Code as a Substrate for LLM Self-Improvement

Undergraduate Honors Thesis

Kaiyuan Liu

Department of Computer Science & Engineering, University of Washington
Advised by Prof. Natasha Jaques

March 2026

Research Arc: Three Connected Projects

Evaluation
LCB-Pro
FrontierCS → **Data Gen.**
AutoCode → **Self-Play**
(Ongoing)

Central claim: Code is uniquely powerful for evaluating,
training, and self-improving LLM reasoning.

Why code?

- **Executable** — correctness is automatically checkable
- **Formal** — zero ambiguity in evaluation
- **Scalable** — free verification, no human judges needed

Five Contributions

1. LiveCodeBench-Pro benchmark
2. FrontierCS open-ended benchmark
3. AutoCode data-generation framework
4. InvestESG multi-agent LLMs simulation
5. Toward Self-Play (preliminary)

1. LiveCodeBench-Pro: Measuring the Gap

Three systematic failures:

1. **Contamination** — static benchmarks overlap with training data; models may memorize, not reason
2. **Weak test cases** — small/random inputs accept many incorrect solutions
3. **No Case Analysis** — aggregate pass-rates hide *why* and *where* models fail

Our fix: capture problems **in real time** as elite contests end — *before* editorials are published.

LCB-Pro at a Glance

- **584 problems** from Codeforces, ICPC, IOI
- Captured live; problems post-date any training cutoff
- Annotated by **Olympiad medalists**
- Expert failure-mode analysis vs. human contestants

Industrial Impact

Used by Google to evaluate **Gemini 3** at launch.

Benchmark Design: Two Innovations

Difficulty tiers (Codeforces Elo):

Tier	Elo	Typical solver
Easy	$\leq$2000	Top 20%
Medium	2000–3000	Top 1–2%
Hard	$>$3000	$<$ 0.1%; rare

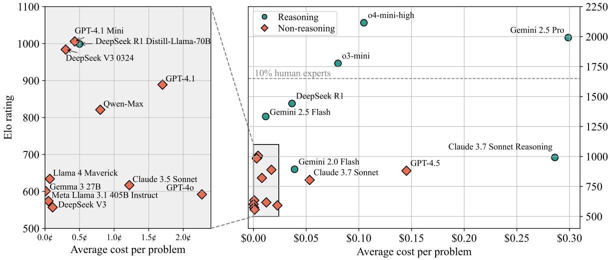
Cognitive-focus taxonomy (novel contribution):

- **Knowledge-heavy** — apply a known template
  e.g., Heavy Data Structures, FFT
- **Logic-heavy** — systematic derivation
  e.g., Combinatorics, DP transitions
- **Observation-heavy** — one non-obvious "aha"
  e.g., Game theory, ad-hoc, constructive

Why cognitive focus matters

Distinguishes fixable weakness (*missing knowledge*) from fundamental gap (*missing creative insight*).

Main Result: The Hard-Problem Wall

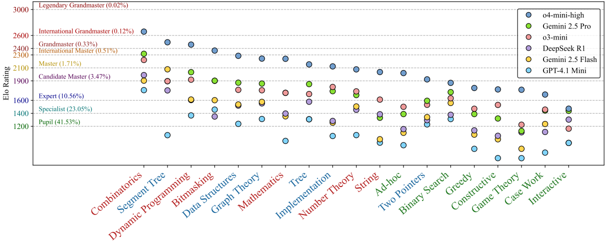


Striking Finding

Every frontier model achieves **0% on Hard** problems — including o4-mini, DeepSeek-R1, and Claude 3.7.

Best result on Medium:

- `o4-mini-high`: 53.5% pass@1
- Elo $\approx$ 2116 (top 1.5% of humans)
- Still far below Int'l Grandmaster ($\geq$2600)



Tag-wise Elo ratings — left: knowledge-heavy, right: observation-heavy

Pattern:

