From Tokens to Productivity:

Scalable Agentic Workflows for Engineering and Science

Ligeng Zhu

NVIDIA Research

Monthly Avg — Tokens: 1.28T Lines of Code: 16.6M

2026-Oct: Ligeng's Token Cost

Tokens
1,469,611,345,436

Lines of code generated
19,397,307

2026-Sept: Ligeng's Token Cost

Tokens
865,549,942,365

Lines of code generated w
22,376,894

2026-Aug: Ligeng's Token Cost

Tokens used
1,590,841,944,468

Lines of code
16,426,139

2026-July: Ligeng's Token Cost

Tokens used
1,210,508,599,012

Lines of code
8,294,718



About Me

Scalable and efficient agents · Multi-modal VLMs



Ligeng Zhu

Researcher, NVIDIA Research
Efficient AI Team (PI: Song Han)

- Agents for L2–L4 automatic research & engineering
- Scalable and efficient agent harnesses and swarms
- Efficient multi-modal VLMs

13,381
citations · h-index 29 (Google Scholar, Nov 3, 2026)

Core author projects at NVIDIA Research

1. **Humanize**: Judgement Engineering for Agentic Coding

powered by Humanize

2. **KDA**: Kernel Design Agents
3. **QDA**: Closed-Loop LLM Quantization Design with Agents
4. **Humanize-Prover**: Towards Scalable and Efficient Math Engineering
5. **Lean Formalization of Geo Poincare**: A Lean Formalization of the Hamilton–Perelman Proof of Thurston's Geometrization Conjecture
6. **Humanize Olympiad Agents**: A Multi-Session Workflow for Olympiad Problem Solving with Solve, Review, Wiki, Verify
7. **Humanize MLE Agents**: Multi-Agent Session Alternation Beyond Single-Agent Limits
8. **Parason**: subtask and trial parallelism in LLM reasoning
9. **OckBench**: measuring the efficiency of LLM reasoning
10. **EGM**: efficient visual grounding language models
11. **Fast-dVLM**: block-diffusion VLM converted from an autoregressive VLM
12. **NVILA**: efficient frontier visual language models; first multi-modal model at NVIDIA; bootstrap NVIDIA **Cosmos** data pipeline

Core author projects at MIT

1. PockEngine: sparse and efficient fine-tuning in a pocket
2. On-device training under 256KB memory
3. Distributed training across the world
4. Deep Leakage from Gradients (most cited paper at MIT HAN Lab)
5. ProxylessNAS

Open source on GitHub · ~26k stars total

NVlabs/Sana	★ 9.8k
Lyken17/pytorch-OpCounter	★ 5.1k
NVlabs/VILA	★ 3.9k
PolyArch/humanize	★ 1.6k
NVlabs/kda	★ 1.5k
mit-han-lab/proxylessnas	★ 1.5k
Lyken17/Efficient-PyTorch	★ 1.1k
Lyken17/pytorch-memonger	★ 602
mit-han-lab/tiny-training	★ 528

Token maxing: what Humanize + KDA did in 2026

O(tokens) = O(productivity): mainly done July–October (4 months), small team (< 10 core members)

Humanize · math, science, software

11-02	Formalized <omit> docs and Located <omit> bugs for **NVIDIA** <omit> hardware (omitted due to external share)
10-25	Formalized **8 frontier math problems** from OpenAI's release in Lean, incl. first **4D Kakeya** & **Hilbert–Smith**
10-09	Lean formalization of the Hamilton–Perelman proof of **Thurston's Geometrization**
10-02	**Lean-Eval #1** again: 251/304 (first #1 on 08-18)
09-23	**Quantum** related breakthrough: <hidden due to external share>
09-12	ProgramBench, all 200 tasks: **13.5%** full score vs 4.5% best CLI agent
08-15	Kaggle: **top 5%** in 14 of 19 completed competitions
08-12	**Quantum** related breakthrough: <hidden due to external share>
07-29	IPhO 2026: all **23/23** theory subproblems formalized and verified in Lean
07-22	IMO 2026 **6/6** in Lean with GPT-5.6-Sol and Kimi-K3, 3.2× faster than AxiomProver
07-12	**Quantum** related breakthrough: <hidden due to external share>
07-08	Model ability booster: Physics Cup **22** → **33** → **40**/50, HLE subset 29 → 63%
06-26	PutnamBench **670/672** (99.7%), later 672/672
03-01	gem5: SCons → CMake migration, **567 files**

