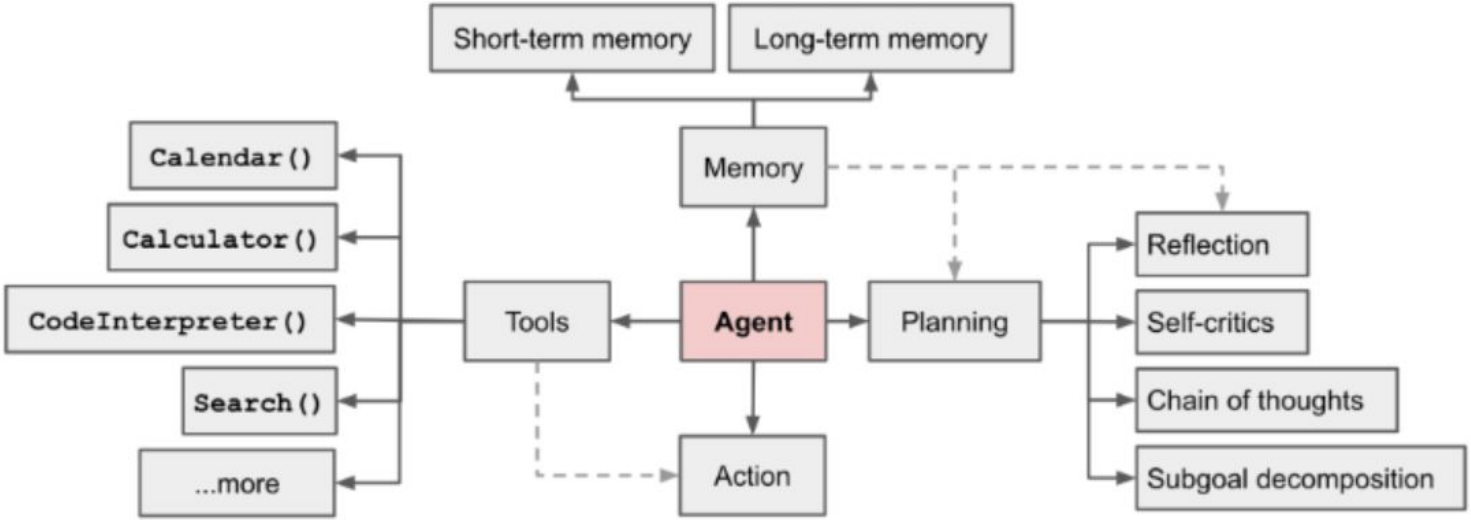
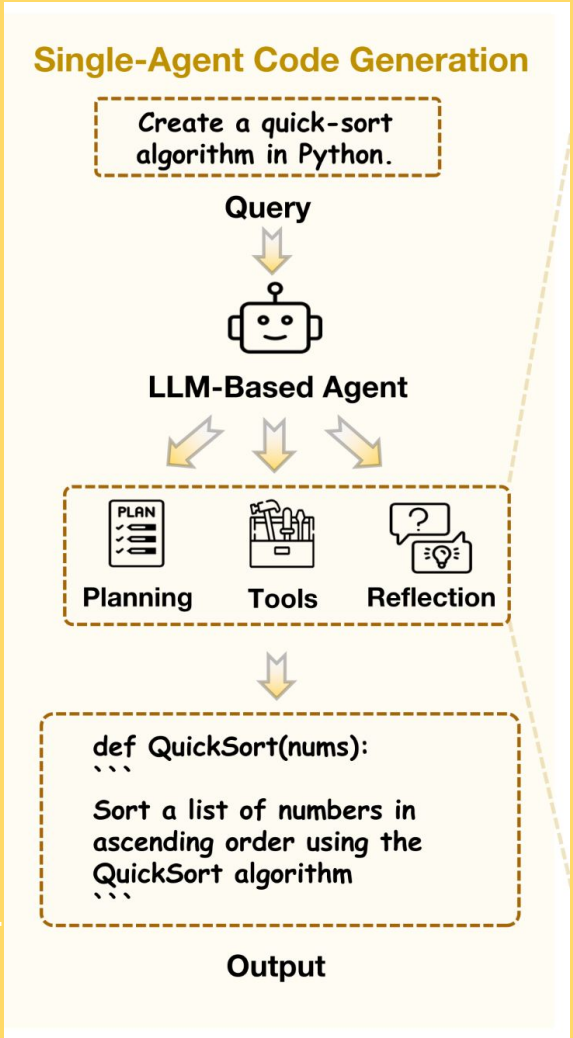


Figure 1: Overview of a LLM-powered autonomous agent system.

souce: https://lilianweng.github.io/posts/2023-06-23-agent/

Single-Agent

- Planning and Reasoning
- Tools and Retrieval
- Reflection and Self-Improvement

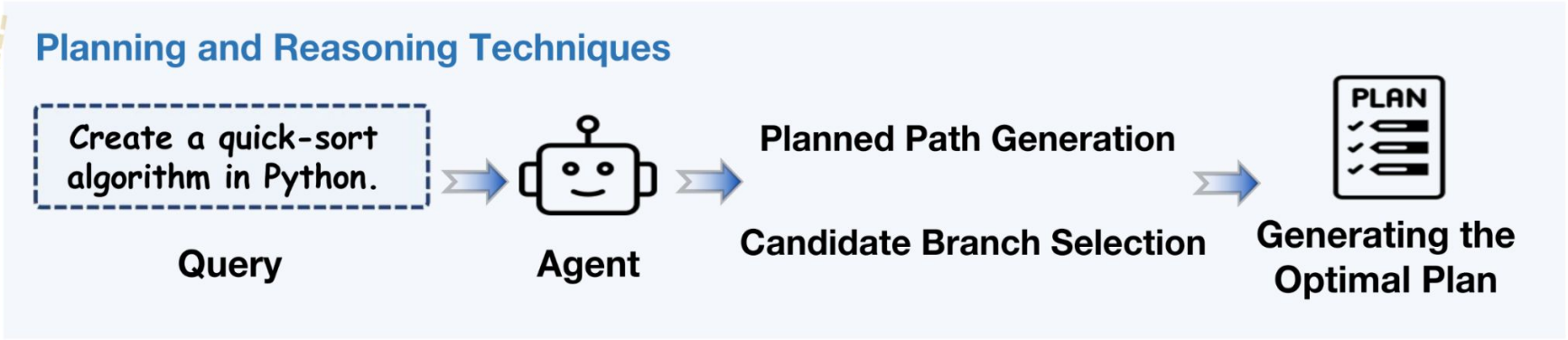


Planning and Reasoning Techniques

Self-Planning

Self-Revision of planning, e.g., CodeChain

Single-path to multipath exploration, e.g. GIF-MCTS, PlanSearch



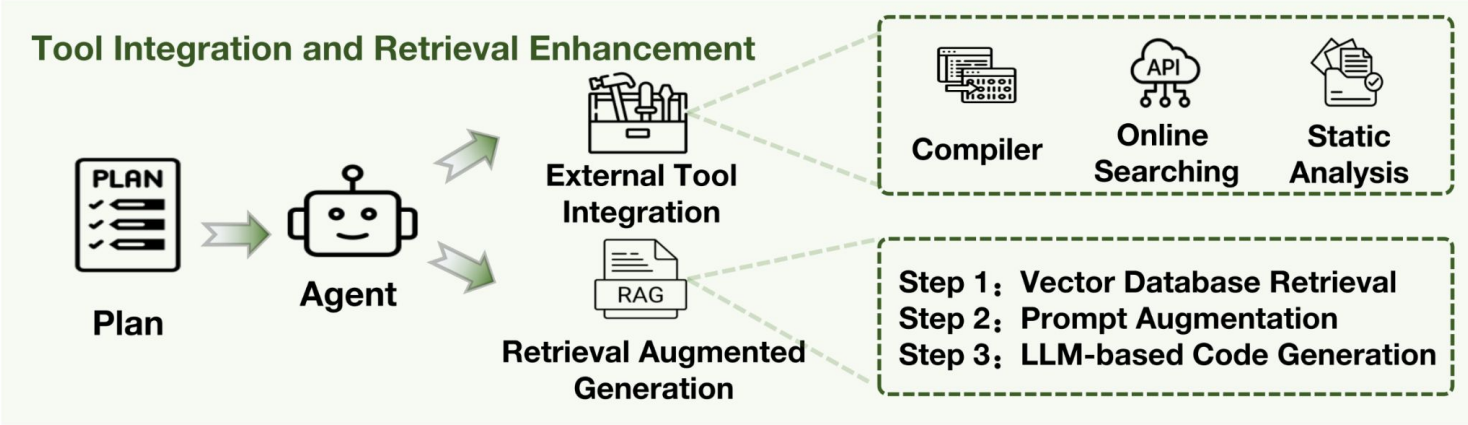
Tool Integration and Retrieval Enhancement

Tools extend LLM capabilities (e.g., calculator, API calls)

Handle dependency issues via automatic completion (e.g., ToolGen)

Reduce hallucination through retrieval (e.g., Code Agent for web/doc access)

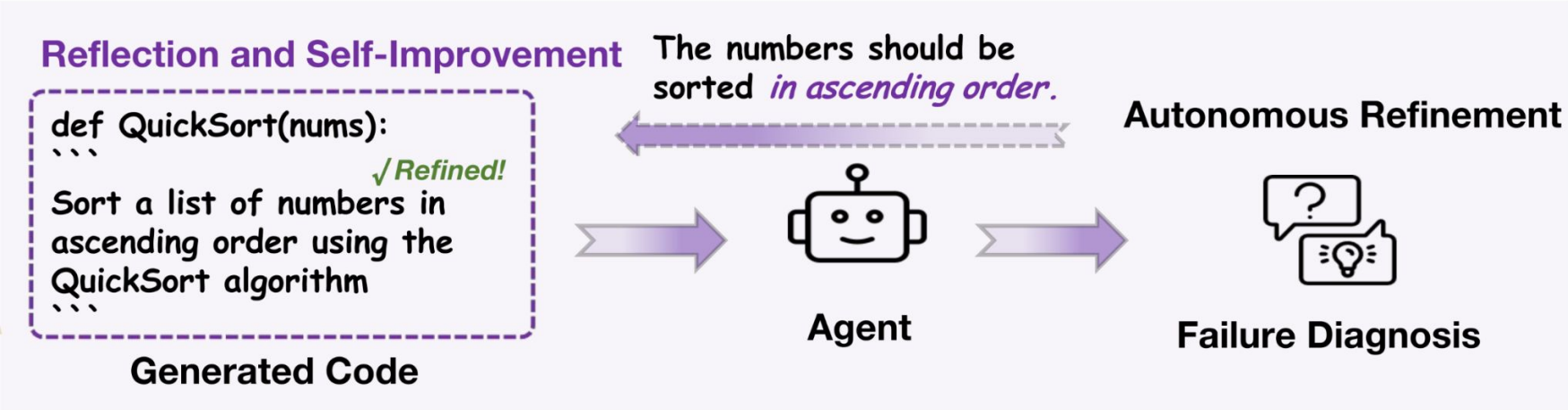
Support debugging via tool feedback (e.g., CodeTool, ROCODE)



Reflection and Self-Improvement

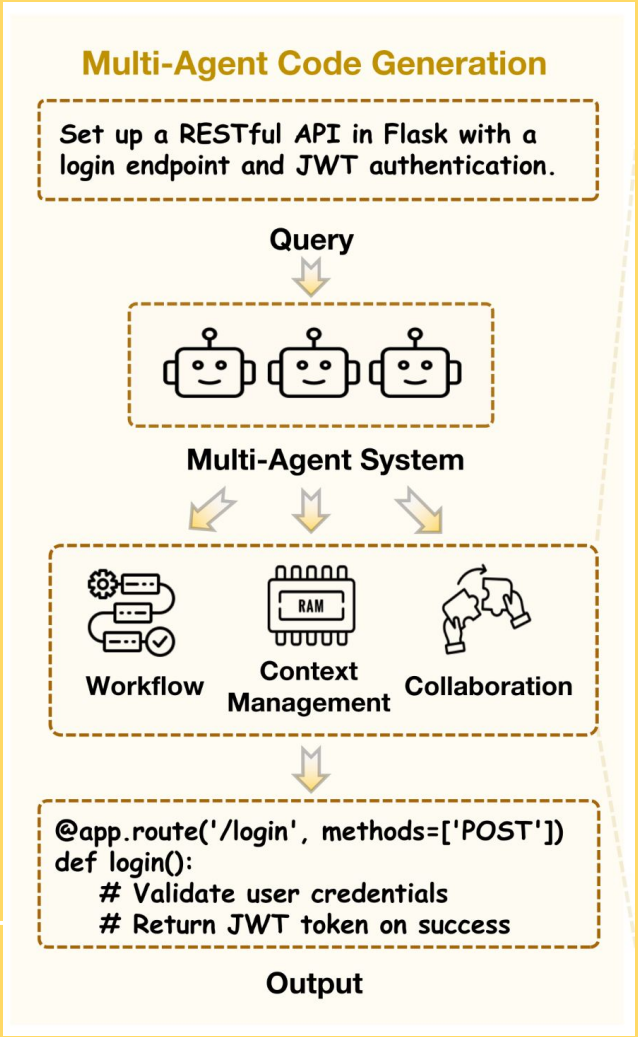
Mimics human process of generating, evaluating and revising code.

Example: Self-Debug (“rubber ducking”)



Multi-Agent

- Multi-Agent System Workflows
- Context Management and Memory Technologies
- Collaborative Optimization of Multi-Agents



Multi-Agent System Workflows

Multi-Agent System Workflows

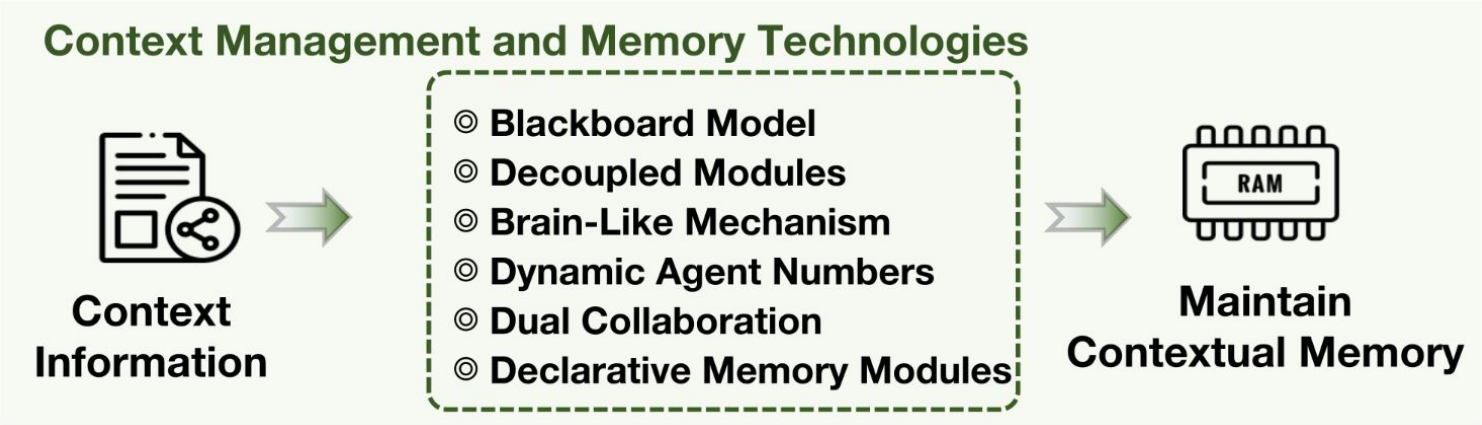
Role-Playing Mechanism

- Aligns agent behavior with defined roles and reasoning patterns
- Enhances task understanding and interaction stability by constraining behavior scope

Context Management and Memory Technologies

Multi-agent systems share long-range context: task descriptions, history, and intermediate results

Examples: Blackboard model (Self-Collaboration), Decoupled Modules (L2MAC)

