Current State of LLM/SLM-based Code Generation and Future Directions

Srinath Perera
Head of Research, WSO2

Speaker

- Srinath is a scientist, software architect, and programmer who works on distributed systems and AI.
  - Apache Member, 20+ years on open source projects
  - Head of research at WSO2
  - WSO2
    - Well known for API management, integration, and identity platform products
    - 700+ enterprise customers, including banks, governments, airlines, etc.
- He has co-architected a dozen real-world distributed systems (e.g., Apache Axis2, WSO2 SIddhi, key role in many WSO2 products)
- Author of the book Software Architecture and Decision-Making (Pearson)
- 1500+ citations (Scholar Profile)
- Ph.D. from Indiana University, USA, in 2009

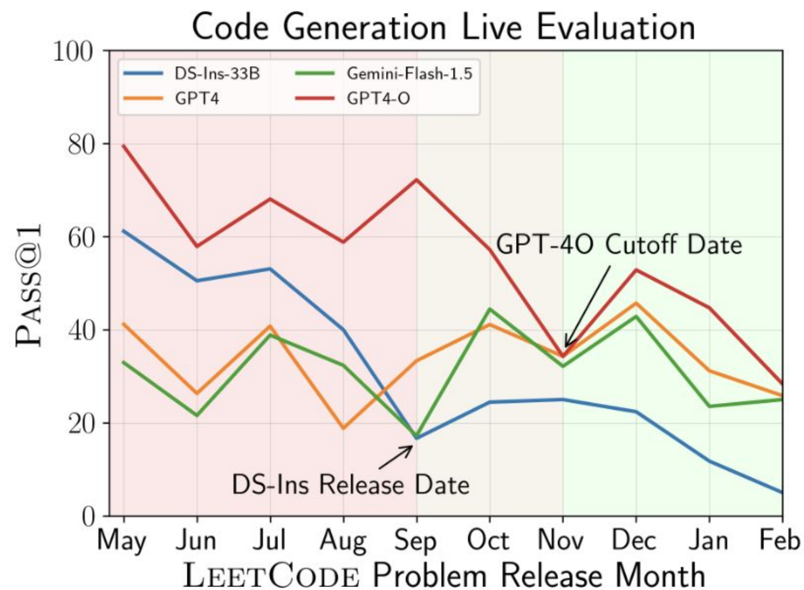
Is Code Generation a Solved problem?

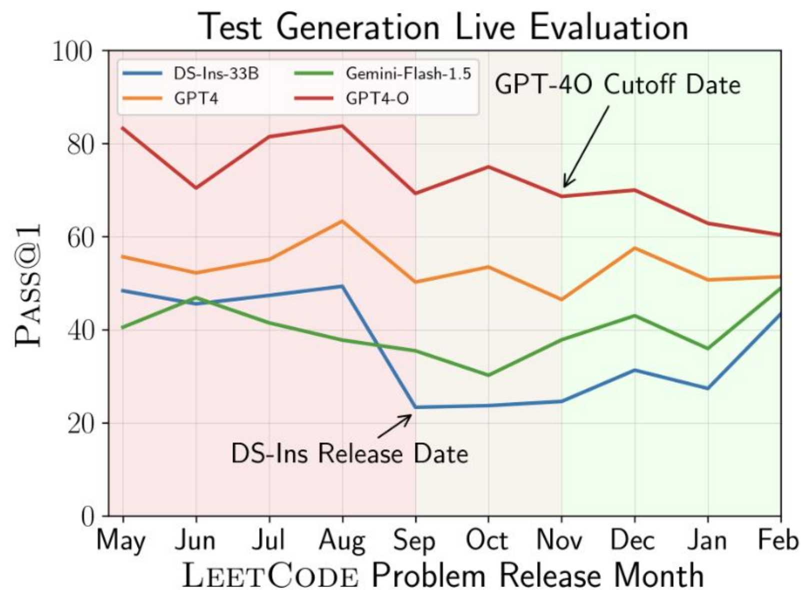
- While at first glance, Code Generation looks like a solved problem:
  - HumanEval at 94%+
  - SWE-bench Verified at 76%+
  - Claude Code writes full apps in minutes
  - Many success stories
    - including olympiad-level performance on competitive programming problems.