KDA · kernels and serving systems

10-25	Nebius tops **6 of 8** tracks on **GPU MODE** using KDA
10-13	Megatron-LM (merged): Gated DeltaProduct decode **1.26×** e2e, GB300
09-28	KDA²: Kimi Delta Attention **2.96×** vs FlashKDA on B300
09-25	Databricks tops **all tracks** on **SOL-ExecBench** using KDA
09-23	Rubin: <omit due to external sharing> **1.84×** vs SGLang
09-03	cuDNN: found a hidden TMA overflow, fix **merged** (cudnn-frontend #861)
08-26	Flash-KMeans **7.3×** · DeepSeek-V4-Flash megakernel −6.2% TPOT
08-14	MSA indexer in production: prefill **6.5×**, decode up to 3.3× (B300)
08-02	FlashInfer contest re-run: **1.17–1.69×** over the human SOTA
07-22	DeepSeek-V4-Pro mHC **−28.1%** latency (TensorRT-LLM MR)
07-22	Rubin with KDA + <omit due to external sharing> **94.9%** SOL for GEMM backward
07-11	DeepSeek-V4-Flash on SM120: **2.5×** end to end (295 → 743 tok/s)
07-10	ModelOpt NVFP4 SVDQuant kernel in **4 days** (FlashInfer #3858)
07-02	**#1** on SOL-ExecBench L1 single-operation track (0.7608)
06-05	SGLang: **40+** kernels; Qwen3-Next allreduce fusion +71.4% thpt
05-15	MLSys'26 FlashInfer contest: **top 3 in all tracks**

Various achievements across **agent flow**, **infra**, **kernel**, **math**, **formal**, **quantum**

Scale tokens (top users at NV), not humans (3 fulltime, 6 interns)

Monthly avg: **1.28T** tokens · **16.6M** lines of code
Oct 1.47T / 19.4M · Sep 0.87T / 22.4M · Aug 1.59T / 16.4M · Jul 1.21T / 8.29M

Code is cheap. Finished work is not.

How do we turn model tokens into sustained, verifiable productivity?

Belief: LLMs > humans

- More **knowledgeable**: >>30T tokens of pre-train + post-train
- More **skillful**: IMO, IPhO, IOI, ACM, Putnam, Erdős problems
- Remember more **tricks**: 1M context >> a human's 20–30k
- **7×24**: never tired, never rests



"Why do you think you could be smarter than Codex?"

OK, but why do my agents still fail?



"You're as smart as Doubao!"

- Agents now write code in seconds, but **the agent that writes the code is a weak judge of whether it is done**
- Long single sessions **drift**, **over-claim** and **reward-hack**: in 45 of 118 real-loop postmortems, the builder claimed more than it had done [arXiv:2610.08900]
- Bigger models help, but **workflow design is a major source of capability**, not a thin layer around the base model: with HOA, open Kimi-K3 matches GPT-5.6-Sol on 2026 olympiads [HOA, arXiv:2610.13632]
- **Good models = good students**
  - Good students need **mentorship**
  - Good models need **workflow**

Humanize (arXiv 2610.08900) p.1, p.8; HOA (arXiv 2610.13632) p.10

NVIDIA

PART 1

Humanize: Judgement Engineering for Agentic Coding

Why agents fail, and three flows that fix it: RLCR, Ralph, Flame Chase

Sihao Liu[1,2], **Ligeng Zhu**[1], Zijian Zhang[1,4], Dongyun Zou[3,5], Zhengyang Zhang[4], Changye Li[4], Song Bian[1], Song Han[1,3], Tony Nowatzki[2]

[1]NVIDIA · [2]UCLA · [3]MIT · [4]Tsinghua University · [5]Nunchux AI

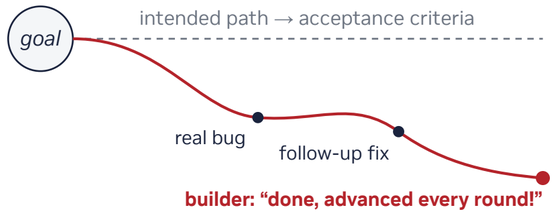
Why agent's native /goal fails?