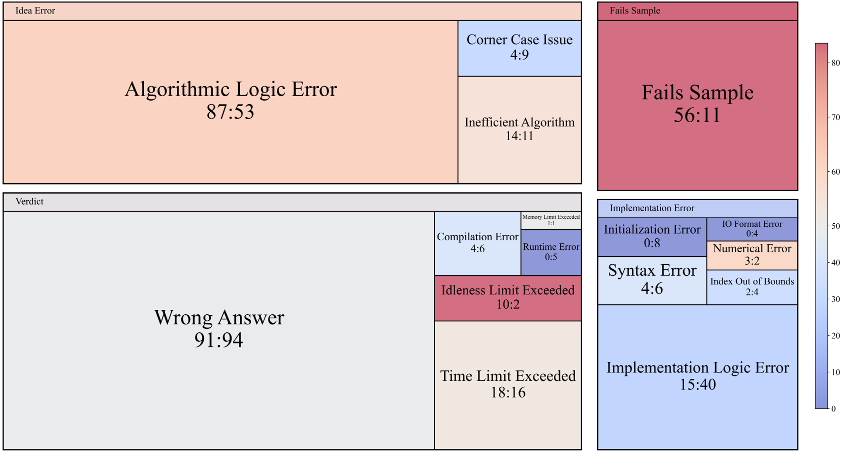
- 500–800 Elo gap between knowledge-heavy and observation-heavy categories
- Reasoning models gain most in *combinatorics*
- Near-zero gain in *game theory*, *ad-hoc*, *constructive*

Failure Mode Analysis

`o3-mini` vs. human (125 annotated failures each):

- Models: more **algorithm logic errors**
- Humans: more **implementation bugs**
- Models: almost zero runtime / I/O errors

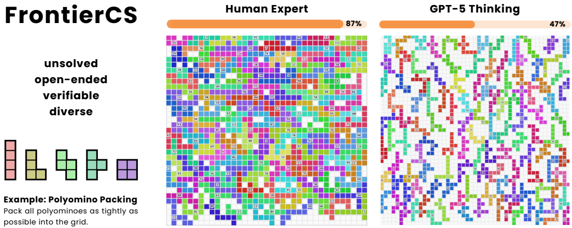
Failure Mode Analysis: LLMs vs. Human Contestants



Treemap of 125 annotated failures each for `o3-mini` and human contestants. Block size ∝ rejection count; red = higher `o3-mini` share, blue = higher human share.

2. FrontierCS: Open-Ended Evaluation

FrontierCS: Evaluating Without a Known Optimum



156 problems where no optimal solution is known,
but every solution is automatically scored.

- **Algorithmic Track** (107): NP-hard packing, scheduling, interactive problems
- **Research Track** (49): OS, HPC, AI, Databases, PL, Security

Result

Frontier models remain far behind human experts on both tracks. Increasing reasoning budget yields diminishing returns — models cannot discover the higher-level strategies humans use.

Example: Human 87% packing density; GPT-5 only 47%.

Why Continuous Verifiable Rewards Are a Big Deal

Two reward regimes for code:

	Binary	FrontierCS
Signal	pass / fail	$s \in [0, 100]$
Density	sparse	dense
Optimum	known	unknown
Ceiling	hard	open-ended

Key property

Continuous scores are still **automatically computable** — no human judges, no LLM-as-judge bias.

Why this matters for learning:

1. **Self-evolve methods**
   Self-evolve methods (GEPA, Alpha-Evolve) rely on a reward signal to drive curriculum generation. Continuous rewards allow it to adapt as the model improves.
2. **Test-time training**
   TTT needs a differentiable or rankable loss at inference.
3. **RL in general**
   Denser reward $\Rightarrow$ better credit assignment (?), less variance, smoother policy gradient. However, it also introduces new failure modes (reward hacking, local optima).

3. AutoCode: Code Verification & Data Generation

Why Code? Verifiability as the Key Ingredient for Self-Play

The core advantage

Running code *is* the oracle. No LLM judge, no human annotator. The verdict is provably correct given the test suite — not probabilistic.

Two requirements for code-based self-play:

1. Reliable Verification Pipeline

Given a problem + candidate solution, produce the **correct** verdict with near-zero error.

Noisy verdicts corrupt RL gradients — wrong data in, wrong policy out.

2. Novel Problem Generation Pipeline

Generate new, unseen, auto-verifiable problems that track the solver's evolving frontier.

Without novelty the model memorises seed problems rather than generalising.

AutoCode is the unified framework that delivers both.

AutoCode: Reliable Test Case Generation

Method	Consist.	FPR	FNR
CodeContests	72.9%	7.7%	46.3%
CodeContests+	79.9%	8.6%	31.6%
TACO	80.7%	11.5%	26.9%
HardTests	81.0%	12.1%	25.8%
AutoCode	**91.1%**	**3.7%**	**14.1%**

Why FPR is more dangerous than FNR for RL:

High FPR ⇒ Poisoned Policy

Wrong solution rewarded ⇒ policy pushed toward **incorrect reasoning**. Model enters a reward-hacking loop it cannot exit.

High FNR ⇒ Wasted Examples Only

Correct solution rejected ⇒ in GRPO all rollouts get reward 0 ⇒ **zero gradient**. Problem silently skipped.