A Closer Look:

However, a careful look shows problems:

- LiveCode Bench showed clear signs of contamination.





A Closer Look:(contd.)

However, a careful look shows problems:

- Kevanoski prize Kaggle winners only report 5% accuracy, where they only had a few problems they were sure about.
- Furthermore, benchmark representatives have been challenged.
  - e.g., a study has shown that accuracy drops from 80+% to 20+% when faced with real-world problems, etc. METR papers argued that using AI tools added 20% slowdown experience developers.
- AI models sometimes can get very easy (for humans) problems wrong while getting Olympiad-level problems right. Most AI models exhibit weak monotonicity and are thus much more unpredictable than humans.

Code Generation Needs High Accuracy

- Writing code for the real world is less about the code itself and more about who will fix it if there is a bug. With code-generation tools, the answer is not clear. Organizations using AI more with code have to answer this question.
- Correctness of code is critical. I tell my students that if a doctor makes a mistake, one person can die. But programmers make mistakes, 10,000 people can be affected, and we might not even know.
- The ability to reject wrong answers is critical.
  - So testing plays a key role, too.

Taxonomy of Sub Problems

- Type of Coding problem
  - Competitive, Single Function Level Problems (HumanEval, LiveCode Bench)
  - Program Repair (SWE-bench-*)
    - APR/ Fixes, Porting, Patches
  - Program Translation
- Language Popularity
  - High Resource Programming Language
  - Low Resource Programming Language
- When techniques applied
  - Inference/ Test time scaling
  - Training time
    - Foundation models
    - Fine tuning
      - SLMs

How to measure success?

- Based on Unit test success
  - Limited as it might accept wrong answers
  - However, we do not have a better answer
  - Pass@X (Chen, M., et al. (2021). "Evaluating Large Language Models Trained on Code." arXiv:2107.03374)
- CodeBLEU score (Ren, Shuo, et al. "Codebleu: a method for automatic evaluation of code synthesis." arXiv preprint arXiv:2009.10297 (2020).)
- We need to consider letting the model abstain and then measure precision, recall, F1
- Other
  - AST Similarity, Cyclomatic Complexity, CWE (Common Weakness Enumeration), Acceptance Rate (% changes developers actually keep), Edit Distance: How many keystrokes a human had to make to "fix" the AI’s suggestion

Broad Approaches

- RLVR phase in foundation model creation adds significant coding skills
- Distillation based on high-quality datasets can significantly improve SLM coding performance.
- Prompting methods
- Fine Tuning
  - SFT
  - GRPO
- Self Reflection
  - LLM feedback (e.g., Tong, Weixi, and Tianyi Zhang. "Codejudge: Evaluating code generation with large language models." arXiv preprint arXiv:2410.02184 (2024)
  - Simulations with LLM (e.g., CodeSim)
- Feedback
  - Using execution feedback to iteratively improve code
  - Testing and verification are critical. Code and test are connected. If you can generate good tests, the feedback can fix the code (AgentCoder).
  - Can we generate formal code that supports correctness proofs?
- The last two methods fall under "Inference/ Test time scaling."

Case 1. Competitive, Single Function Level Problems

- Benchmarks
  - HumanEval, LiveCode Bench
- Solutions
  - Fine Tunning (SFT, GRPO)
  - Agent Coder
  - CodeSim
  - BACE
  - Agreement
    - Code T

Approaches: RL

- Language Models have significant fine-tuning for code via SFT or GRPO, which increases inference and instruction-following skills.
- The high-level idea is to teach LLM how to do better code and better fixes with feedback.
  - PPOCoder - Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. 2023. Execution-based code generation using deep reinforcement learning. arXiv preprint arXiv:2301.13816 (2023)
  - CodeRL- Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. Advances in Neural Information Processing Systems 35 (2022), 21314–21328.
- There is room for unsupervised or semi- supervied learning (e.g., Phi-3)
  - using a break-fix cycle to improve the model, Phi-3 Safety Post-Training: Aligning Language Models with a "Break-Fix" Cycle, https://arxiv.org/abs/2407.13833 (2024)

Approaches: RL (Contd.)