The completion decision is left with the agent doing the work

1. **Mis-alignment between agents and humans**: models have their own preferences and often finish tasks their way rather than the human's

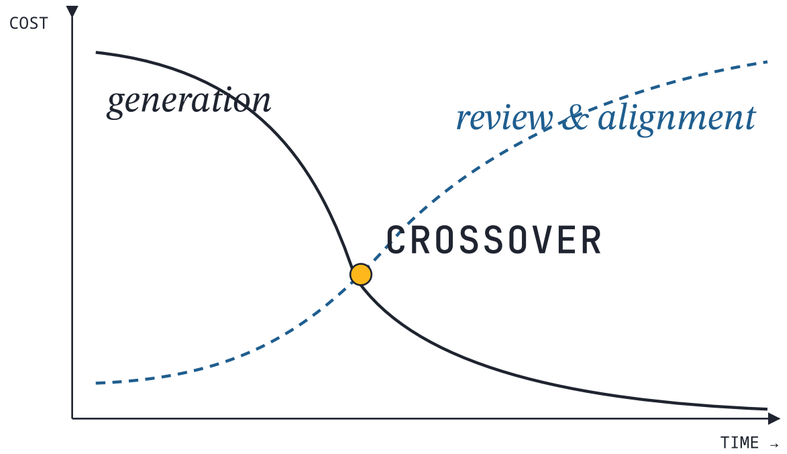
> For example, Ligeng is invited to give a talk. If he just replies **"yes"** with no follow-up questions, it ends in chaos. Why?
>
> **When?** Friday or Monday? | **Where?** Online or offline? | **If online**, Zoom, Teams or Google Meet? | **If offline**, how big a room? | ... and many more

2. **Over-claiming and drifting from "what has been done"**



Productive motion is not goal progress
KDA: a builder made **its own first kernel the baseline**, then claimed done

3. **Shared blind spots and lack of evaluation**: models favor their own outputs, and always write thousands of lines in minutes first, then tend to accept them



Past the crossover, review and evaluation, not generation, dominate the cost

Fast implementation (a.k.a. 16M LoC per month) × **Weak alignment** × **Zero judgement** = **Fast, wrong garbage work** (boom!)

Humanize (arXiv 2610.08900) p.1–3, p.8–9

NVIDIA

Three levels of agentic work

Same model; what wraps it decides how long it can work and who judges the result

Model level

a single API call **(1 - 10 mins)**

no environment, no feedback

→

Tool level

one CLI agent session **(30 mins - hours)**

real environment, but the agent judges itself

→

Flow level

many coordinated sessions **(days - weeks)**

plan → build → independent review, repeated

```
response = client.responses.create(
  model="gpt-5.6-sol",
  input="Profile FP8MoE and "
        "improve the implementation."
)
# one shot: no tools, no feedback
```

```
>_ OpenAI Codex
│ permissions: YOLO mode

› Profile FP8MoE @fp8_moe.py and
  improve the implementation
# the agent decides when it is done
```

```
› /humanize:gen-plan \
    --input draft.md --output docs/plan.md
› /humanize:refine-plan \
    --input docs/plan.md
› /humanize:start-rlcr-loop docs/plan.md
# a reviewer decides when it is done
```

Everything we build is **flow-level**: Humanize, KDA, QDA, HOA, Humanize-Prover

Humanize (arXiv 2610.08900)

NVIDIA.

What is a flow-level agent?