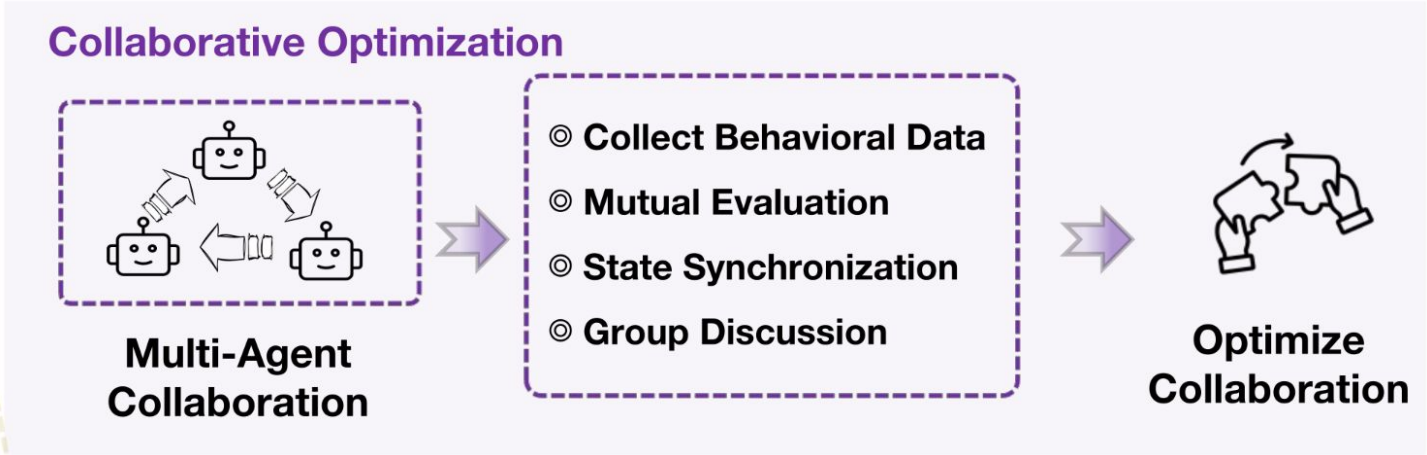


Collaborative Optimization of Multi-Agents

Research now focuses on enhancing inter-agent collaboration in code generation

Still an emerging area with limited and immature methods

Example: Mutual Evaluation (CodeCor)



Applications

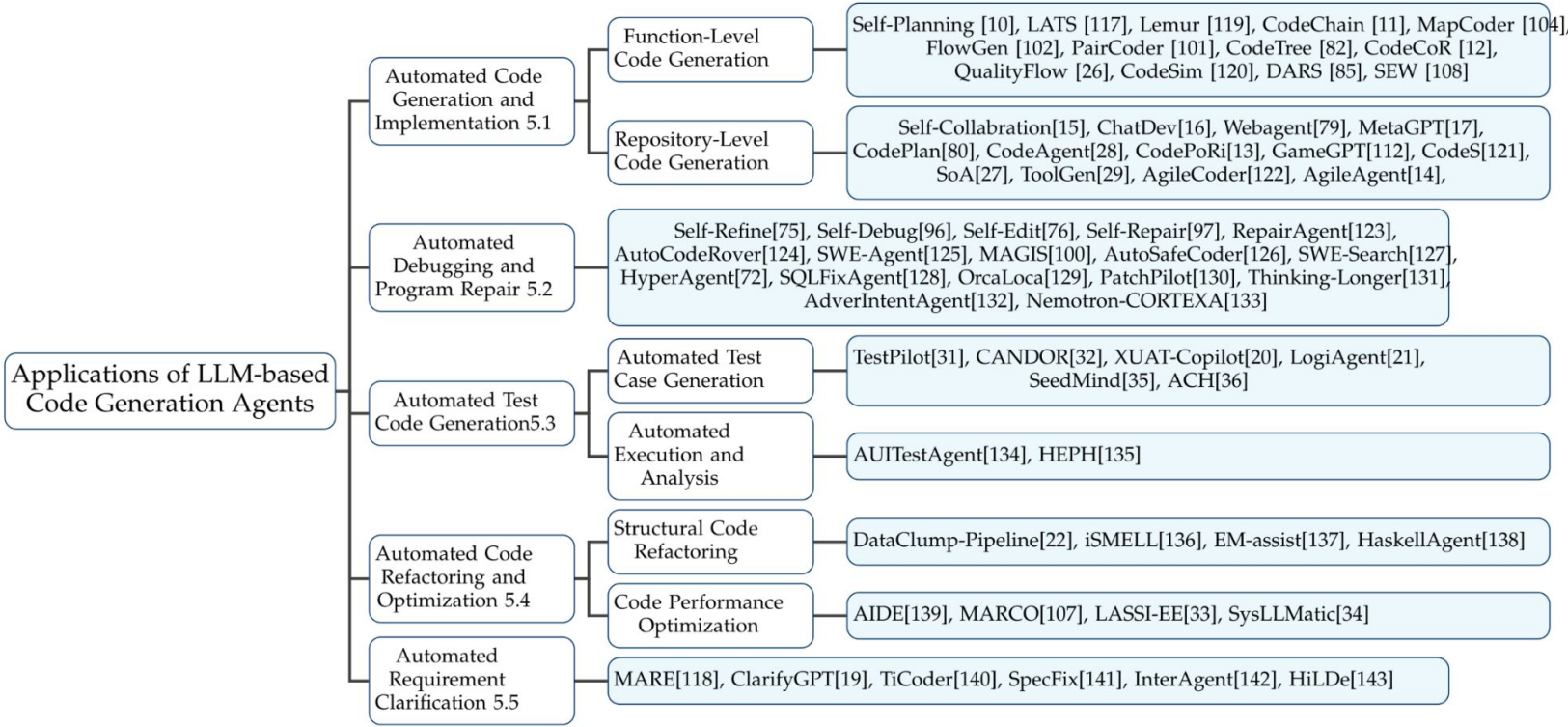


Fig. 5: Overview of applications of LLM-based code generation agents in software development tasks

Evaluation Metrics

- Functional Correctness Metrics
  - Pass@k

$$\text{pass@k} = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

allowed tries

Evaluation Metrics

- Functional Correctness Metrics
  - Pass@k

total number of samples generated for a single problem

correct samples

$$\text{pass@k} = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

allowed tries

to remove the sampling bias

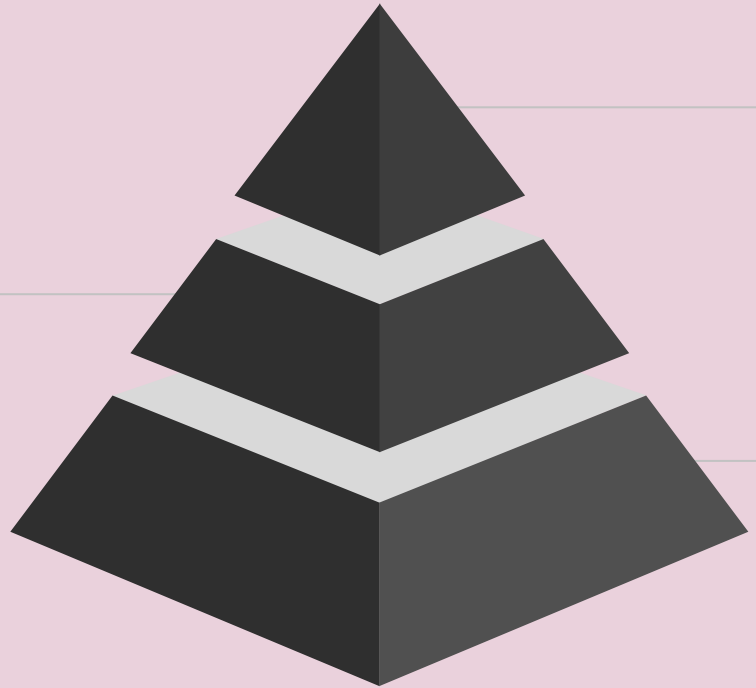
Evaluation Metrics

- Functional Correctness Metrics
  - Pass@k
- Evaluation Metrics for Code Generation Agents
  - efficiency
  - cost
  - task success rate
  - intermediate steps, e.g., tool usage accuracy

Evaluation Metrics

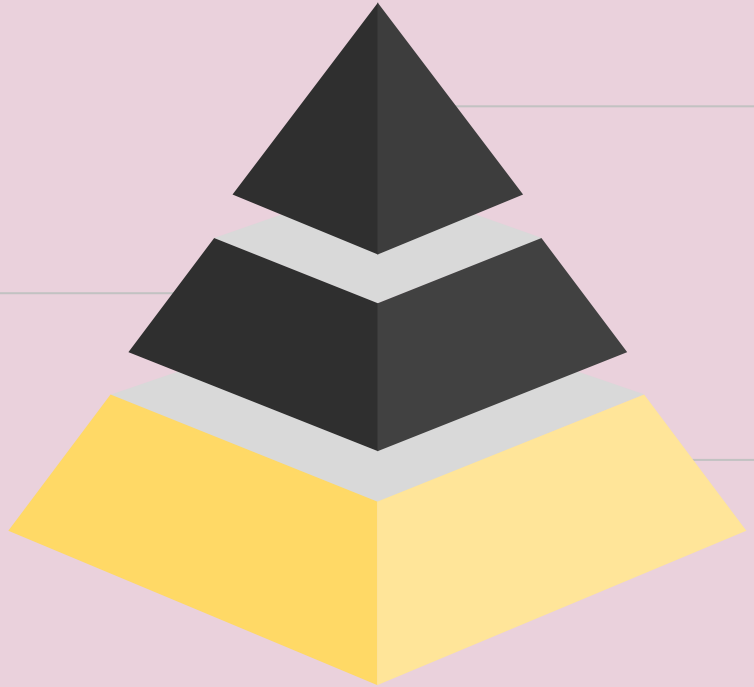
- Functional Correctness Metrics
  - Pass@k
- Evaluation Metrics for Code Generation Agents
  - efficiency
  - cost
  - task success rate
  - intermediate steps, e.g., tool usage accuracy
- Other Software Quality Evaluation Metrics
  - security evaluation (SEC-Bench)
  - code quality (static analysis)

Evaluation Benchmarks



1. **Benchmarks for Real Software Development**
2. **Programming Contest Code Generation**
3. **Method/Class-level Code Generation**

Evaluation Benchmarks



1. **Benchmarks for Real Software Development**
2. **Programming Contest Code Generation**
3. **Method/Class-level Code Generation**

HumanEval (OpenAI, 2021)

- milestone significance
- inspired by online programming platforms
- automatable code execution testing (unit tests)

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

source: Evaluating Large Language Models Trained on Code, https://arxiv.org/abs/2107.03374

Mostly Basic Python Problems Dataset (Google Research,2021)

- 1,000 crowd-sourced, entry-level Python programming tasks
- natural language descriptions, reference answers, 3 assertion test
- problem types from mathematical calculations to basic data processing

Table 4: Qualitative analysis of highest- and lowest-performing problems

	Theme	Examples

Highest-performing problems	Single operations	`Write a function to remove all whitespaces from a string.`
		
		`Write a python function to find the maximum of two numbers.`

	Common "coding interview" type questions	`Write a function to merge multiple sorted inputs into a single sorted iterator`

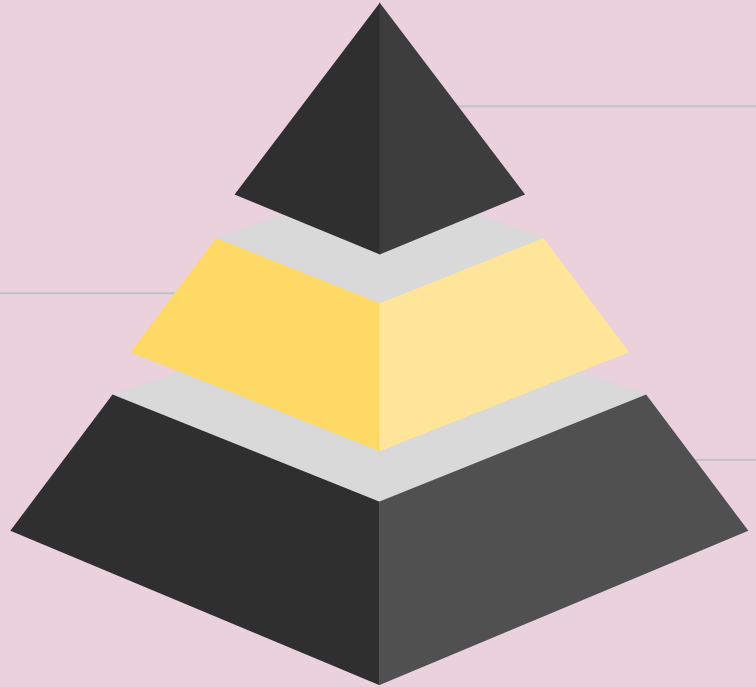
Lowest-performing problems	Problems demanding multiple constraints or multiple sub-problems	`Write a function to find the maximum difference between the number of 0s and number of 1s in any sub-string of the given binary string`
		(Sub-problems: count 0s and 1s, find difference, find max across all sub-strings)
		
		`Write a function to find the longest palindromic subsequence in the given string`
		(Sub-problems: keep track of mirror-imaged letters, find palindromes, find longest one)

	Problems that have a more-common sibling with similar keywords	`Write a python function to find the largest number that can be formed with the given list of digits.`
		(Model solves more-common problem: finds the largest number among the list of digits)
		
		`Write a python function to reverse only the vowels of a given string.`
		(Model solves more-common problem: finds all vowels in the string)

	Specialized math problems	`Write a function to find eulerian number a(n, m).`

source: Program Synthesis with Large Language Models, https://arxiv.org/abs/2108.07732

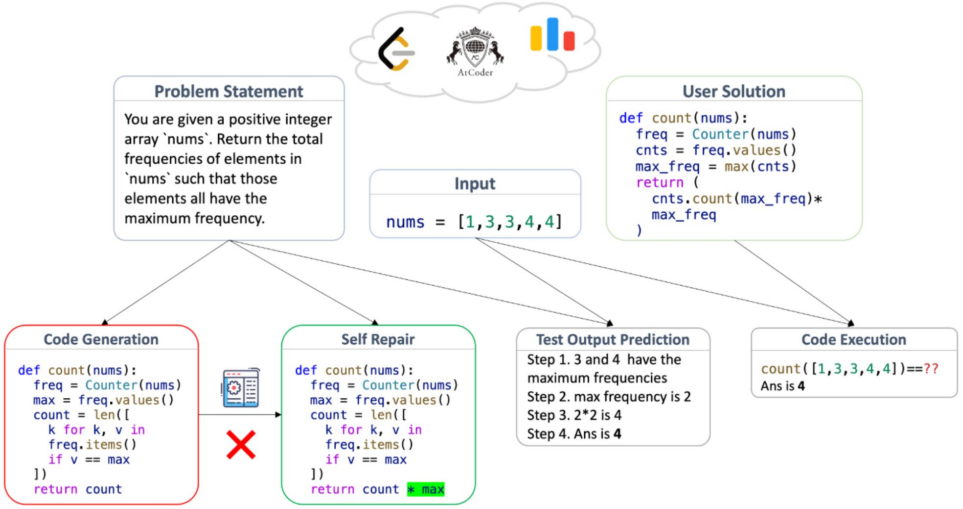
Evaluation Benchmarks



1. **Benchmarks for Real Software Development**
2. **Programming Contest Code Generation**
3. **Method/Class-level Code Generation**

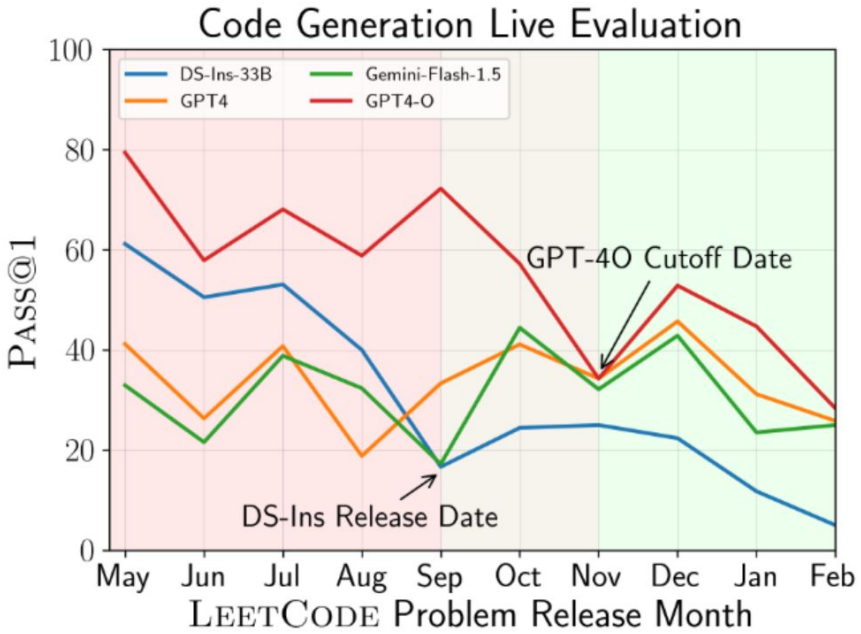
LiveCodeBench (UC Berkeley,MIT,Cornell University)