- Language Models have significant fine-tuning for code via SFT or GRPO, which increases inference and instruction-following skills.
- The high-level idea is to teach LLM how to do better code and better fixes with feedback.
  - PPOCoder - Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K Reddy. 2023. Execution-based code generation using deep reinforcement learning. arXiv preprint arXiv:2301.13816 (2023)
  - CodeRL- Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. Advances in Neural Information Processing Systems 35 (2022), 21314–21328.
- There is room for unsupervised or semi- supervied learning (e.g., Phi-3)
  - using a break-fix cycle to improve the model, Phi-3 Safety Post-Training: Aligning Language Models with a "Break-Fix" Cycle, https://arxiv.org/abs/2407.13833 (2024)
- After 23-24, agent-based approaches took over.
- RL is still used in
  - SLM - E.g. Cho, Jeonghun, et al. "Self-Correcting Code Generation Using Small Language Models." arXiv preprint arXiv:2505.23060 (2025)
  - LRPL
  - Personalization to a repo, organization

Approaches: Agents

- Agent-based solutions overtook fine-tuning/RL-based solutions around 2023-24
- The main idea is having several specialized agents (often playing roles similar to humans -e.g., coder, reviewer, test.
  - Generate candidates, generate tests, run them, and use feedback to improve solutions.
  - Language models are trained to fix the code given the code + feedback.
- AgentCoder (2023) is a key milestone, reaching 90+% pass@1of at HumanEval.
  - Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. 2023. AgentCoder: Multi-Agent-based Code Generation with Iterative Testing and Optimisation. arXiv preprint arXiv:2312.13010 (2023).
  - Still very competitive, and simple (they reached 96% at 2024 update)
- The quality of tests is critical.
  - Bad tests will derail the results

Approaches: Agents (contd.)

- Several other variations has beein proposed
  - INTERVENOR (teacher, student) - Wang, Hanbin, et al. "Intervenor: Prompting the coding ability of large language models with the interactive chain of repair." Findings of the Association for Computational Linguistics: ACL 2024. 2024.[
  - Chen, Jizheng, et al. "DebateCoder: Towards Collective Intelligence of LLMs via Test Case Driven LLM Debate for Code Generation." Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2025.
  - 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).
- Agent approaches highly dependent on instruction following performance
  - Hence their performance on SLMs can be limited - E.g, Agent Coder LCB performance reduced from 52% to 12.5% (GPT5-mini -> Quwan7B)

Approaches: Simulations feedback in Agents

- Next breakthrough came from simulations - asking LLM to simulate (think) execution and fix.
  - When LLM can successfully simulate, this improves results; if it can't, it tends to hallucinate
- Can fail in SLMs
- References
  - Islam, Md Ashraful, Mohammed Eunus Ali, and Md Rizwan Parvez. "Codesim: Multi-agent code generation and problem solving through simulation-driven planning and debugging." arXiv preprint arXiv:2502.05664 (2025).

Approach	HumanEval	MBPP
Direct	90.20%	81.10%
CoT	90.90%	82.90%
Self-Planning	89.00%	82.60%
Analogical	88.40%	75.10%
Reflexion	87.20%	81.10%
LATS	88.80%	-
MapCoder	90.20%	88.70%
CodeSim	**95.10%**	**90.70%**

Results from Codesim paper

Approaches: Using Better Test cases

- Quality of test cases are critical
  - With comprehensive public test cases, AgentCoder will solve the problem
- Any new improvement in automatically generating test cases can help
  - Property tests
    - E.g. sort - has all inputs, no other data, in sorted order
  - Diffeerntial tests - find inputs where two solutions will be different
  - Finding inputs that will force edge cases
  - This is open area

