ACL 2026 SAN DIEGO JULY 2–7

Oral Presentation

CodeEvo

Interaction-Driven Synthesis of Code-centric Data through Hybrid and Iterative Feedback

Qiushi Sun Jinyang Gong Lei Li Qipeng Guo Fei Yuan

The University of Hong Kong · Shanghai AI Laboratory · NYU
Carnegie Mellon University · Shanghai Innovation Institute

github.com/QiushiSun/CodeEvo

MOTIVATION

Code LLMs are only as good as their training data

Training strong code models needs instruction–code pairs that are:

- **Complex**
- **Diverse**
- **Grounded & executable**

But manually curated data is **expensive, hard to scale,** and gradually exhausted.

→ **Synthetic generation is the way forward.**

The data bottleneck

Human curation

ideal — but doesn't scale

↓

Symbolic augmentation

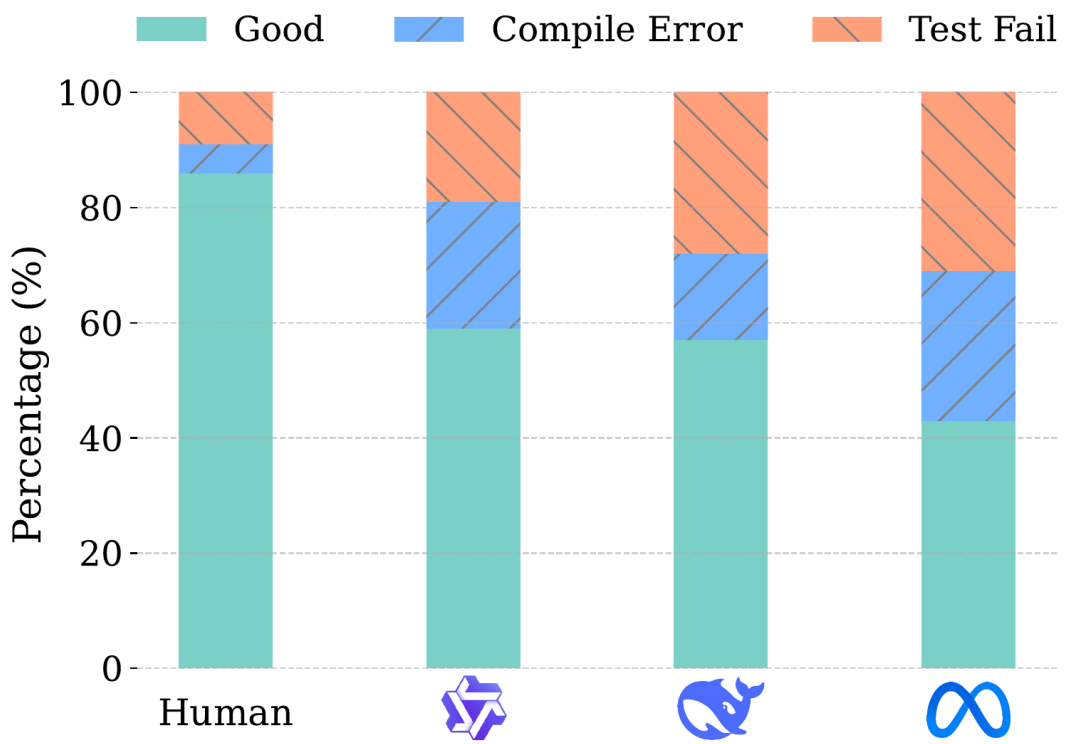
limited, reference-bound

↓

LLM self-synthesis

scalable — but quality?

Today's synthetic code data is largely broken



Quality of Evol-Instruct data across models — much fails to compile or pass tests.

Heuristic methods (Self-/Evol-/OSS-Instruct)

rely on rigid prompts — two failure modes:

① Ungrounded instructions

"make it harder" → vague, inconsistent objectives detached from real code.