A new kind of “programming”: scripts that manage agents across sessions

*Imagine, years from now: a new undergrad in **Intro to Programming** views Python the way we view assembly*

	Before: C++ / Python	Now: flow-level harness
What is the hardware?	CMOS / transistors	**agents / sessions**
What is the program?	C++ / Python code	**flow scripts (hmz exec -f …)**
What is a function call?	a subroutine	**one agent call**
Where does state live?	variables, memory	**plan, repo, evidence on disk**
What is control flow?	if / break / while / **for-loop** ↑ **Loop Engineering**	
Who checks correctness?	asserts, unit tests	**a reviewer model, oracles**

Markdown-level harness

Single session, **soft control**

AGENTS.md, skills, prompts: like comments, the model may ignore them

Flow-level harness

Cross sessions, **hard control**

Scripts decide who runs, what passes and when it stops: like code

Analogy: speaker’s framing · Humanize (arXiv 2610.08900) Sec. 3

NVIDIA

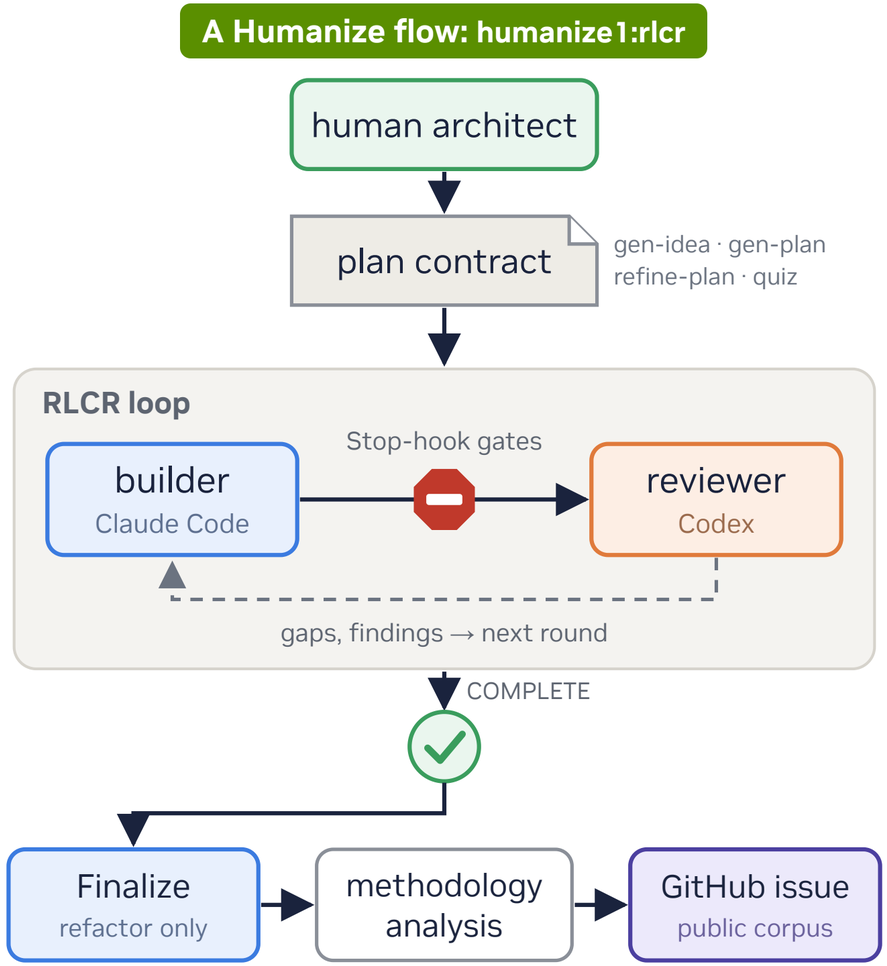
Humanize flow #1: RLCR in pseudo-code

RLCR = Ralph loop with Codex review: deterministic hooks route the work, an independent reviewer decides “done”

```