Approaches: Inference Time/ Test time Scaling

- Assumption: that it is harder for wrong versions to agree on the same input than the right versions to agree on the same input.
- Generate many solution candidates and select the best one.
  - Chen, Bei, et al. "Codet: Code generation with generated tests." arXiv preprint arXiv:2207.10397 (2022).
    - Generate many tests, run multiple solutions, and look for agreement.
  - Li, Dacheng, et al. "S*: Test time scaling for code generation." arXiv preprint arXiv:2502.14382 1.2 (2025).
    - Cluster by output from tests, do pair-wise comparisons by using a differential test to generate differences, and decide based on the results.
    - 83% on LCB V2

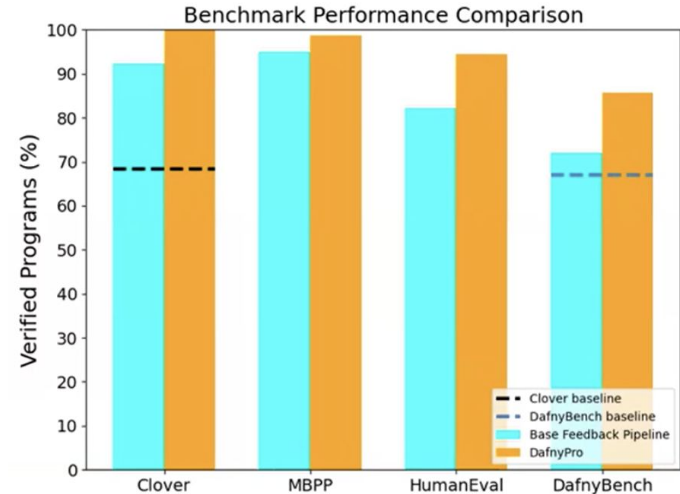
Approaches: Coevolution

- Generate code population and test population
- Run them and use feedback to rank them
- Run co-evolution by evolving both code and test populations by mapping operators to LLM-based operators
- Can combine many candidate generation techniques and test generation techniques
    - E.g., Kaushitha Silva, Srinath Perera, BACE: LLM-based Code Generation through Bayesian Anchored Co-Evolution of Code and Test Populations, https://arxiv.org/abs/2603.28653v1

	GPT-5-Mini				Qwen2.5-Coder:7b			
Method	Easy	Med	Hard	Overall	Easy	Med	Hard	Overall
Direct	83.3	60.0	27.0	50.0	27.8	4.0	0.0	7.5
AgentCoder	88.9	52.0	35.1	52.5	38.9	8.0	2.7	12.5
MapCoder	83.3	80.0	35.1	60.0	–	–	–	–
CodeSIM	**94.4**	80.0	45.9	67.5	**72.2**	16.0	8.1	25.0
BACE (Ours)	**94.4**	**84.0**	**51.4**	**71.3**	**72.2**	**28.0**	**10.8**	**30.0**

Approaches: Formal Approches

- Idea is to generate code in formally verifiable programming language (e.g., Dafny - https://dafny.org/)
  - Code will include pre, post conidtions and inverients as part of compilations
- Require interactive code refinement until code complies
- Writing such code is hard, but LLM does not get to complain
  - Solved with Multi-turn inference time scaling
  - Banerjee, Debangshu, Olivier Bouissou, and Stefan Zetzsche. "DafnyPro: LLM-Assisted Automated Verification for Dafny Programs." arXiv preprint arXiv:2601.05385 (2026).
  - Chuyue Sun, Ying Sheng, Oded Padon, and Clark Barrett. 2024. Clover: Closed-Loop Verifiable Code Generation. In AI Verification: First International
  - Symposium, SAIV 2024, Montreal, QC, Canada, July 22–23, 2024, Proceedings (Montreal, QC, Canada). Springer-Verlag, Berlin, Heidelberg, 134–155.
  - doi:10.1007/978-3-031-65112-0_7
- Will allow us to prove correctness of the programs, and one day we might be able compose code using formally proved libraries



(b) DafnyPro performance with Claude 3.7 Sonnet across benchmarks.