② Unvalidated code

no correctness check during synthesis → compile errors & test failures.

Can we build a fully automated, reference-free pipeline that yields well-grounded and ***executable instruction–code pairs?***

POSITIONING

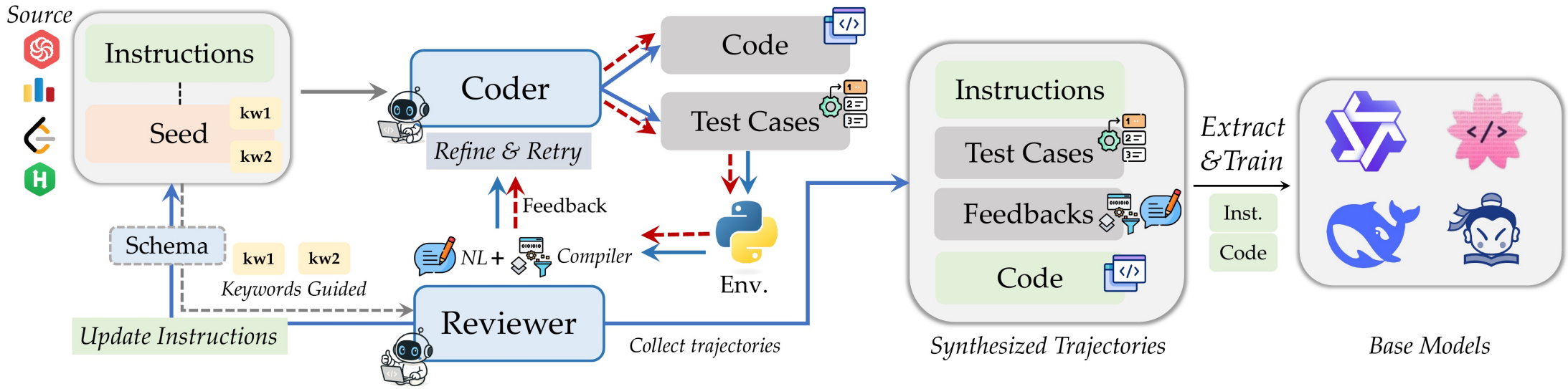
How CodeEvo differs from prior synthesis

Prior methods each miss something — rigid heuristics, required code references, or no correctness check.

Method	Reference-free	Grounded instructions	Validated code	Difficulty control
Self-Instruct	✓	✗	✗	✗
Evol-Instruct	✓	✗	✗	~
OSS-Instruct	✗	✓	✗	✗
CodeEvo (ours)	✓	✓	✓	✓

CodeEvo is the only fully automated pipeline that is reference-free, grounded, execution-validated, and **difficulty-aware.**

CodeEvo: two agents that synthesize data by interacting



(a) Interaction-Driven Synthesis

(b) Downstream Application

Coder

Generates candidate solutions + test cases; refines them on feedback.