# 1. plan: the human must be able to judge it
plan = gen_plan(idea)                          # Claude drafts, Codex reviews
plan = refine_plan(plan, human)                # goal + acceptance criteria
quiz(human, plan)                              # does the human get it?

# 2. RLCR: builder rounds, fresh reviewer
for r in range(1, 43):                         # cap: 42 rounds
    work = builder.run(prompt)                 # code + contract + claims
    if not hook.gates_pass(work):              # 72 gates, not a model
        continue                               # blocked: same round
    rv = Reviewer().review(plan, repo, work)   # no builder reasoning
    if rv.verdict == COMPLETE: break
    if rv.verdict == STOP and r % 5 == 0:      # alignment review
        return STOP
    if stalled(rv, rounds=3): return STOP
    prompt = next_prompt(plan, rv.findings)    # builder can't edit

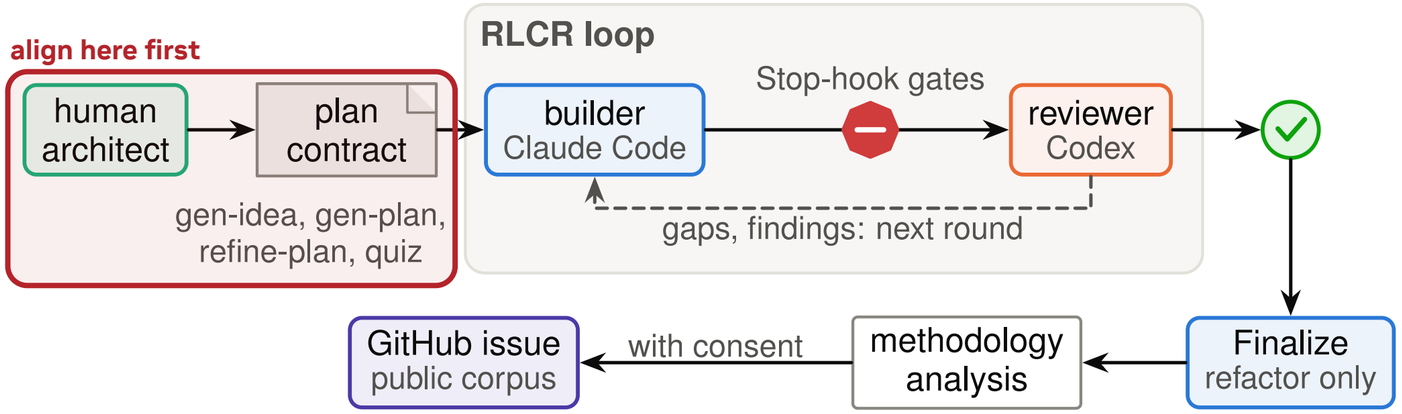
# 3. code review: no round cap
while f := codex_review(diff_only=True):
    builder.fix(f)
finalize()                                     # refactor only
methodology_analysis()                         # postmortem → corpus
```