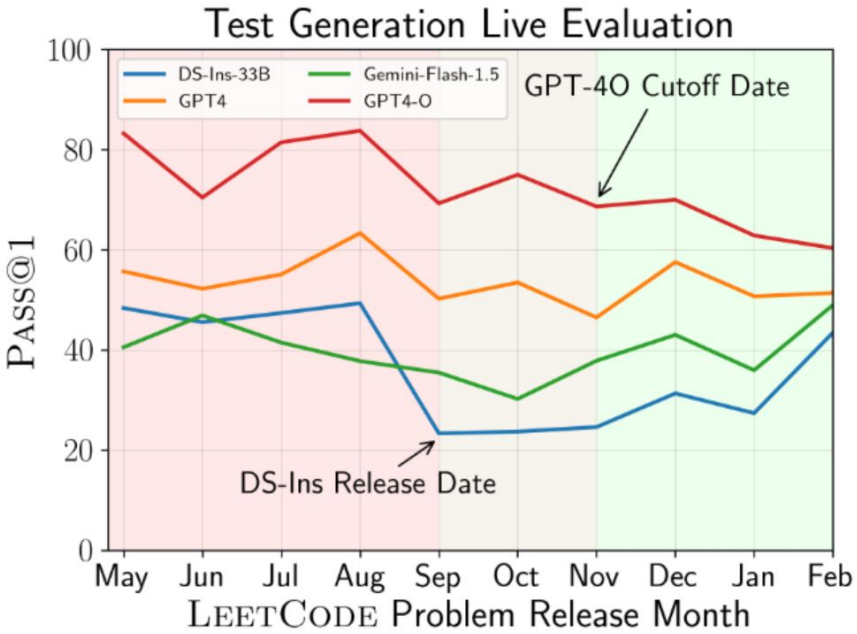
- Holistic and contamination-free
- Focuses on broader code-related capabilities, such as self-repair, code execution, and test output prediction
- 2024, but updated continuously



source: https://livecodebench.github.io/

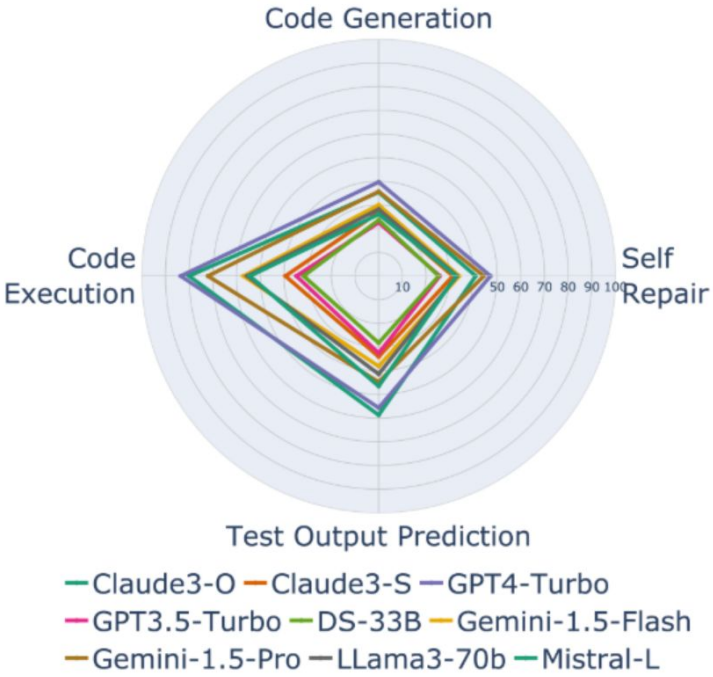
LiveCodeBench (UC Berkeley,MIT,Cornell University)

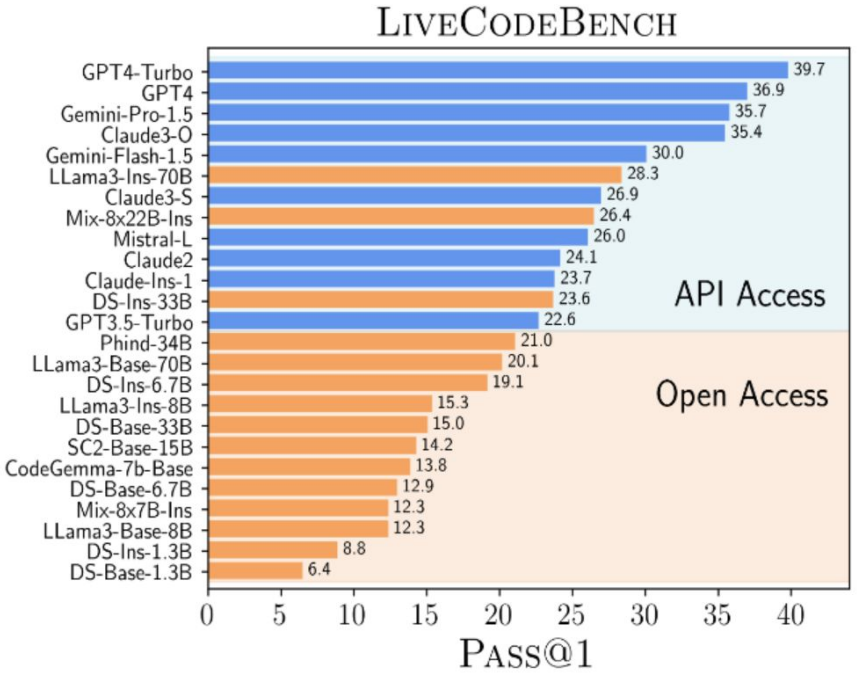




source: https://livecodebench.github.io/

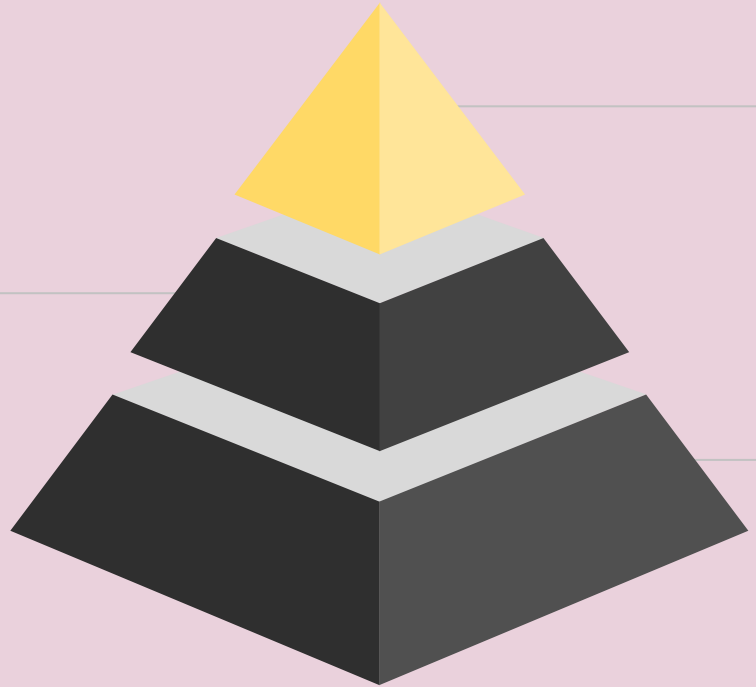
LiveCodeBench (UC Berkeley,MIT,Cornell University)





source: https://livecodebench.github.io/

Evaluation Benchmarks



1 **Benchmarks for Real Software Development**

2 **Programming Contest Code Generation**

3 **Method/Class-level Code Generation**

SWE-Bench

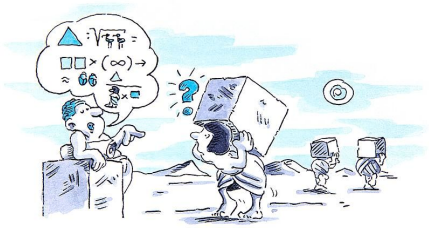
- Real-world GitHub issue resolution benchmark (thousands of samples)
- versions: Lite, Verified, Bash Only, Multilingual, Multimodal

Model	% Resolved	Org	Date	Site
✅ SWE-agent 1.0 (Claude 3.7 Sonnet)	33.83		2025-02-27	
✅ OpenHands + CodeAct v2.1 (claude-3-5-sonnet-20241022)	29.38		2024-11-03	
AutoCodeRover-v2.0 (Claude-3.5-Sonnet-20241022)	24.89		2024-11-21	
✅ SWE-agent + Claude 3.5 Sonnet	18.13		2024-06-20	-
✅ SWE-agent + GPT 4 (1106)	12.47		2024-04-02	
✅ SWE-agent + GPT 4o (2024-05-13)	11.99		2024-07-28	
✅ SWE-agent + Claude 3 Opus	10.51		2024-04-02	-
✅ RAG + Claude 3 Opus	3.79		2024-04-02	

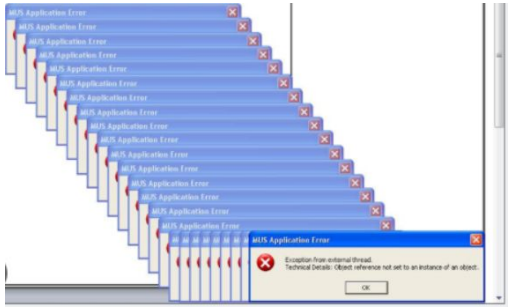
source: https://www.swebench.com/

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



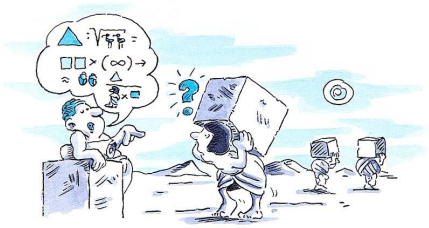
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

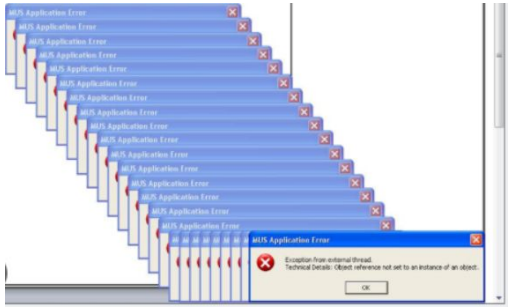
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



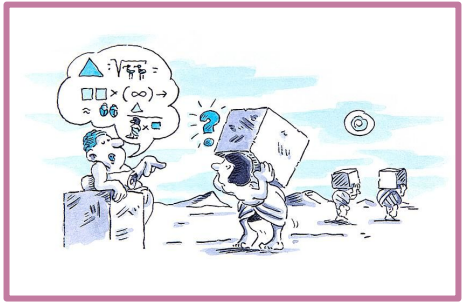
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

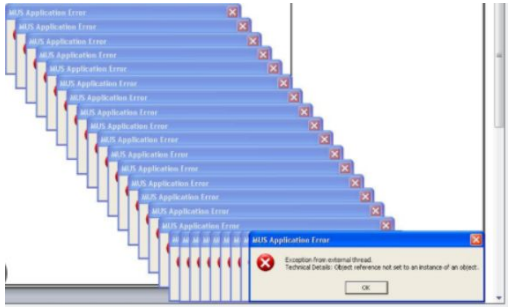
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



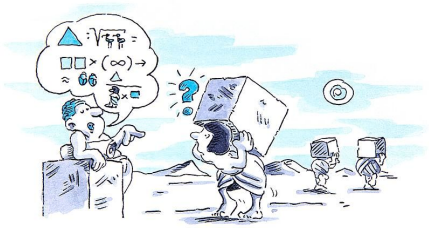
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

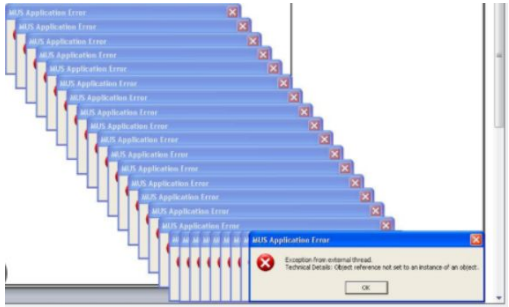
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



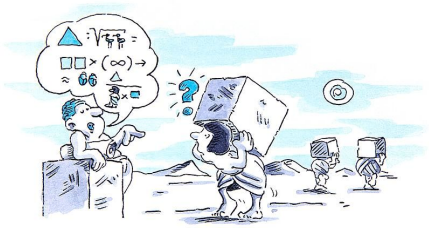
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

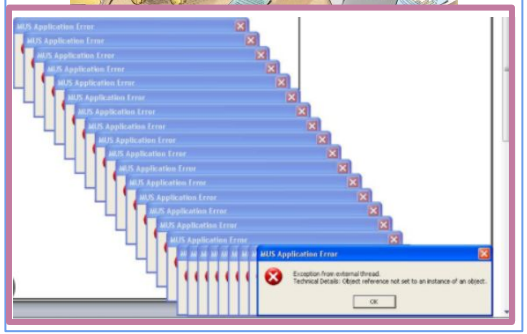
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



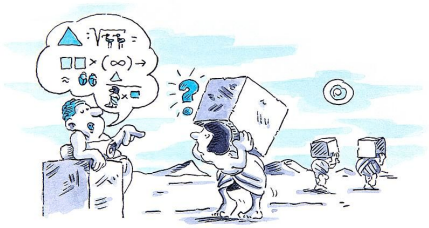
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

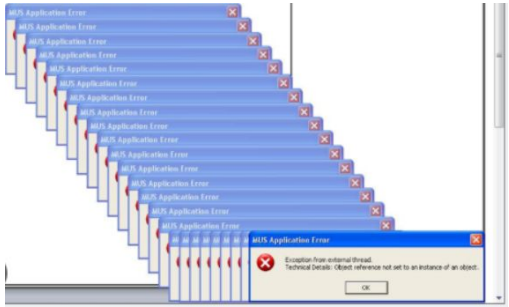
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



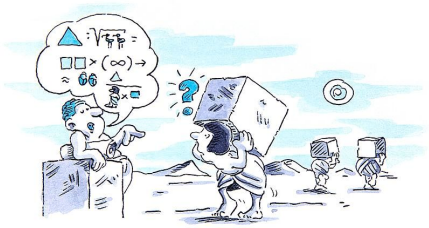
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

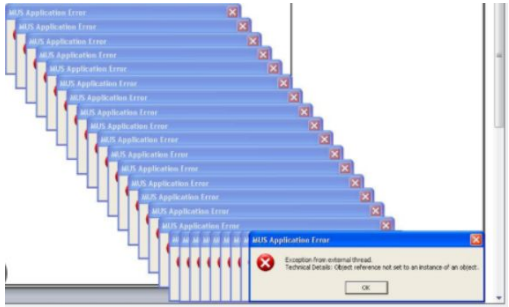
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



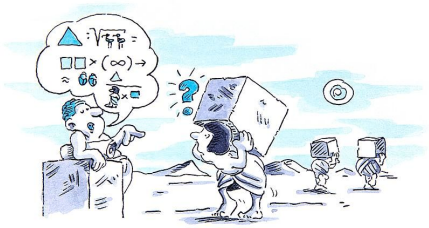
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

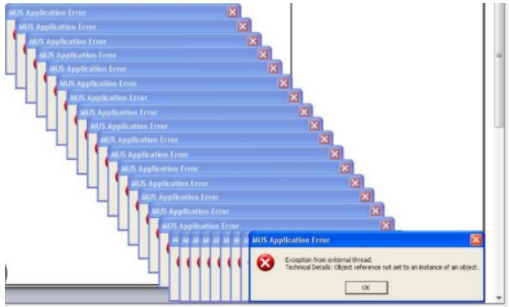
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