BuildValidator($\mathcal{S}$)

$\mathcal{E} \leftarrow \text{LLM.GenCases}(\mathcal{S};\ \text{10 valid},\ \text{30 near-valid})$
$\{V_1, V_2, V_3\} \leftarrow \text{LLM.EmitValidators}(\mathcal{S})$
return $V^\star = \arg\max_V \sum_{(x,l)\in\mathcal{E}} [V(x)=l]$

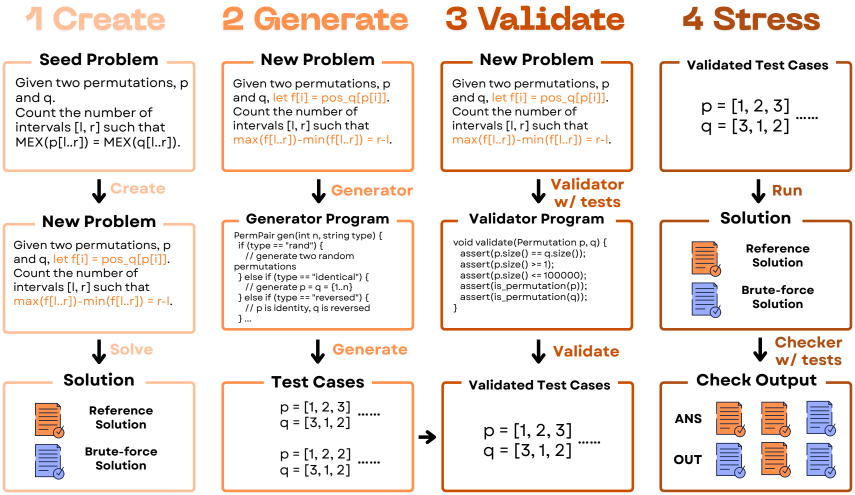
BuildGenerator($\mathcal{S}$, $V^\star$)

$\mathcal{G}_1 \leftarrow \text{ExhaustiveSmall}(\mathcal{S})$
$\mathcal{G}_2 \leftarrow \text{RandomExtreme}(\mathcal{S})$ *overflows, hash-collisions*
$\mathcal{G}_3 \leftarrow \text{TLEInducing}(\mathcal{S})$ *worst-case size/structure*
$\mathcal{T} \leftarrow \text{Filter}(\mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_3,\ V^\star)$
return $\text{SampleWithCoverage}(\mathcal{T})$

BuildChecker($\mathcal{S}, \mathcal{R}, V^\star$)

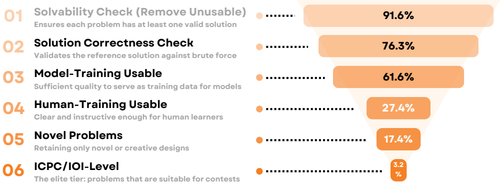
$\mathcal{Q} \leftarrow \text{LLM.GenScenarios}(\mathcal{S}, \mathcal{R};\ N{=}40)$
$\mathcal{Q} \leftarrow \{q \in \mathcal{Q} : V^\star(q.\text{in}) = \text{valid}\}$
$\{C_1, C_2, C_3\} \leftarrow \text{LLM.EmitCheckers}(\mathcal{S})$
return $C^\star = \arg\max_C \ \text{acc}(C, \mathcal{Q})$

AutoCode: The Full Closed-Loop Framework



Validator-Generator-Checker pipeline achieves **98.7% consistency** with official Codeforces judges on 720 unfiltered problems.

AutoCode: Generating Novel Problems



Quality distribution graded by elite competitive programmers. Levels 1–2 = automated checks; Levels 3–6 = human evaluation.

Industrial Impact

Adopted by **Stepfun** for RL post-training data generation.

Dual-verification protocol:

1. LLM generates: problem statement + `std.cpp` (efficient) + `brute.cpp` (slow but reliable)
2. Run both on full test suite; accept only if outputs agree on every case
3. Result: correctness 86% → **94%**; filters 27% of error-prone problems

Key findings:

- $>$ 60% of verified problems are RL training-ready
- 3.2% reach ICPC/IOI competition quality
- **4.2%: LLM generates problems it cannot itself solve** — natural frontier data for self-play
- Difficulty gain is the best proxy for quality — larger mutation = more novelty

4. InvestESG: LLMs in Social Dilemmas

InvestESG: LLMs as Agents in a Climate Social Dilemma

Setting: InvestESG MARL platform — companies choose climate mitigation investment; investors allocate capital. Textbook **social dilemma**: individually costly, collectively beneficial.

We replace RL agents with LLMs (GPT-4.1, o4-mini) compare against PPO baseline, social planner, and human participants.