Humanize (arXiv 2610.08900) Sec. 3, Fig. 3–4; simplified

1 · Mis-alignment between agents and humans

Natural language is ambiguous: the agent fills the gaps with its own guesses



Humanize RLCR: the human and the plan are judged first

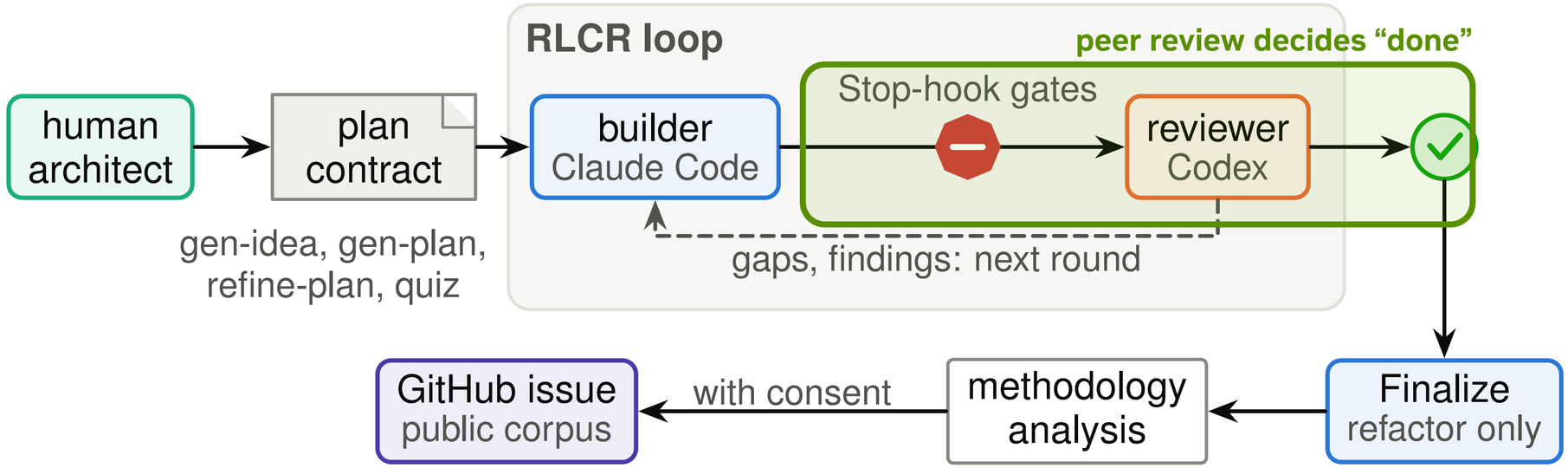
- Think hard **before** any code
- **Plan contract**: goal + acceptance criteria
- **Quiz**: does the human really get the plan?

Agents and the human on the same page, **before the first line of code**

Humanize — design initiatives; Humanize (arXiv 2610.08900) Sec. 1–2, Sec. 3.1

2 · Over-claiming and drifting from “what has been done”

Smart = tricky: an agent that cannot find a way to hack the reward is not smart enough



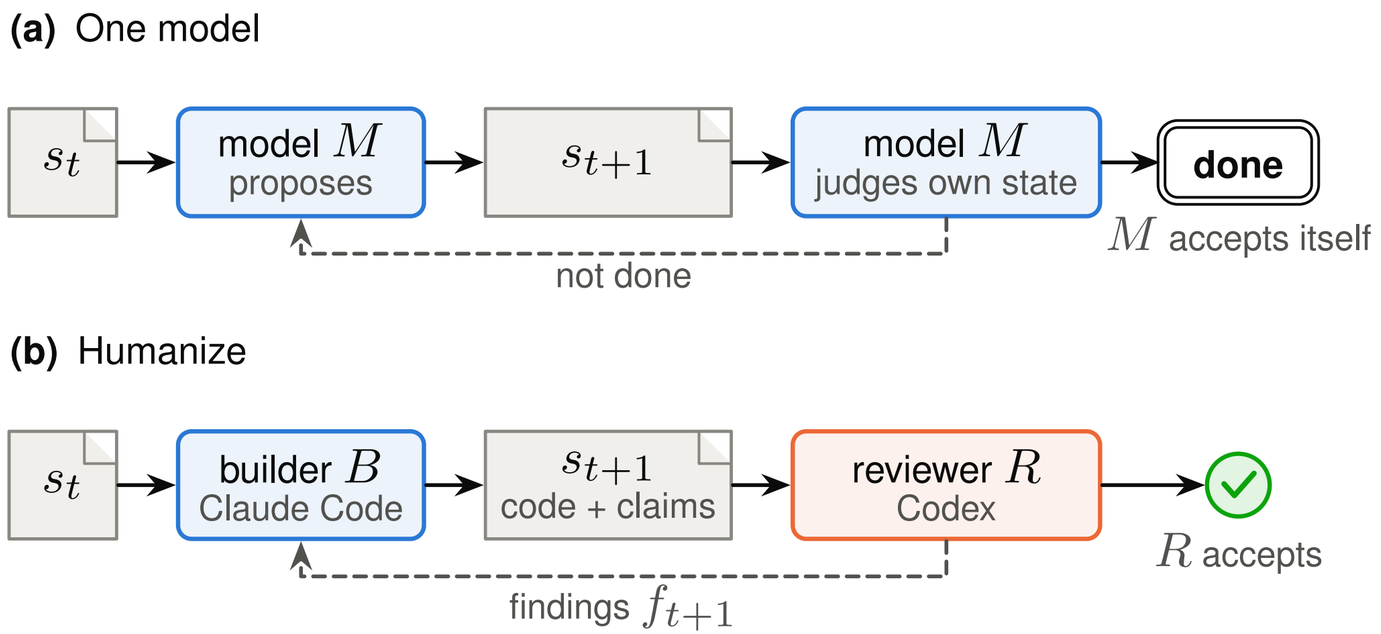
Every round passes the gates and an independent reviewer