** the figure from DafnyPro paper

Program Repair (Fixing Bugs)

- Given a Repo, Bug description and (optional test case - not shown to the system), create a patch that fix the problem
- Benchmarks
    - SWE-bench
    - SWE-bench live
- Approches
    - Static pipeline
    - Planning based (Single Agent)
    - Multiple Agents (with agents having specialization)

Program Repair (Contd.)

- Sub problems
  - Bug localization
  - Generating a validating test case
    - Cheng, Runxiang, et al. "Agentic bug reproduction for effective automated program repair at google." arXiv preprint arXiv:2502.01821 (2025).
  - Patch Generation
    - This has overlap with iteratively improving solutions with competition code generation

SWE-Live leaderboard

Rank	Method / Tool	Resolved	Date	Key Paper Describing Tool
🥇 1	**SWE-agent** + Claude-4.5-Sonnet	36.00%	11/30/2025	*SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering* (Yang et al., 2024)
🥈 2	**OpenHands** + Qwen3-Coder-480B-A35B	24.67%	07/24/2025	*OpenHands: An Open Platform for AI Software Developers as Generalist Agents* (ICLR 2025)
🥉 3	**SWE-agent** + GPT5-Thinking (Medium)	24.33%	10/31/2025	*SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering* (Yang et al., 2024)
4	**SWE-agent** + Claude 3.7 Sonnet	17.67%	04/30/2025	*SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering* (Yang et al., 2024)
5	**OpenHands** + Claude 3.7 Sonnet	17.67%	04/30/2025	*OpenHands: An Open Platform for AI Software Developers as Generalist Agents* (ICLR 2025)

Contd.

- From: Lingxi: Repository-Level Issue Resolution Framework Enhanced by Procedural Knowledge Guided Scaling.
-

Approach	Base Model	Pass@1(%)
mini-SWE-agent	GPT-4.1 mini	23.94
Lingxi	GPT-4.1 mini	**45.20**
mini-SWE-agent	Kimi-K2	43.80
OpenHands	Kimi-K2	65.40
Lingxi	Kimi-K2	**70.33**
mini-SWE-agent	Claude 4 Sonnet	64.93
SWE-agent	Claude 4 Sonnet	66.60
SWE-search	Claude 4 Sonnet	70.08
OpenHands	Claude 4 Sonnet	70.40
Augment Agent v1	Claude 4 Sonnet	70.40
Lingxi	Claude 4 Sonnet	**74.60**

Solutions: SWE Agent

- SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering (2024) (1082+)
- #1, #3, #4, #6 of SWE bench live leaderboard
  - (https://swe-bench-live.github.io visited April 03, 2026)
- Use ReAct to turn LLM into an interactive debugger based on feedback.
- An Agent computer interface (ACI)
  - a small set of simple actions for viewing, searching through, and editing files.
    - Search and navigation
    - File viewer.
    - File edit
    - Context management
- Guardrails (e.g., compiler errors)
- Demonstrate how careful ACI design can substantially improve LM agent performance
- Provides concise feedback about a command’s effects at every turn

Solutions: OpenHands

- https://openhands.dev/ & Wang, Xingyao, et al. "Openhands: An open platform for AI software developers as generalist agents." arXiv preprint arXiv:2407.16741 (2024). [500+]
- #2, #5, #9 of SWE bench live leaderboard
  - (https://swe-bench-live.github.io/ visited April 03, 2026)
- Compared to SWE-Agent, OpenHands is
  - Multi-agent (SWE-Agent is a single ReAct agent)
  - Provide a co-worker-like experience with browser access, running commands (docker sandbox, etc.)
  - includes an events stream, a chronological collection of past actions and observations, including the agent's own actions and user interactions (e.g., instructions, feedback).