Reviewer

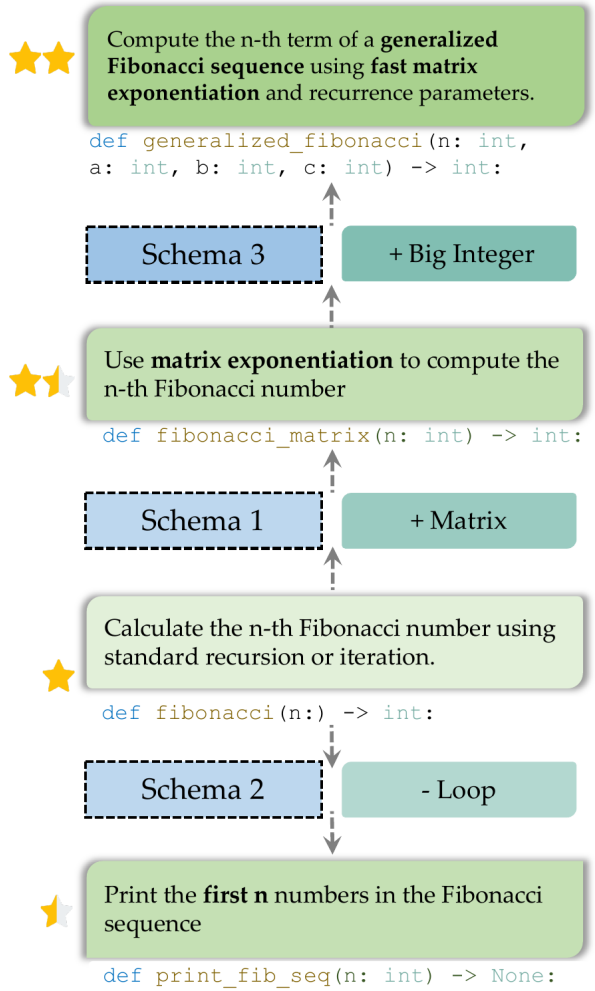
Plans new instructions (Schema) + judges solutions, providing feedback.

Schema-guided, keyword-driven generation

Instead of abstract "evolve" commands, the Reviewer first writes a **Schema** — a structured blueprint of logic, complexity, and goal.

Keywords → **Schema** → **Instr. + Code**

- **Co-evolution:** instructions & code are designed together → intrinsically grounded.
- **Stepped difficulty:** add keywords to harden, or drop them to simplify.
- **Bidirectional:** if a task is too hard, plan an easier variant — no dead ends.



Seed → progressively harder, grounded tasks

Hybrid feedback: compiler meets semantic judge

Compiler

Deterministic pass/fail by executing tests.

But self-made tests miss cases → false positives.

Semantic Judge

Reviewer audits logic, constraints, edge cases.

Catches what tests can't.

+

Hybrid Feedback

f_hybrid

rigor of execution
+ flexibility of an LLM

→

Data selection

keep only pairs that clear a strict quality bar.

Iterative refinement

rich signal guides the Coder to fix subtle bugs.

Neither signal alone is enough — **compiler determinism + semantic nuance** together ensure correctness AND logical integrity, with no human labels.

PUTTING IT TOGETHER

An adaptive, self-correcting synthesis loop

1 Propose	Reviewer plans a Schema → new instruction
2 Attempt	Coder writes solution + tests
3 Verify	Hybrid feedback: compiler + semantic judge
4a Evolve ✓	valid → keep pair, harden for next round
4b Simplify ✗	too hard → plan an easier variant, backtrack

Why this matters

Intrinsic quality control

self-correction keeps yield high & grounded

Dynamic difficulty

never stuck — adapts to agent capability

Full trajectories

captures the whole solve–feedback–fix cycle

Interaction-driven synthesis (single seed)

Input: *seed s, keyword set T, max iters N*

Output: *validated instruction–code pairs Q*

```
Q ← ∅,   q ← s,   k ← 0
while k < N:
    (c, g, f_comp) ← Coder(q)
    f_NL ← Reviewer(q, c, g, f_comp)
    f_hybrid ← Reviewer(f_comp, f_NL)
    if f_hybrid is valid:          # success
        add (q, c) to Q
        Schema ← Reviewer.Plan(q, sample(T))
        q ← Reviewer.Write(q, Schema);   k += 1
    else:                          # too hard
        Schema⁻ ← Reviewer.Plan(q, sample(T))
        q⁻ ← Reviewer.Write(q, Schema⁻)   # simpler
        re-verify q⁻; add if valid;   break
return Q
```

Two branches

Success → Evolve

keep the pair, then plan a harder Schema for the next round ($q \rightarrow q^{+}$).

Failure → Simplify

plan a simpler Schema ($q \rightarrow q^{-}$), re-verify once, then end the trajectory.