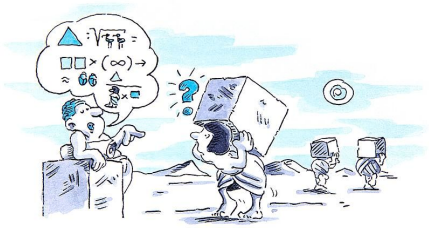
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

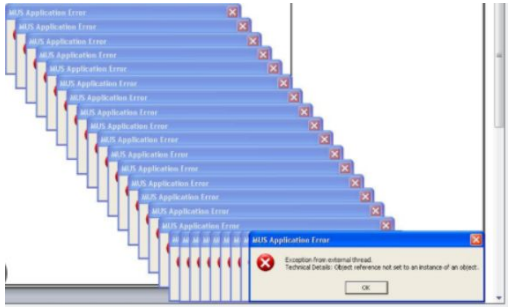
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



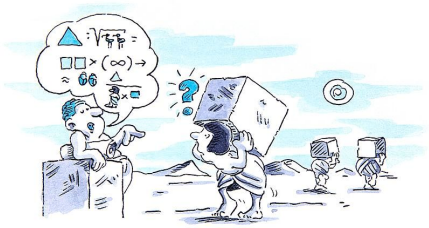
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

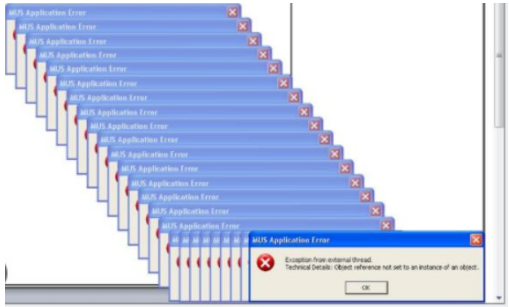
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



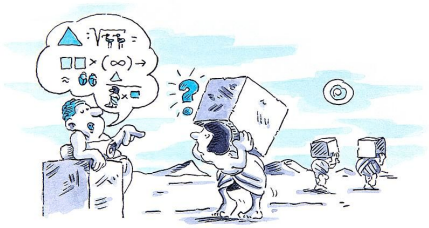
Evaluation system and Software Paradigm Evolution

Trustworthiness, Security, and Ethical Risks

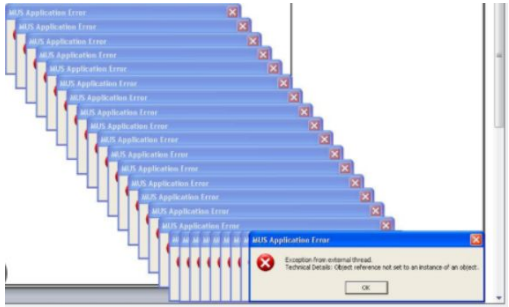
Challenges of Tool Integration and Deployment for Agents in Open Environments

Challenges

Limitations of Agent Core Capabilities



Robustness and Updatability Challenges of Agent Systems



Evaluation system and Software Paradigm Evolution

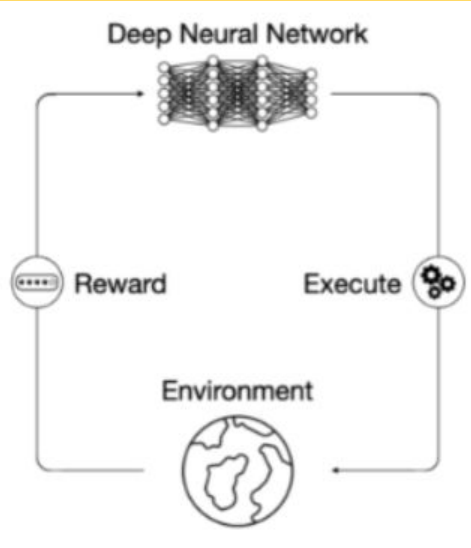
Trustworthiness, Security, and Ethical Risks

Challenges of Tool Integration and Deployment for Agents in Open Environments

Programmatic Reinforcement Learning

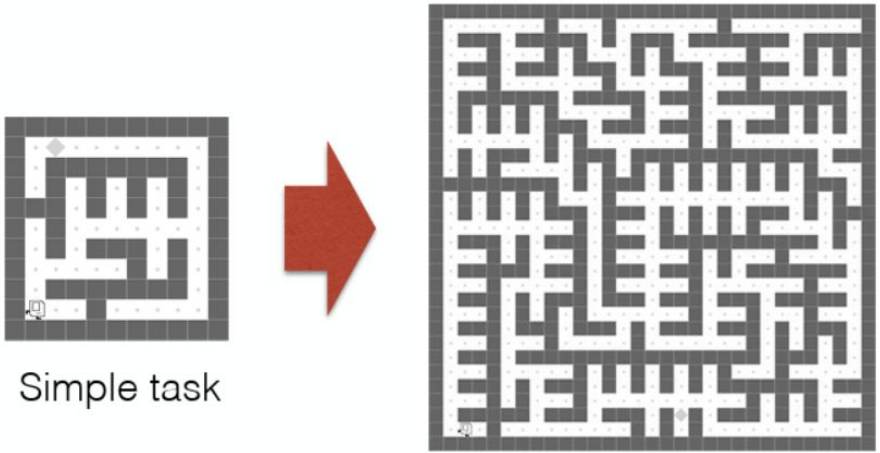
Reinforcement Learning (RL)

Deep Reinforcement Learning (DRL)

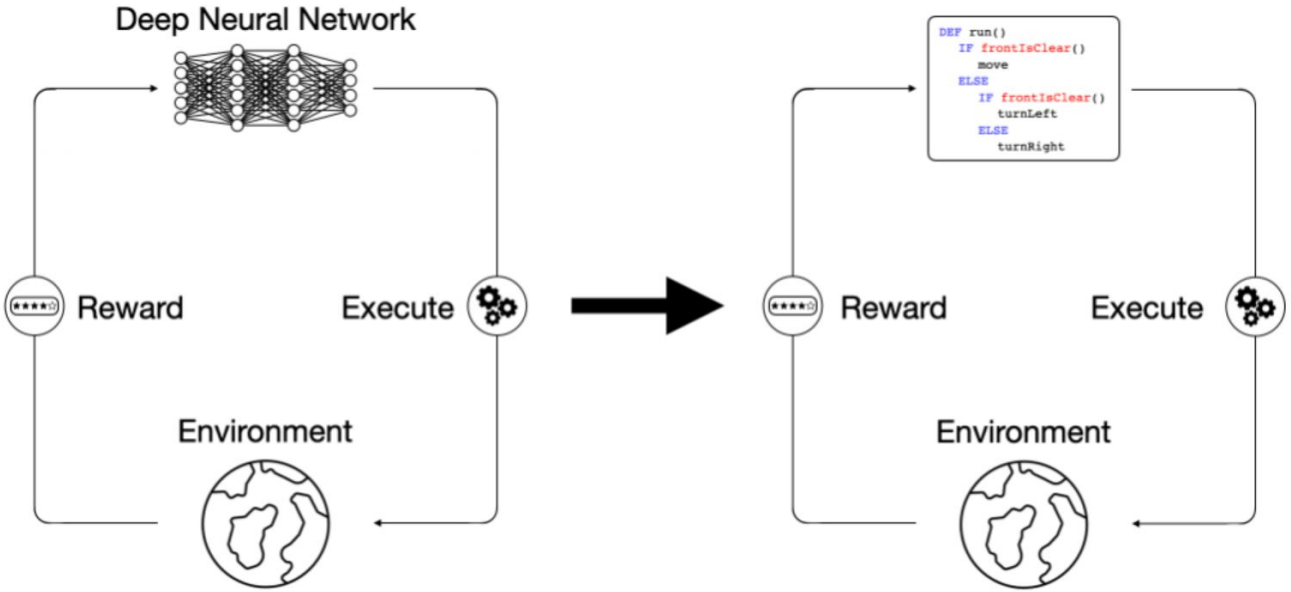


Problems with DRL methods

- Interpretability
- Generability

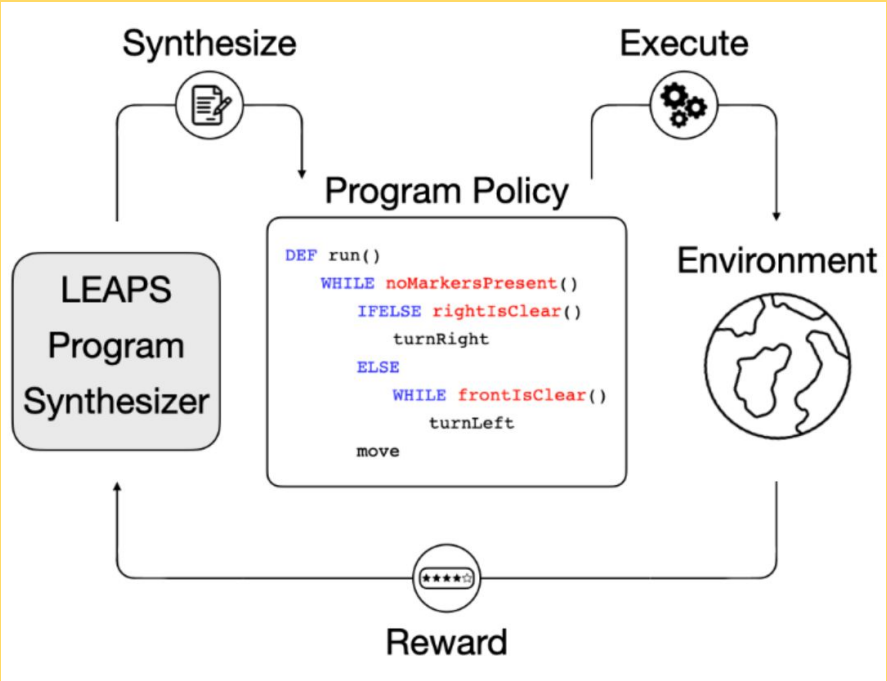


Code as a Policy



“Learning to Synthesize Programs as Interpretable and Generalizable Policies”, 2022, Trivedi et al.

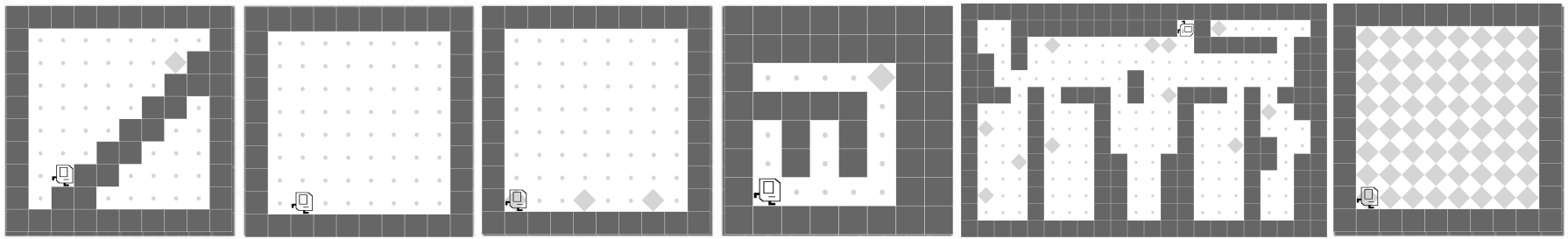
Learning Embeddings for lAtent Program Synthesis (LEAPS)



Problem Formulation

- Learn to synthesize a program in a given DSL
- The program executes to solve a task defined by an MDP
- Learning is driven purely by reward signals

Karel Domain



(a) StairClimber (b) FourCorner (c) TopOff (d) Maze (e) CleanHouse (f) Harvester

Figure 3: **The Karel problem set**: the domain features an agent navigating through a gridworld with walls and interacting with markers, allowing for designing tasks that demand certain behaviors. The tasks are further described in Section K with visualizations in Figure 15.

source: https://clvrai.github.io/leaps/

Problem Formulation

- Learn to synthesize a program in a given DSL
- The program executes to solve a task defined by an MDP
- Learning is driven purely by reward signals

Program	ρ	:= DEF run m(s m)
Repetition	n	:= Number of repetitions
Perception	h	:= Domain-dependent perceptions
Condition	b	:= perception h | not perception h
Action	a	:= Domain-dependent actions
Statement	s	:= while c(b c) w(s w) | s_1; s_2 | a |
		repeat R=n r(s r) | if c(b c) i(s i) |
		ifelse c(b c) i(s_1 i) else e(s_2 e)

Figure 1: The domain-specific language (DSL) for constructing programs.

Problem Formulation

- Learn to synthesize a program in a given DSL
- The program executes to solve a task defined by an MDP
- Learning is driven purely by reward signals

Program	ρ	:= DEF run m(s m)
Repetition	n	:= Number of repetitions
Perception	h	:= Domain-dependent perceptions
Condition	b	:= perception h | not perception h
Action	a	:= Domain-dependent actions
Statement	s	:= while c(b c) w(s w) | s_1; s_2 | a | repeat R=n r(s r) | if c(b c) i(s i) | ifelse c(b c) i(s_1 i) else e(s_2 e)

Figure 1: The domain-specific language (DSL) for constructing programs.

Objective: $\max_\rho \mathbb{E}_{a \sim \text{EXEC}(\rho), s_0 \sim \mu}[\sum_{t=0}^{T} \gamma^t r_t]$

EXEC returns the actions induced by executing a program policy ρ in the environment.