Solutions: Agentless

- Xia, Chunqiu Steven, et al. "Agentless: Demystifying llm-based software engineering agents." arXiv preprint arXiv:2407.01489 (2024). (408)
- #8 of SWE bench live leaderboard
    - (https://swe-bench-live.github.io/ visited April 03, 2026)
- How
    - AGENTLESS employs a simplistic three-phase process
    - Localization - file (prompt LLM with repostrctures in the context + embedding search) -> class/function (query LLM with compressed format of the selected files) -> lines (give code to LLM and ask)
    - Repair - diff format - generate reproduction test and use majority voting and regression testing.
    - Patch validation.
- LLM can't use too many actions or operate with complex tools.
- https://github.com/OpenAutoCoder/Agentless

Bug Localization

- Simple search and filtering (embeddings or IR)
- Multi-level (e.g., Repository → File → Function → Statement).
  - can use emebdding/ IR
  - E.g., AutoCodeRover (AST), BugCerberus (three fine-tuned LLMs), Agentless
- Newer methods
  - Depth-first search on the code dependency graph
  - Causal reasoning/ call graph-based RCA
  - Neuro-Symbolic & Semi-Formal Reasoning
  - Embedding built from control and data flow

Low Resource Languages

- Benchmarks
  - MultiPL-E - adopted Human Eval
  - Ag-LCB - adopted Livecode Bench
- Main idea is to SFT or GRPO on coding mode
  - Key challenge is lack of training data

Method	Base Model	MultiPL-E	Ag-LCB
Giagnorio et al. [9]	DeepSeek Coder	41.2	-
	CodeLlama-7B	28.4	-
Agnostics [1]	Qwen3-4B-MBPP	62	15
	Qwen3-8B-CF	61	25

- GRPO
  - Given compile feedback give some improvement
  - Generating code + test, running them, and giving feedback takes the number further, but eventuall model learn to cheat
  - Agnostics [2026] proposed to use outputs from Python answers to the same problem as ground truth data
  - Selecting problems with right hardness maximize the learning

Adjacent Problems

- Porting Patches
- Porting Dependencies
- Language Translation
- Performance/ Cost/ Energy efficiency of generated code
- Security, vulnerability of the generated code
- Explainability, human understandability
- Repo specialization, personalization (e.g., by keeping separate LoRA)
- Reduce recreating the wheel (hit the right level of reuse of the code)
- ..

Where to find Datasets?

- Benchmarks
- Compititve Programming Platforms has good data sets - problem, multiple verified answers, sometimes in several languages
    - Some problems we can create an IO ground truth by running Python answers
- Opensource code
    - Bug, patch pairs
        - Fixing issues
        - Fixing vinurbilities
        - Backporting patching
        - Porting to new dependency versions

Other

- Using execution feedback to iteratively improve code
- Testing and verification are critical. Code and test are connected. If you can generate good tests, the feedback can fix the code.
  - Still they do not guarantee completeness - all bugs are addressed
- Can we generate formal code that supports correct proofs?
- What part of the problem can be handed over to more deterministic models? e.g., finding which part of the code to change, high-level architecture?
- Can we generate explainable code and use human feedback to improve interactively?

Takeaways

- Correctness of Generated code is critical
- Contamination is real
  - Report results against Live Benchmarks
- Tests are critical, if there are good tests they can drive the correctness
  - Tests that do not handle edge cases can hurt
- Program simulation with LLMs (a.k.a. Is underrated)
- Bug Localization is a challenging and open problem.

Who we are?

- WSO2
  - Owned by EQT (https://en.wikipedia.org/wiki/EQT_AB)
  - Well known for API management, integration, and identity platform products
  - 700+ enterprise customers, including banks, governments, airlines, etc.
  - Comes from an open source background - All our products are open source
- Research Topics
  - Code Generation
  - AI-related middleware
    - Agent Observability
    - Anomaly detection (e.g., kill switch)
    - Root cause analysis
    - Learning to Defer