Bidirectional difficulty = no dead ends, high yield.

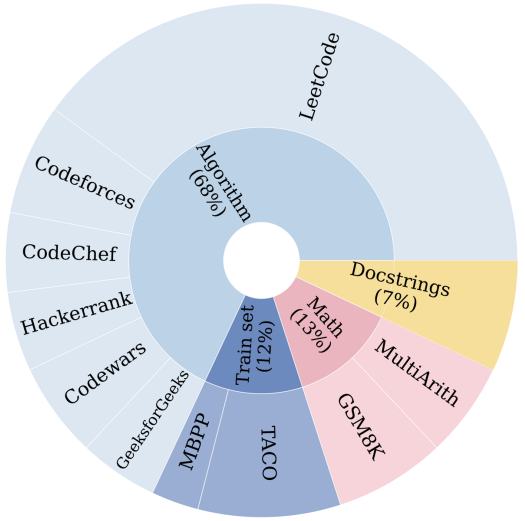
CodeEvo-100K: stepped difficulty + full trajectories

Difficulty tiers by evolution depth

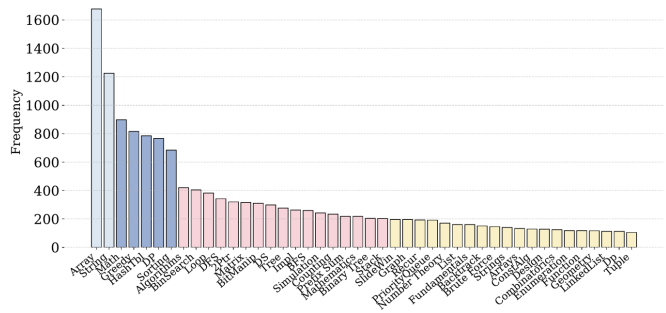
Easy — seed + 1-round evolutions

Medium — 2-round evolutions

Hard — survives ≥ 3 evolutionary cycles



Seed types & sources (~5K)



Keyword distribution (~3 tags / seed)

Preserves the complete evolutionary trajectory
— intermediate reasoning + refinement, not just final pairs.

~5K	100K	3
seed instructions	synthesized pairs	difficulty tiers

MAIN RESULTS

Beats baselines — and wins on data efficiency

17K CodeEvo samples > 75K OSS-Instruct samples

A better synthesis algorithm beats sheer data volume — at 4–5× less data.

Method	Data	HE+	MBPP+	BCB-Hard	LiveCodeBench
Qwen3-8B (base)	–	77.4	70.9	22.3	39.1
OSS-Instruct	75K	78.5	67.5	24.1	36.3
Evol-Instruct	25K	78.2	72.5	25.2	40.9
CodeEvo (hard)	**17K**	**79.8**	**74.1**	**24.1**	**42.7**
CodeEvo (full)	**100K**	**81.7**	**73.8**	**26.1**	**46.9**