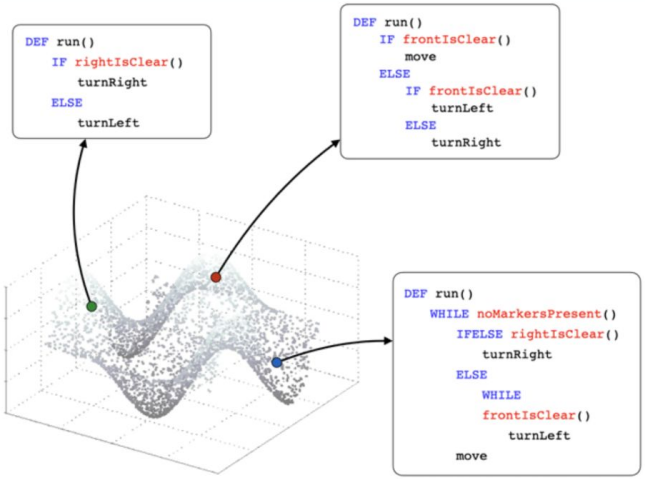
LEAPS - the Overview

Stage 1

Learning a program embedding space from a set of randomly generated programs

Grammar

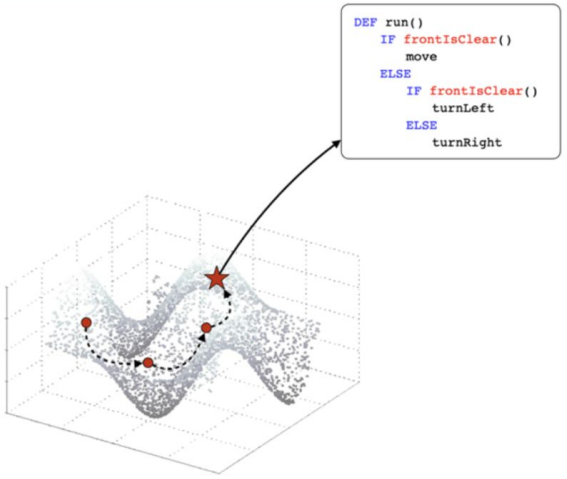
Environment Dynamics



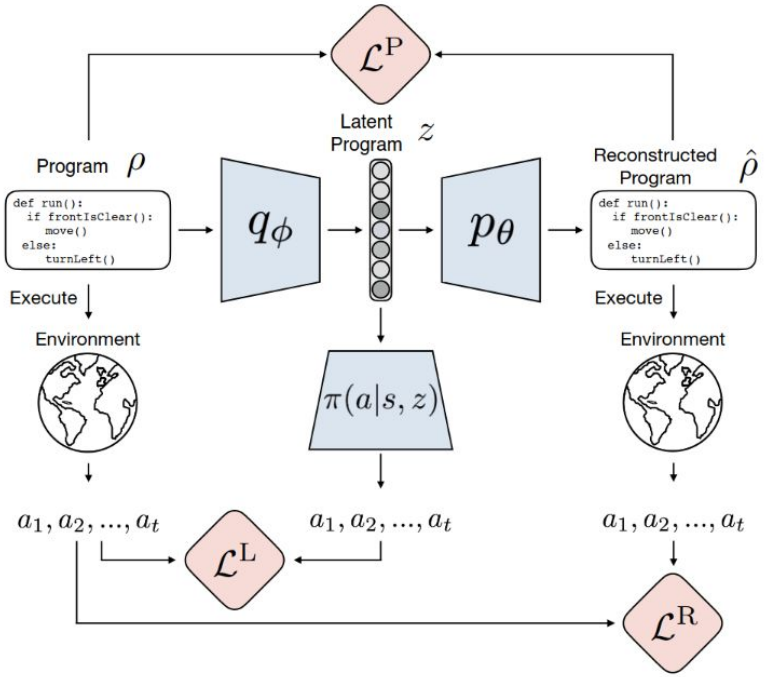
Stage 2

Searching for a task-solving program

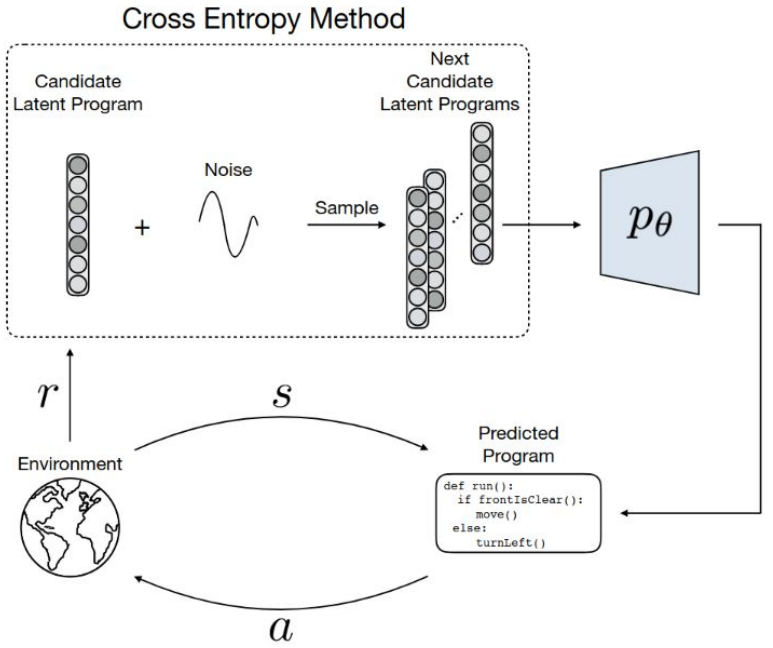
Desired Behavior



LEAPS - the Overview

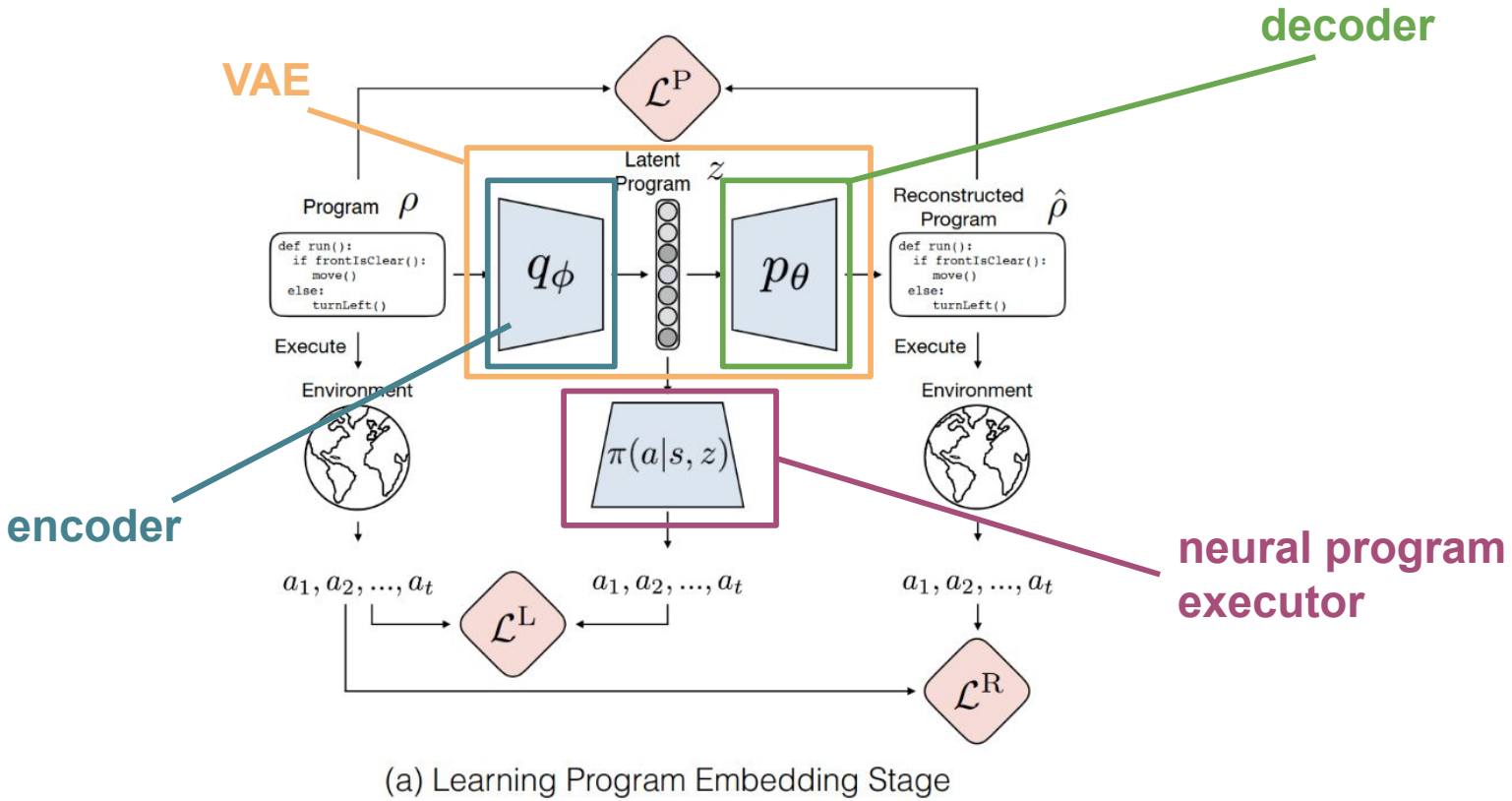


(a) Learning Program Embedding Stage



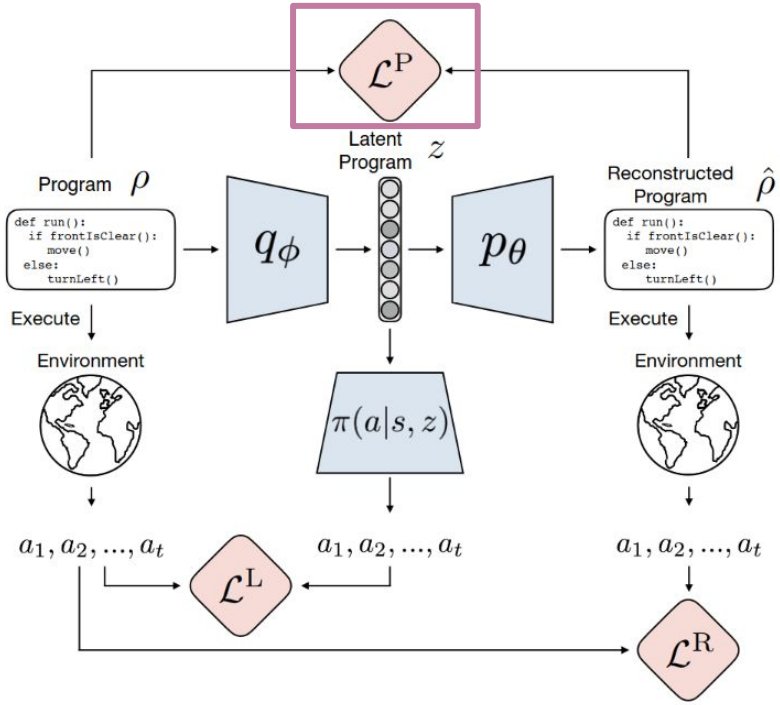
(b) Latent Program Search Stage

Learning Program Embedding Stage



(a) Learning Program Embedding Stage

Program Reconstruction

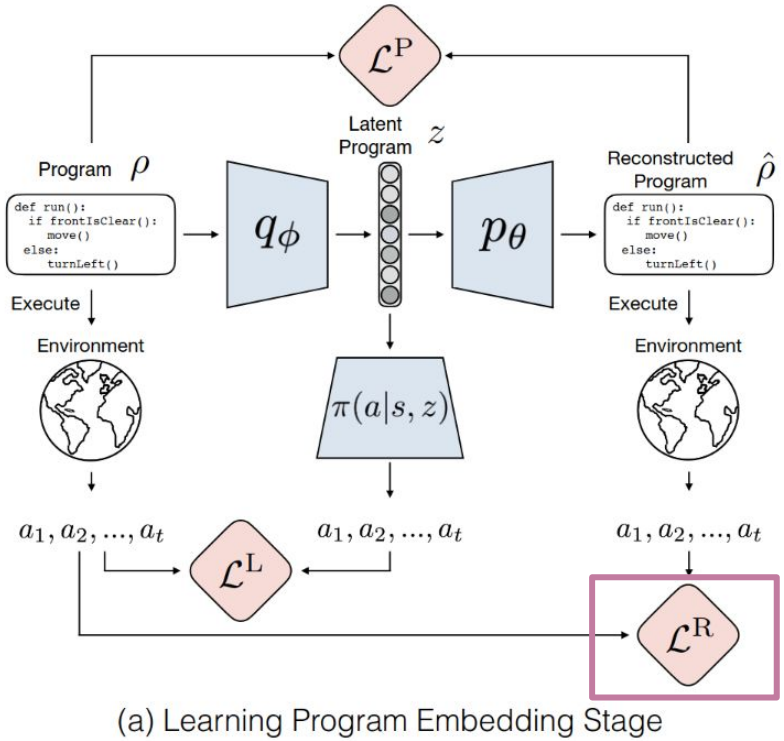


(a) Learning Program Embedding Stage

$$\mathcal{L}^{\mathrm{P}}_{\theta,\phi}(\rho) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\rho)}[\log p_\theta(\boldsymbol{\rho}|\mathbf{z})] + \beta D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\boldsymbol{\rho})\|p_\theta(\mathbf{z}))$$

β-VAE loss is used to enforce “text similarity”

Program Behaviour Reconstruction



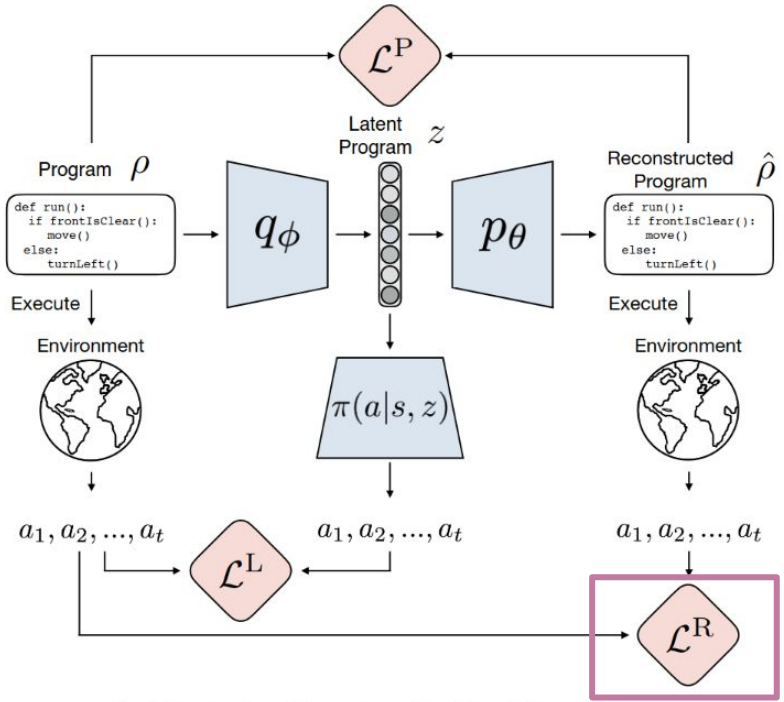
(a) Learning Program Embedding Stage

These two programs are semantically the same:

repeat(2): move()

move() move()

Program Behaviour Reconstruction



(a) Learning Program Embedding Stage

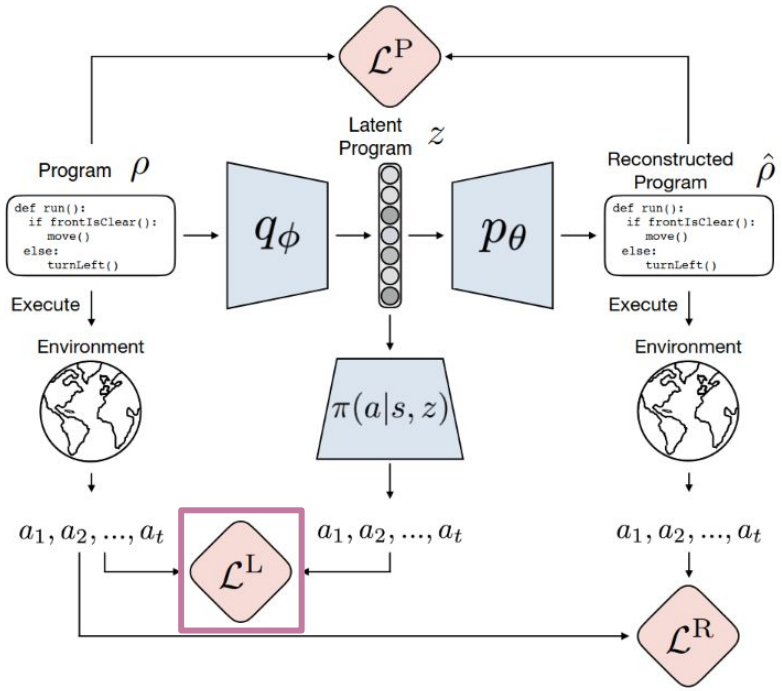
$$\mathcal{L}^{\mathrm{R}}_{\theta,\phi}(\rho) = -\mathbb{E}_{z\sim q_\phi(z|\rho)}[R_{\mathrm{mat}}(p_\theta(\rho|z),\rho)]$$

REINFORCE loss is used to enforce "semantic similarity"

where **reward** for matching behaviour is defined as:

$$R_{\mathrm{mat}}(\hat{\rho},\rho) = \mathbb{E}_\mu[\frac{1}{N}\sum_{t=1}^{N}\underbrace{\mathbb{1}\{\mathrm{EXEC}_i(\hat{\rho}) == \mathrm{EXEC}_i(\rho)\ \forall i = 1,2,...t\}}_{\text{stays 0 after the first } t \text{ where } \mathrm{EXEC}_t(\hat{\rho})\ !=\ \mathrm{EXEC}_t(\rho)}]$$

Latent Behaviour Reconstruction

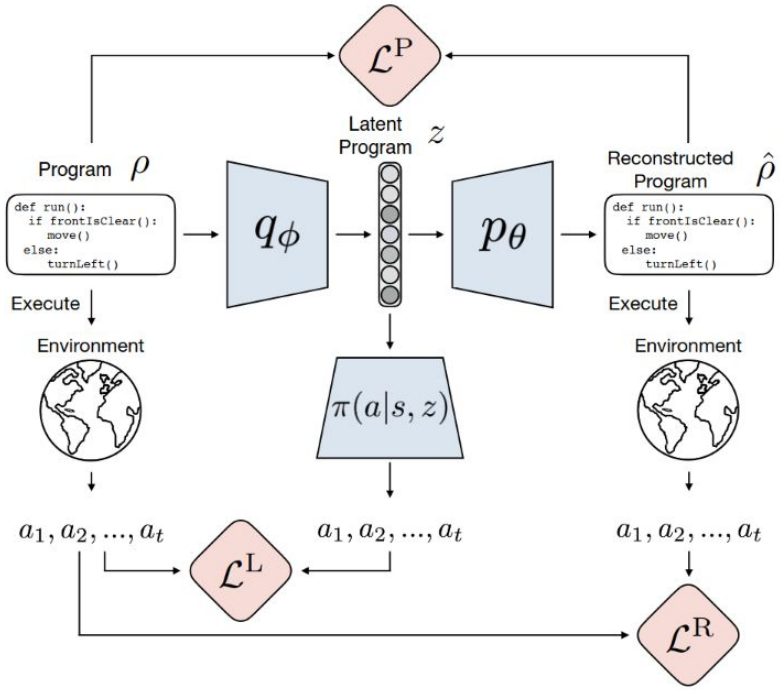


(a) Learning Program Embedding Stage

$$\mathcal{L}^{\mathrm{L}}_{\pi}(\rho,\pi) = -\mathbb{E}_{\mu}[\sum_{t=1}^{M}\sum_{i=1}^{|\mathcal{A}|}\mathbb{1}\{\mathrm{EXEC}_i(\hat{\rho}) == \mathrm{EXEC}_i(\rho)\}\log\pi(a_i|z,s_t)]$$

cross entropy loss to encourage learning a program embedding space that allows smooth behaviour interpolation