F1: LLMs Over-Cooperate — But Why?

LLMs invest *far* more than PPO or the social optimum.

- Tool-use (cost-benefit table) *reduces* over-cooperation but does not eliminate it
- Context-free prompts (no "climate" language) also reduce it — but not to zero

⇒ Both numerical reasoning limits *and* deep prosocial priors are at work

F2: Communication ⇒ Collusion

Structured inter-company negotiation causes companies to coordinate on **low** mitigation.

- >50% of dialogue is alliance-building / coalition maintenance

Conclusion: LLMs are better human proxies than RL agents for complex social dilemmas.

5. Toward Self-Play

Do AutoCode-generated problems improve performance on real human-designed problems?

Configuration	Pass@1	Δ
Qwen3-8B (base)	46.2%	—
+ RLVR on AutoCode data	**48.5%**	**+2.3%**

133 AutoCode problems, 5 epochs, GRPO,
binary pass/fail reward from AutoCode test cases.

Eval: held-out LiveCodeBench problems.

What this demonstrates

- Fully **synthetic** problems improve performance on real human-designed problems
- AutoCode's low FPR provides clean rewards.
- Principle is validated; *scale and curriculum* are the next challenge.

Why Natural Data Isn't Enough: The Pyramid Problem

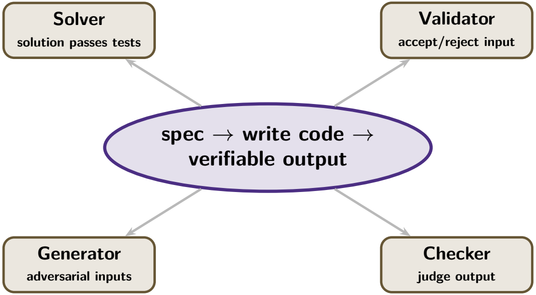
Three problems with natural CP data:

1. **The effective zone is tiny.**
   Problems in the proper Elo range where frontier models sometimes succeed and sometimes fail are a small fraction of public archives.
2. **Human curation is expensive.**
   ICPC/IOI-quality problems require expert setters working for days. Valid novel problems require hours of work and worth hundreds of dollars in human labor.
3. **The zone shifts.**
   As the model improves, what was "hard" becomes easy — the curriculum must continuously track the frontier.

What AutoCode enables

Scale: generating thousands of verifiable problems, automatically.
Harder: seed moderate problems $\rightarrow$ avg. $+334$ Elo gain; novel ones $\rightarrow$ $+498$ Elo.
Easier: freely lower the seed difficulty.

The Co-Evolution Hypothesis: Solving $\approx$ Setting



All four roles share the same primitive: understand the problem's correctness criterion, then write a program that enforces it.

The shared core

Setting and solving both require the model to **understand what "correct" means** for a given problem — well enough to write a program that embodies it.

There is **no categorical distinction** between the solver and the setter.

The hypothesis

Train a better solver $\Rightarrow$ better setter for free.
Iterate. No labels. No human oracle.

Toward Iterative Self-Play: Vision and Open Challenges

The vision: iterative solver–setter co-evolution

1. Solver trains via RL on AutoCode-verified corpus
2. Post-trained solver becomes the new AutoCode setter
3. Setter generates harder, better-specified problems
4. Harder problems push the solver's frontier further
5. Repeat — **no human curation at any step**

Current limitations:

1. **Novelty ceiling** — LLMs recombine existing algorithmic patterns rather than inventing new ones; frontier is limited
2. **Scale** — 133 problems / 8B model; frontier-scale iteration requires significantly more compute and problems
3. **Evaluation gap** — LLM judgment of problem quality has near-zero correlation with human experts; measuring real improvement on Problem Generation is hard

6. Conclusion

Summary of Contributions

1. LiveCodeBench-Pro

584-problem benchmark. **Every frontier model: 0% on Hard.** Models fail on creative observation, not implementation. Used by Google for Gemini 2.5 Pro evaluation.

2. AutoCode

91.1% consistency on 7,538 problems. **94%** novel problem correctness. Adopted by Stepfun for RL post-training.

3. FrontierCS

156 open-ended CS problems with automatic continuous scoring. Frontier models far below human experts; more compute doesn't close the gap.

4. InvestESG

LLMs over-cooperate in social dilemmas, matching human behavior. Structured communication produces emergent collusion. Validates LLMs as human simulators.

5. AutoCode-Train

+2.3% on LiveCodeBench (Qwen3-8B, GRPO, 133 problems). Still working on!

Thank You

Questions welcome

lky04@cs.washington.edu