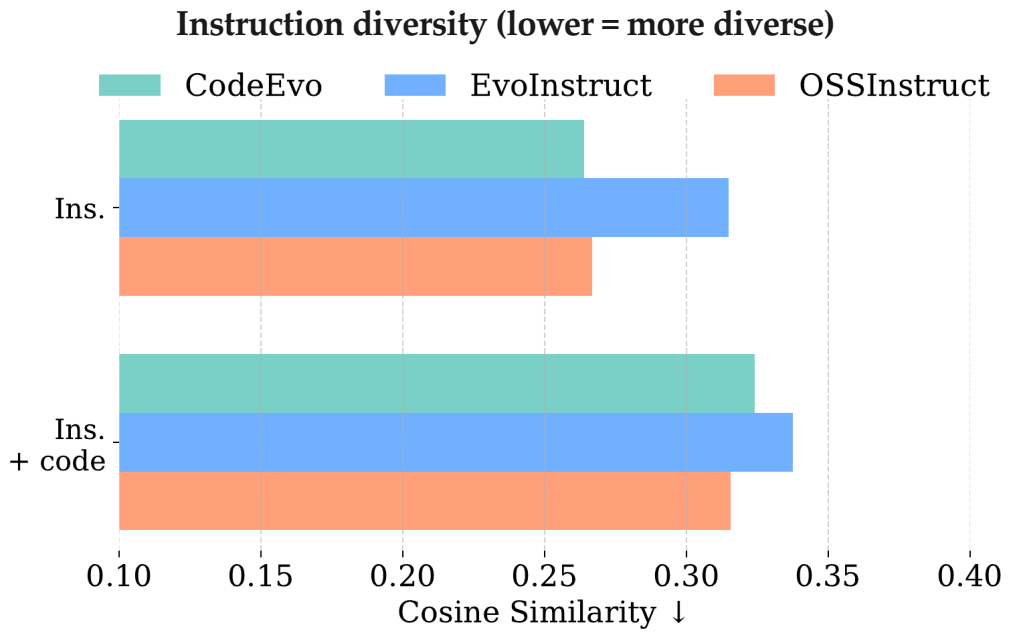
pass@1 (%) — Qwen3-8B backbone; gpt-oss-120B synthesis agents. Full table in paper.

Consistent wins: 32B & 120B agents

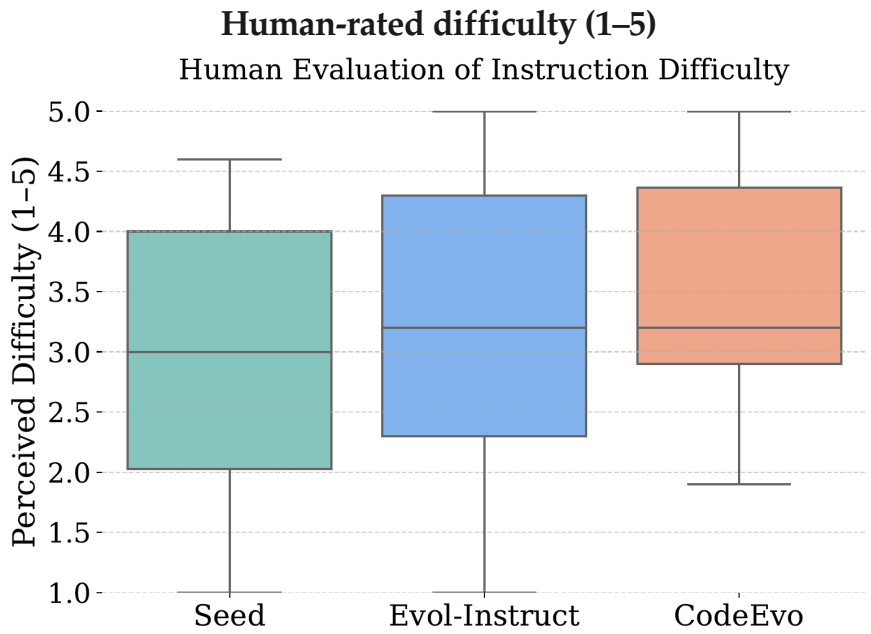
Compiler-in-loop: biggest gains on HE+/MBPP+