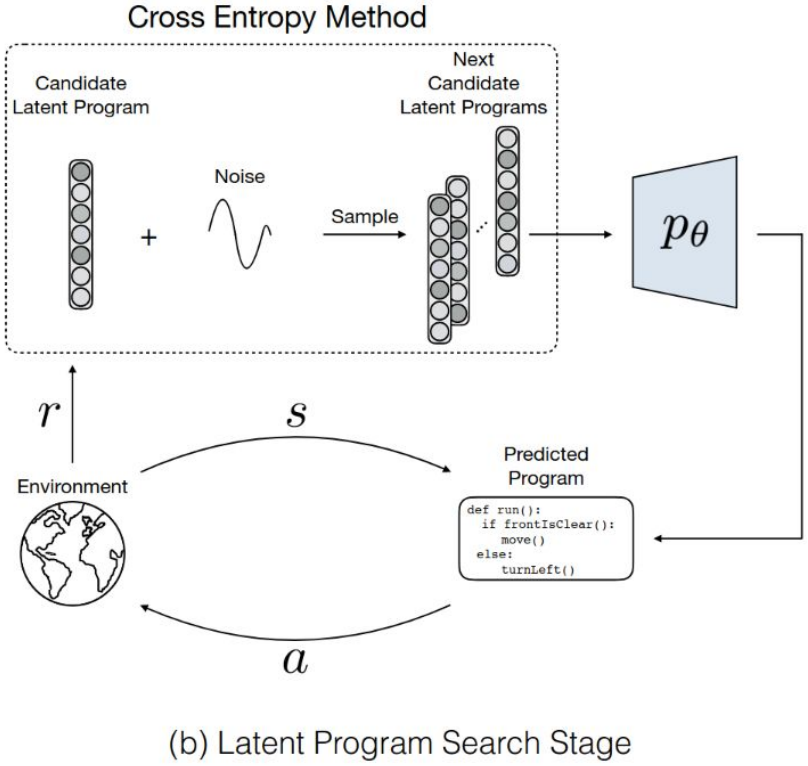
Combined Objective



(a) Learning Program Embedding Stage

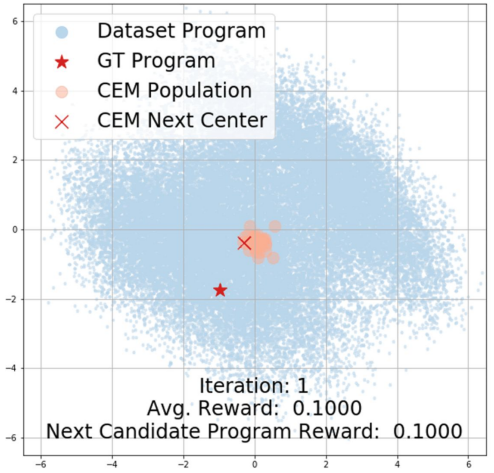
$$\min_{\theta,\phi,\pi} \lambda_1 \mathcal{L}^{\mathrm{P}}_{\theta,\phi}(\rho) + \lambda_2 \mathcal{L}^{\mathrm{R}}_{\theta,\phi}(\rho) + \lambda_3 \mathcal{L}^{\mathrm{L}}_{\pi}(\rho, \pi)$$

Latent Program Search



(b) Latent Program Search Stage

CEM finds a program by only considering rewards



Training

- Data 50k unique programs, generated independently of Karel tasks
  - 35 train set, 7.5k val set, 7,5k test set
- For each program, they sample random Karel states and execute the program on them from different starting states

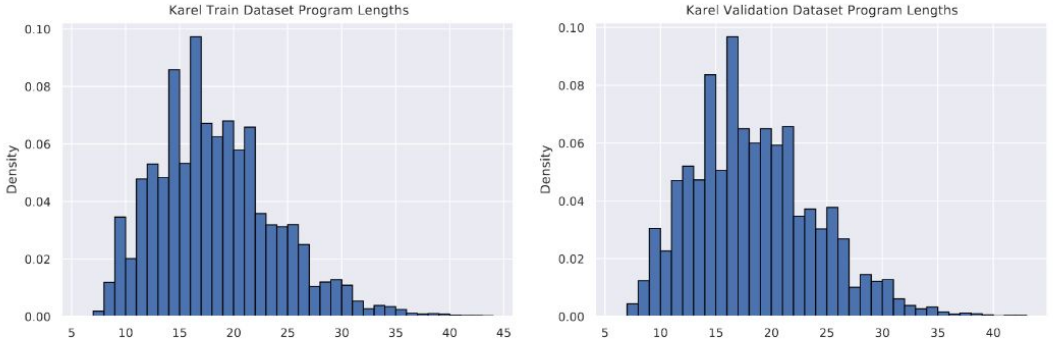


Figure 14: Histograms of the program length (*i.e.* number of program tokens) in the training and validation datasets.

Ablation Study

So all these losses are useful and necessary

Search (CEM) is important too

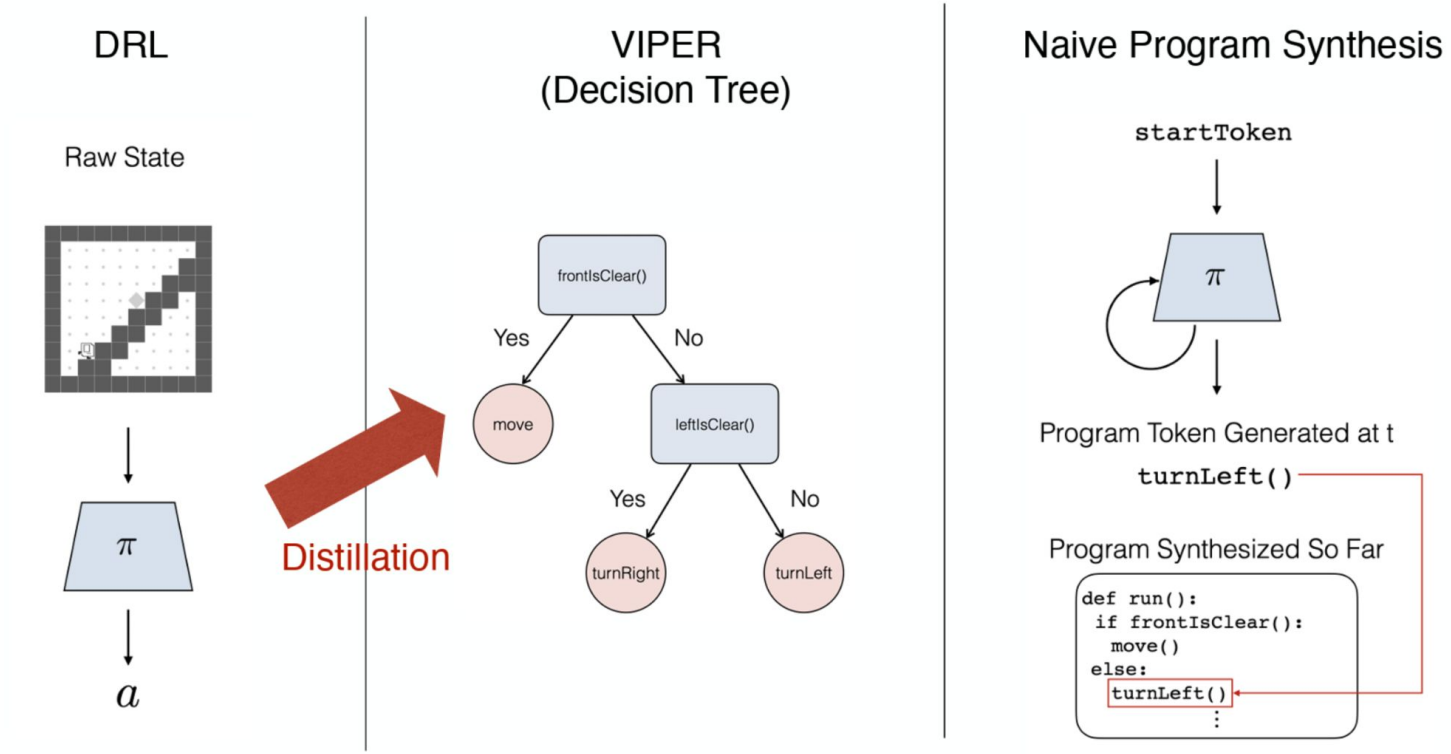
Table 2: Program embedding space smoothness. For each program, we execute the ten nearest programs in the learned embedding space of each model to calculate the mean state-matching reward R_{mat} against the original program. We report R_{mat} averaged over all programs in each dataset.

	LEAPS-P	LEAPS-P+R	LEAPS-P+L	LEAPS
TRAINING	0.22	0.22	**0.31**	**0.31**
VALIDATION	0.22	0.21	**0.27**	**0.27**
TESTING	0.22	0.22	**0.28**	0.27

Table 1: Program behavior reconstruction rewards (standard deviations) across all methods.

	WHILE	IFELSE+WHILE	2IF+IFELSE	WHILE+2IF+IFELSE	Avg Reward
Naïve	0.65 (0.33)	0.83 (0.07)	0.61 (0.33)	0.16 (0.06)	0.56
LEAPS-P	0.95 (0.13)	0.82 (0.08)	0.58 (0.35)	0.33 (0.17)	0.67
LEAPS-P+R	0.98 (0.09)	0.77 (0.05)	0.63 (0.25)	0.52 (0.27)	0.72
LEAPS-P+L	**1.06** (0.00)	0.84 (0.10)	0.77 (0.23)	0.33 (0.13)	0.75
LEAPS-rand-8	0.62 (0.24)	0.49 (0.09)	0.36 (0.18)	0.28 (0.14)	0.44
LEAPS-rand-64	0.78 (0.22)	0.63 (0.09)	0.55 (0.20)	0.37 (0.09)	0.58
LEAPS	**1.06** (0.08)	**0.87** (0.13)	**0.85** (0.30)	**0.57** (0.23)	**0.84**

Baselines

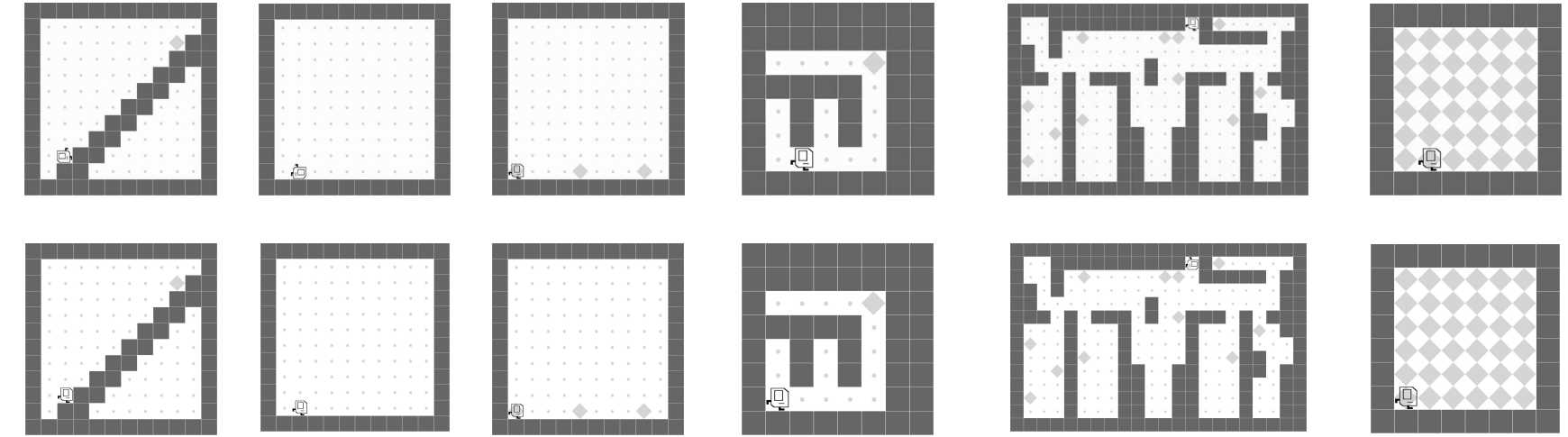


source: https://clvrai.github.io/leaps/

Results

DRL

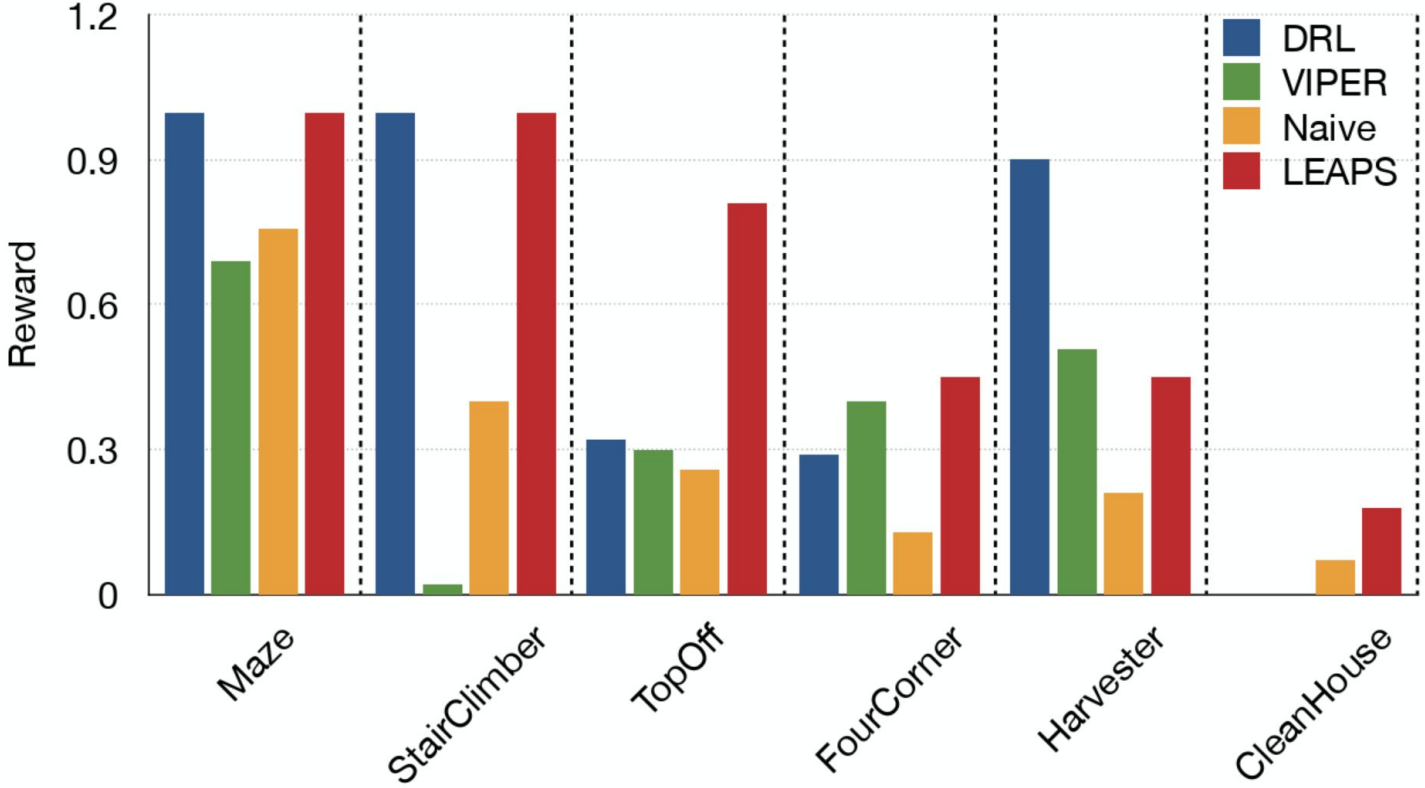
LEAPS



(a) StairClimber (b) FourCorner (c) TopOff (d) Maze (e) CleanHouse (f) Harvester

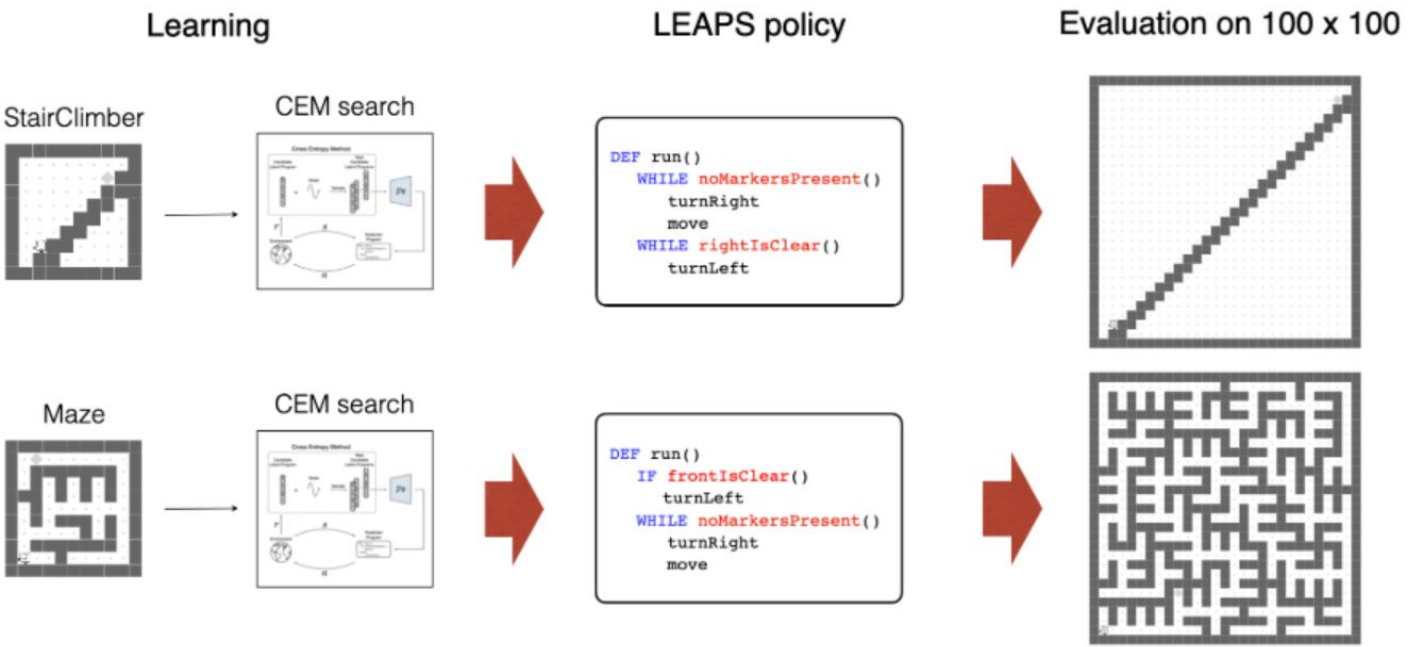
source: https://clvrai.github.io/leaps/

Results - Karel



source: https://clvrai.github.io/leaps/

Generalization Experiment (one-shot)



source: https://clvrai.github.io/leaps/

Generalization Experiment (one-shot)

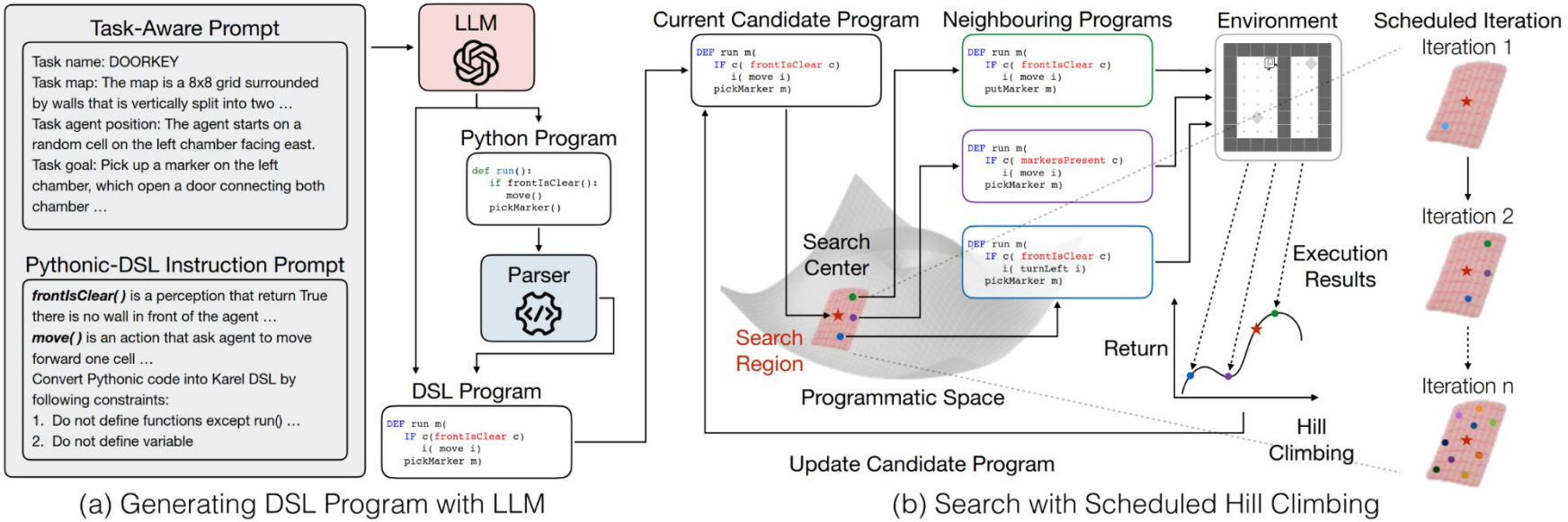
Table 9: Extended reward comparison on original tasks with 8×8 or 12×12 grids and zero-shot generalization to 100×100 grids. LEAPS achieves the best generalization performance on all the tasks except for HARVESTER.

		STAIRCLIMBER	MAZE	FOURCORNER	TOPOFF	HARVESTER
DRL	Original	**1.00** (0.00)	**1.00** (0.00)	0.29 (0.05)	0.32 (0.07)	**0.90** (0.10)
	100x100	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.01 (0.01)	0.00 (0.00)
DRL-abs	Original	0.13 (0.29)	**1.00** (0.00)	0.36 (0.44)	0.63 (0.23)	0.32 (0.18)
	100x100	0.00 (0.00)	0.04 (0.05)	0.37 (0.44)	0.15 (0.12)	0.02 (0.01)
DRL-FCN	Original	**1.00** (0.00)	0.97 (0.03)	0.20 (0.34)	0.28 (0.12)	0.46 (0.16)
	100x100	-0.20 (0.10)	0.01 (0.01)	0.00 (0.00)	0.01 (0.01)	0.02 (0.00)
VIPER	Original	0.02 (0.02)	0.69 (0.05)	0.40 (0.42)	0.30 (0.06)	0.51 (0.07)
	100x100	0.00 (0.00)	0.10 (0.12)	0.40 (0.42)	0.03 (0.00)	**0.04** (0.00)
LEAPS	Original	**1.00** (0.00)	**1.00** (0.00)	**0.45** (0.40)	**0.81** (0.07)	0.45 (0.28)
	100x100	**1.00** (0.00)	**1.00** (0.00)	**0.45** (0.37)	**0.21** (0.03)	0.00 (0.00)

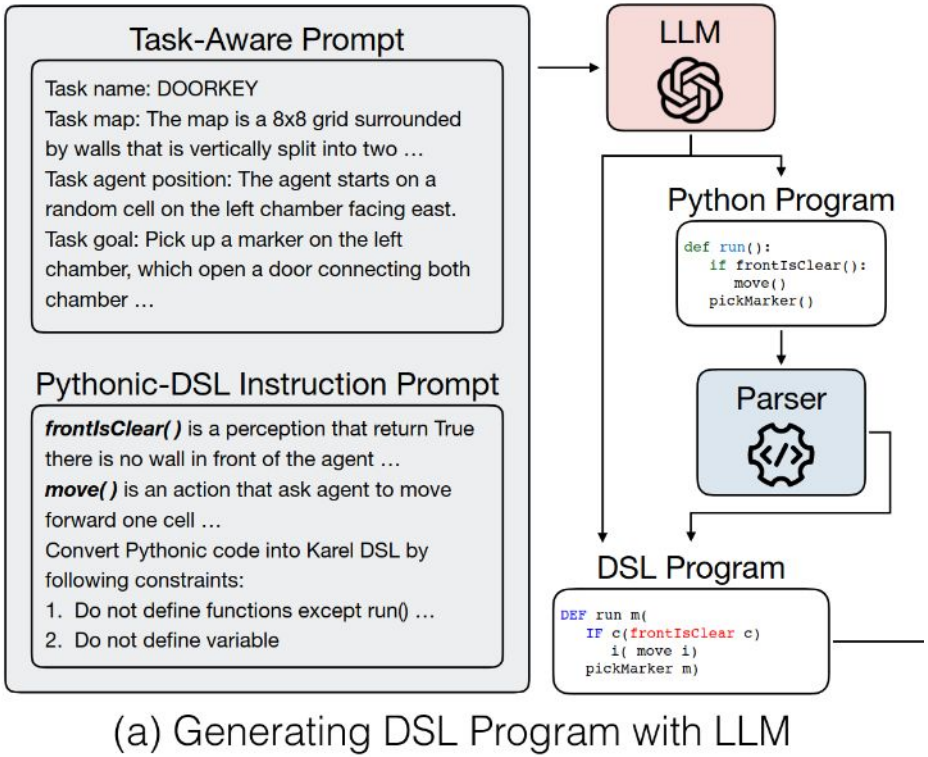
Can we take advantage of LLMs in this framework?

Programmatic RL with LLM Guided Search

“SYNTHESIZING PROGRAMMATIC REINFORCEMENT LEARNING POLICIES WITH LARGE LANGUAGE MODEL GUIDED SEARCH”, 2025, Liu et al.



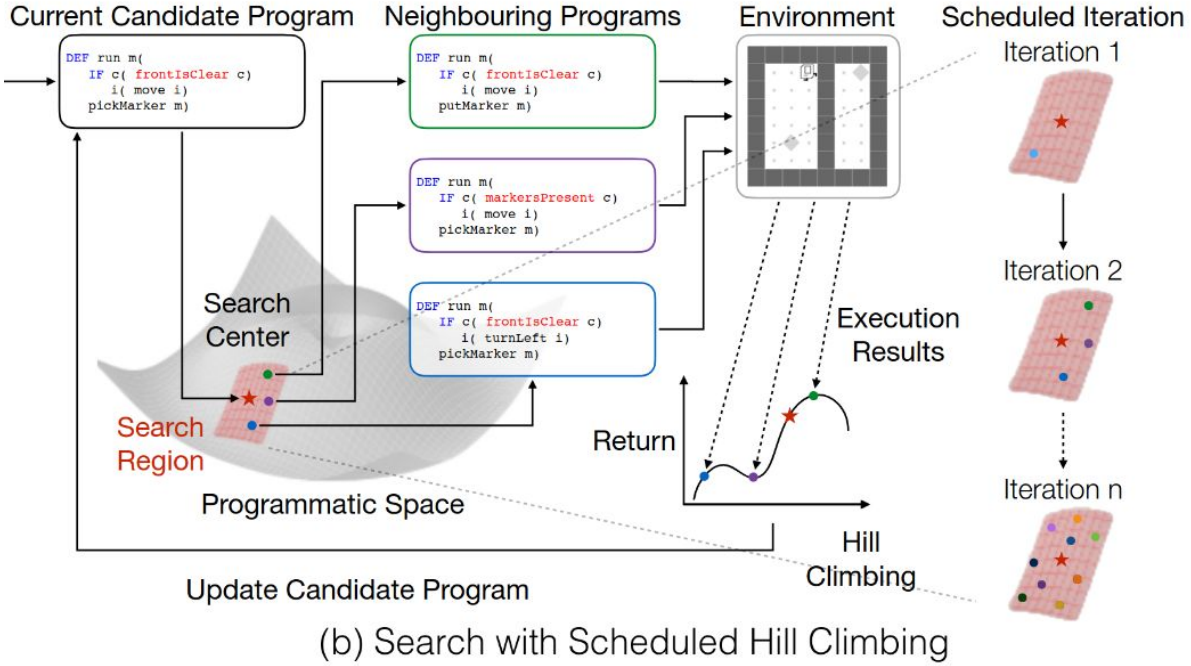
Generating DSL Program with LLM



(a) Generating DSL Program with LLM

LLM might not know DSL and then it's hard for it to generate syntactically correct code immediately

Search with Scheduled Hill Climbing



(b) Search with Scheduled Hill Climbing

This time they test more search techniques (HC, CEM, CEBS)

Karel experiment

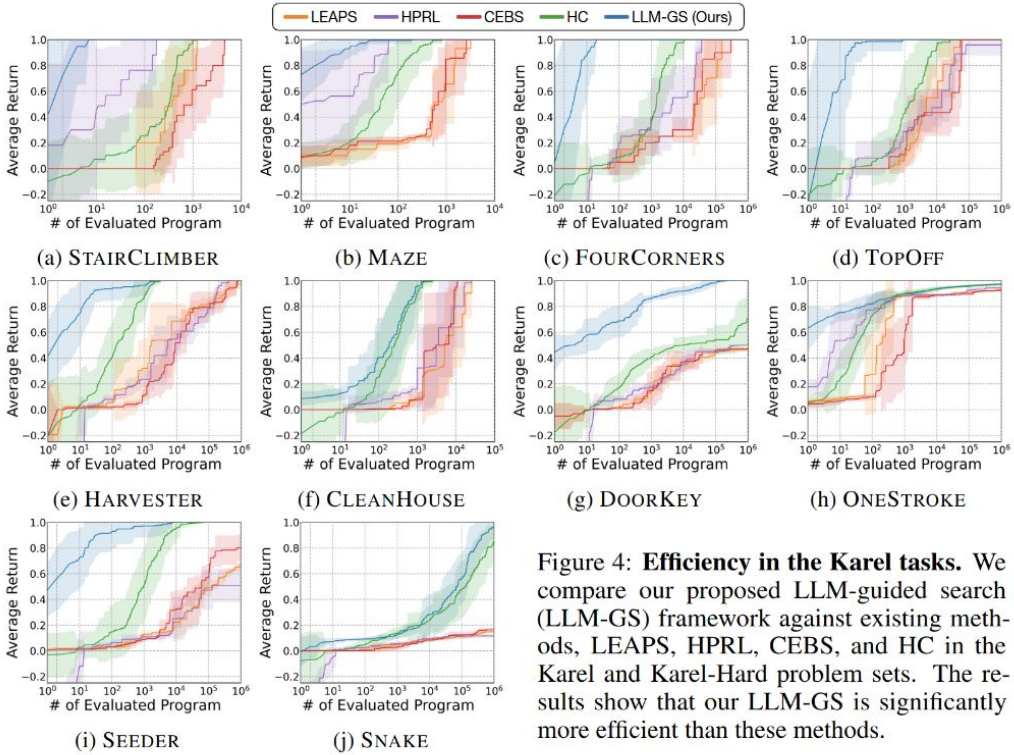
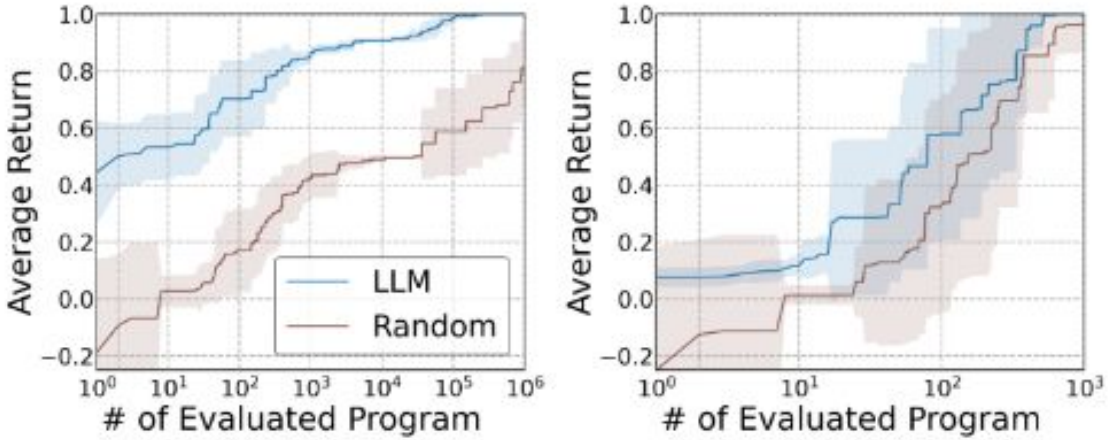


Figure 4: **Efficiency in the Karel tasks.** We compare our proposed LLM-guided search (LLM-GS) framework against existing methods, LEAPS, HPRL, CEBS, and HC in the Karel and Karel-Hard problem sets. The results show that our LLM-GS is significantly more efficient than these methods.