Hardest bench: +7.8 LiveCodeBench vs. base

More diverse — and genuinely harder



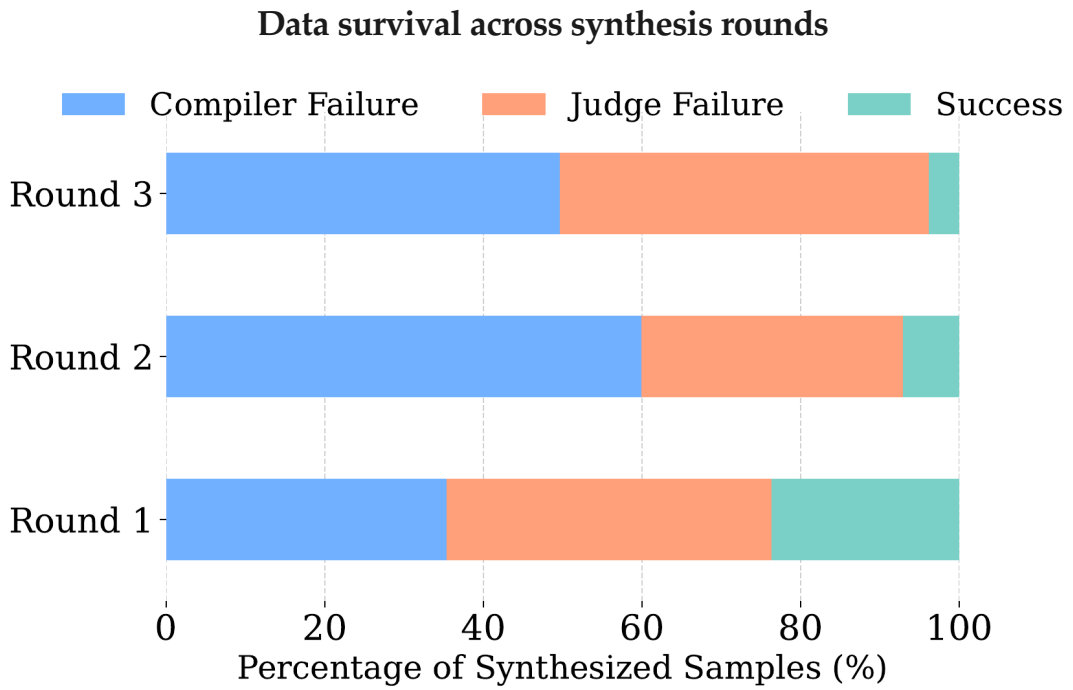
Lowest similarity among instructions; pairs as diverse as human-derived OSS-Instruct.



Highest mean (≈3.5) and higher lower-bound; Evol-Instruct fails to consistently harden.

Keyword guidance is a simple, effective signal that raises both **diversity** and **difficulty**.

Strict filtering, and robustness without seeds



Only a small fraction survives, and survival drops each round — tasks get harder, and we keep only grounded data.

Seed-free ablation

Removing all seed solutions barely changes results — sometimes it's even better.

Qwen2.5-Coder-7B	HE+ 80.9	MBPP+ 73.2
w/o seed	HE+ 80.7	MBPP+ 71.9
Qwen3-8B	HE+ 79.8	MBPP+ 74.1
w/o seed	HE+ 80.1	MBPP+ 73.9

→ The pipeline is genuinely self-grounding:
synthesized data outvalues the original seeds.

CONCLUSION

CodeEvo: verifiable, grounded code-data synthesis

Dual-agent interaction

Coder + Reviewer co-evolve instructions and code into grounded trajectories.

Hybrid feedback

Compiler determinism + semantic judgment → correct AND coherent, no labels.

Algorithm > volume

17K beats 75K; full 100K pushes further on the hardest benchmarks.

A scalable, verifiable path to advancing code intelligence with minimal human supervision.

Code & data: **github.com/QiushiSun/CodeEvo**

Thank you!

Schema-driven evolution in action

Seed ★

starting problem

Given an array nums, find the **next lexicographically greater permutation**; in place, with constant extra memory.

Evolved · 1 ★★

Schema + duplicates · circular

…now handle **duplicate elements efficiently** and output in a **circular format** where the last element points back to the first.

Evolved · 2 ★★★

Schema + linked list · size bounds

Given a **doubly linked list** of integers, find the next greater permutation — circular output, duplicates, and **up to 1000 nodes (values 0–1000)**.

Each step adds grounded logic & complexity — not a vague "make it harder". Every variant stays executable and verified.