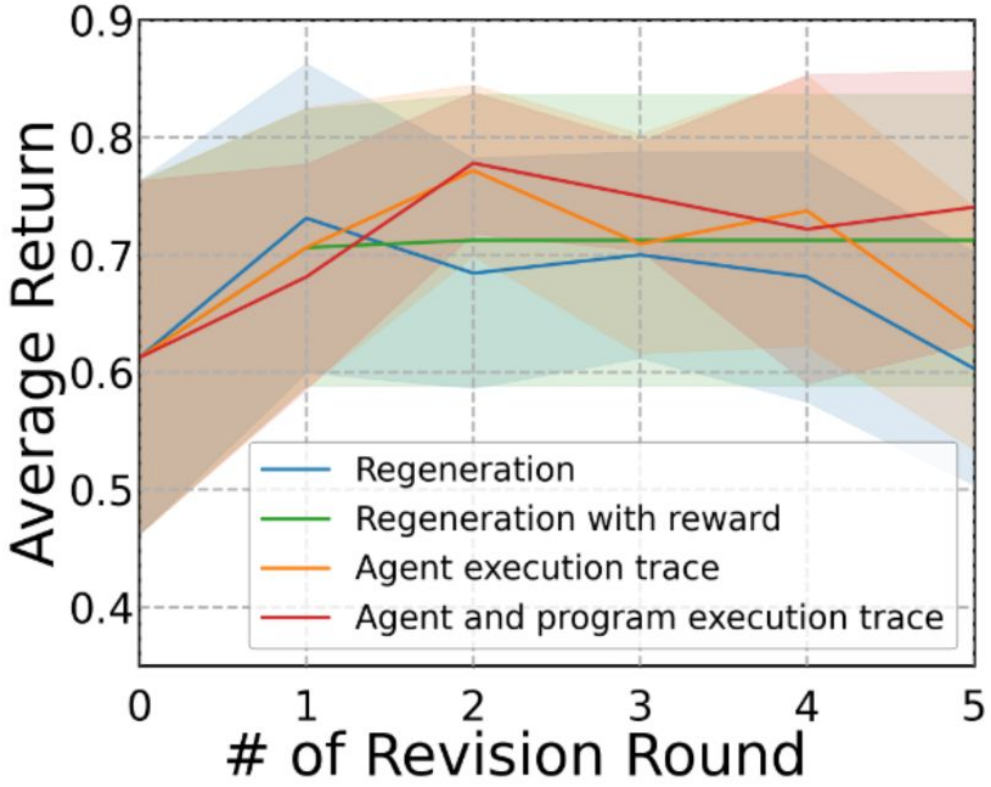
But is it LLM? Maybe it doesn't matter…



(a) DoorKey (b) CleanHouse

Why LLMs can gain so much advantage?

Ok, but maybe LLM alone would do it on it's own



Conclusions

- LLMs and LLM-agents have significantly advanced the field of code generation
- Program Synthesis has a cool application in Reinforcement Learning
- LLMs absorb “human culture” and that can be useful in RL and in programmatic RL as well

References

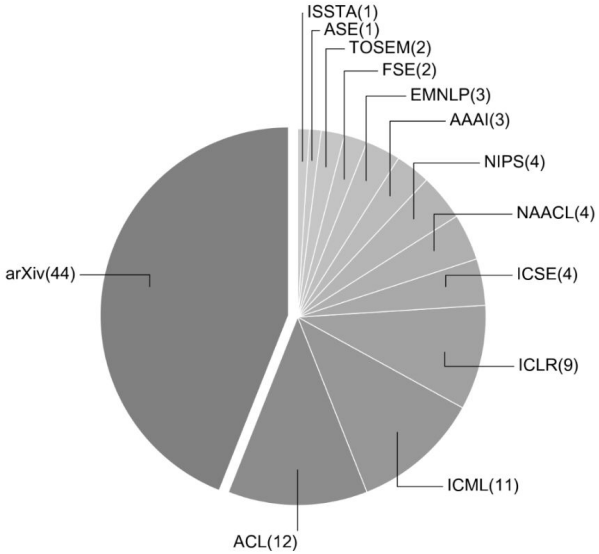
- “A Survey on Code Generation with LLM-based Agents”, 2025, Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, Ge Li, https://arxiv.org/html/2508.00083v1
- “Learning to Synthesize Programs as Interpretable and Generalizable Policies”, 2022, Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, Joseph Lim, https://arxiv.org/abs/2108.13643
- “Synthesizing Programmatic Reinforcement Learning Policies with Large Language Model Guided Search”, 2024, Max Liu, Chan-Hung Yu, Wei-Hsu Lee, Cheng-Wei Hung, Yen-Chun Chen, Shao-Hua Sun, https://arxiv.org/abs/2405.16450
- “From Provable Correctness to Probabilistic Generation: A Comparative Review of Program Synthesis Paradigms”, 2025, Zurabi Kobaladze, Anna Arnania, Tamar Sanikidze, https://arxiv.org/abs/2508.00013

A comparative analysis of program synthesis paradigms

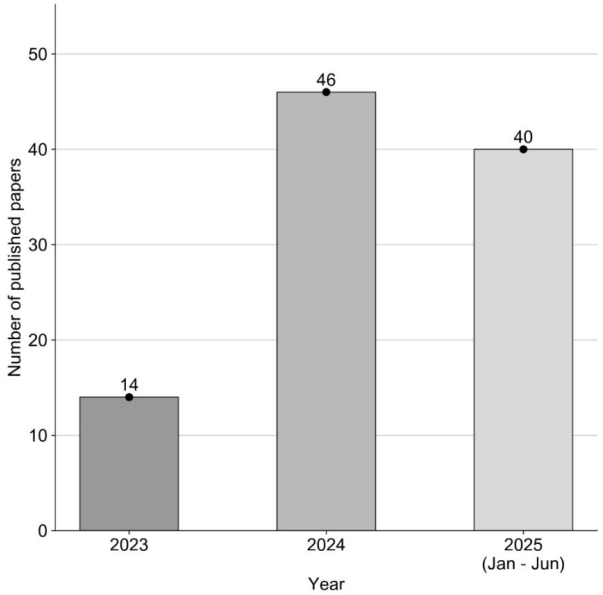
Feature	Deductive Synthesis	Inductive Synthesis (PBE)	Sketch-Based Synthesis	Neuro-Symbolic Synthesis
Primary Specification	Formal Logical Formula (Pre/Post-conditions)	Input-Output Examples	Partial Program + Assertions/Tests	Natural Language, Examples, Demonstrations
Form of Guidance	Proof Steps / Tactics	Additional Examples / User Feedback	Program Structure (Holes, Generators)	Learned Heuristics, Learned Sketches, Neural Priors
Search Strategy	Theorem Proving, Term Rewriting	Enumerative Search, Version Space Algebra	CEGIS, Constraint Solving (SAT/SMT)	Guided Search (Neural) + Symbolic Search
Correctness Guarantee	Correct-by-Construction	Correct on Examples (may not generalize)	Verified w.r.t. Spec & Bounds	Probabilistic, often requires symbolic verifier
Key Tools	Coq, Isabelle/HOL	FlashFill, PROSE	Sketch, Rosette	DeepCoder, [Zhang et al., 2023]
Primary Challenge	Specification Effort, Scalability [Manna and Waldinger, 1980b]	Specification Ambiguity, Generalization	Scalability, Sketch Design Brittleness, Opacity [Singh and Solar-Lezama, 2018]	Data Requirements, Interpretability, Combining Logics

source: From Provable Correctness to Probabilistic Generation: A Comparative Review of Program Synthesis Paradigms, 2025, Arnania, Kobaladze, Sanikize

LLM-based Agents Publishing Statistics



(a) Distribution of papers published by year



(b) Distribution of papers published by conference or journal

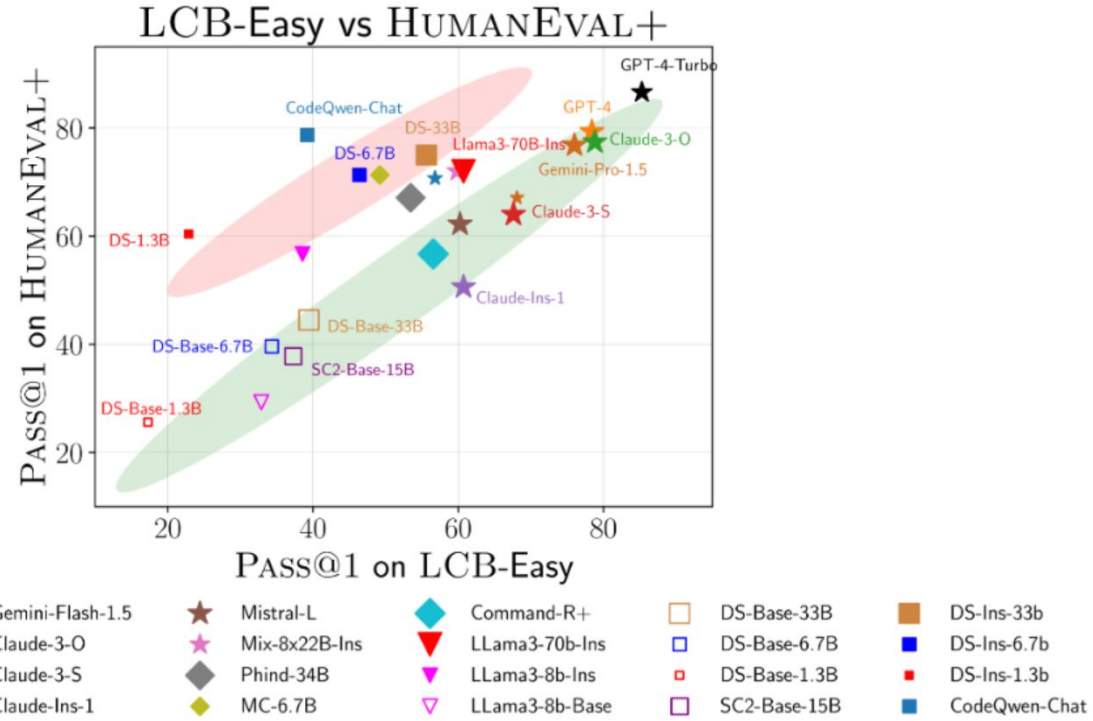
Fig. 1: Distribution of papers related to LLM-based code generation agents.

Deployed Code Generation Agent Tools

TABLE 1: Horizontal Comparison of Code Generation Agent Tools

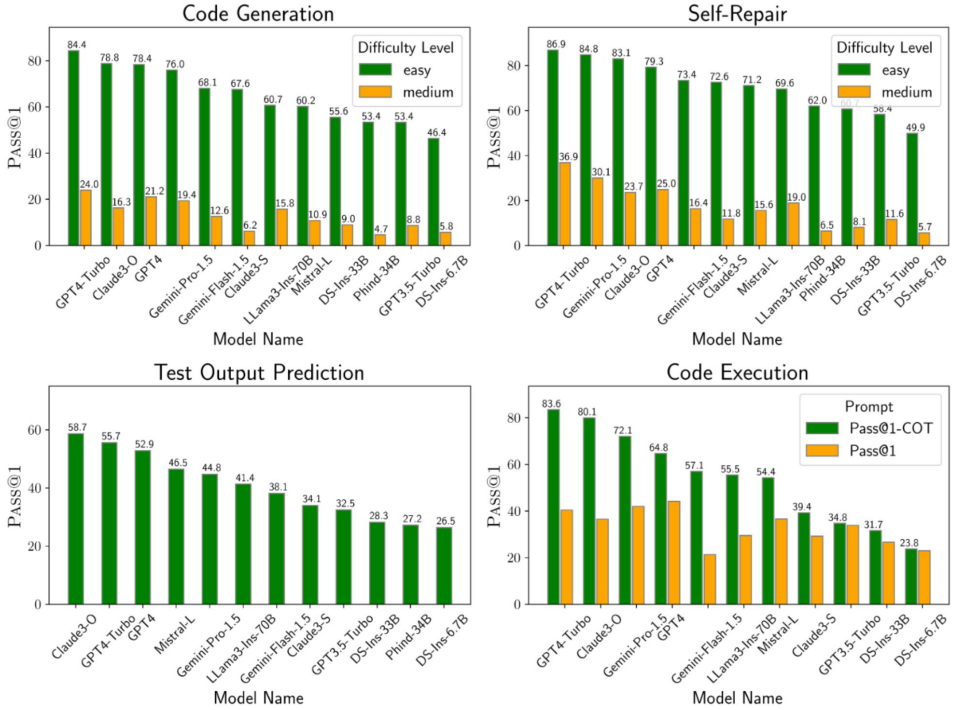
Tool Name	Core Paradigm	Primary Integration Method	Context Engine	Best Application Scenario	Main Advantages	Known Limitations
GitHub Copilot	Programming Assistant	IDE Extension Plugin	Retrieval Augmented	Code Completion	Wide adoption, excels at routine tasks	Requires human supervision, may overlook global context
Devin	Autonomous Development Team	Independent Online Platform	Sandbox-based Knowledge Base	End-to-End Code Generation	High autonomy, handles full task lifecycle	Low practical reliability, unpredictable results, prone to loops
Cursor	Collaborative Partner	AI-Native IDE	Vector Semantic Indexing	Code Completion, Code Development, Codebase Q&A	Deep codebase understanding, excellent interaction experience	Weaker autonomous agent capabilities
Tongyi Lingma	Collaborative Partner	Enterprise Platform Integration	Repository Knowledge Graph	Code Development	Robust and systematic exploration of solution space	More focused on enterprise-level applications
Claude Code	Autonomous Development Team	Command Line Interface	Ultra-Long Context Window + Agent Search	End-to-End Code Generation, Codebase Q&A	Powerful code agent, global codebase understanding	Potentially high cost

LiveCodeBench (UC Berkeley,MIT,Cornell University)



source: https://livecodebench.github.io/

LiveCodeBench (UC Berkeley,MIT,Cornell University)



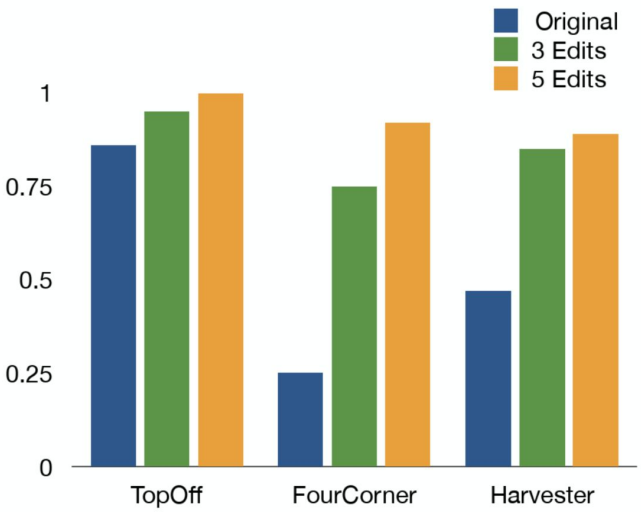
source: https://livecodebench.github.io/

Interpretability Experiment

- Trust
- Contestability
- Safety

Human Debugging Interface

Performance Improvement



source: https://clvrai.github.io/leaps/