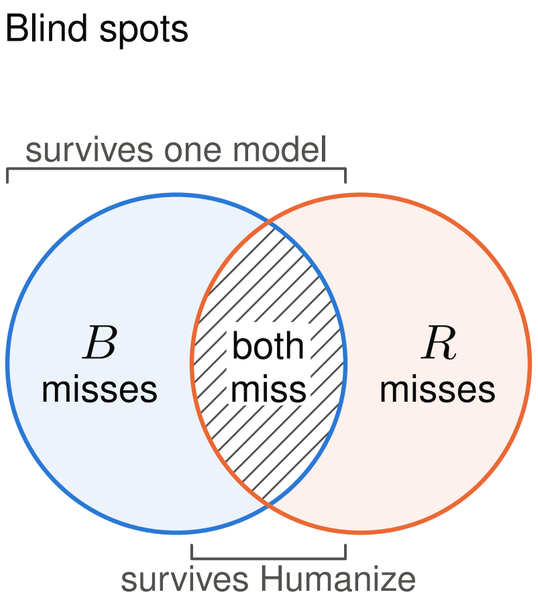
The reviewer, not the builder, decides “done”

Humanize — design initiatives; Humanize (arXiv 2610.08900) Sec. 3.2, Sec. 6

3 · Shared blind spots and lack of evaluation

Whatever a model gets wrong, it also tends to accept





Pr[defect accepted] = **b · m_R · m_O** same model: $m_R \approx 1$ **another vendor: $m_R \downarrow$** **real eval: $m_O \to 0$**

Humanize (arXiv 2610.08900) Fig. 2, Fig. 6, p.3, p.8 · b: builder miss · m_R: reviewer miss · m_O: oracle miss

NVIDIA

RLCR achievements: Humanize improves accuracy

Same model; more time in a coordinated flow buys accuracy (test-time scaling)

Model level
a single API call **(1 - 10 mins)**
no environment, no feedback

→

Tool level
one CLI session **(30 mins - hours)**
Codex / Claude Code, /goal

→

Flow level
many coordinated sessions
(days - weeks)
Humanize: RLCR

50-problem subsets	Model level	Tool level	Flow level (Humanize)
PutnamBench · GPT-5.6-Sol	3	46 +43	**50** +47
PutnamBench · Kimi-K3	1	4 +3	**47** +46
Physics Cup · GPT-5.6-Sol	31	41 +10	**44** +13
Physics Cup · Kimi-K3	35	37 +2	**42** +7

Same tokens, different performance — **(work)flow amplifies the model performance**

HOA (arXiv 2610.13632) Table 3 and Fig. 3; PutnamBench ablation marked provisional in the paper

NVIDIA

RLCR achievements: Humanize v1 in the wild

An open Claude Code plugin, grown from its own users' postmortems

68
versions on main in 108 days

1,468
GitHub stars · 130 forks · PRs from 34 authors

118
postmortems of real loops, filed by users

72
mechanical gates (Stop hook 975 → 2,221 lines)

Software

gem5: SCons → CMake + Ninja, **567 files**, 133 commits
Loom: **2.34×** proposal throughput in a 180 s budget

Kernels

KDA, MLSys'26 FlashInfer Full-Agent: **1st MoE**, 2nd DSA, 3rd GDN
DSA indexer **19.08×** mean speedup

Math & olympiads

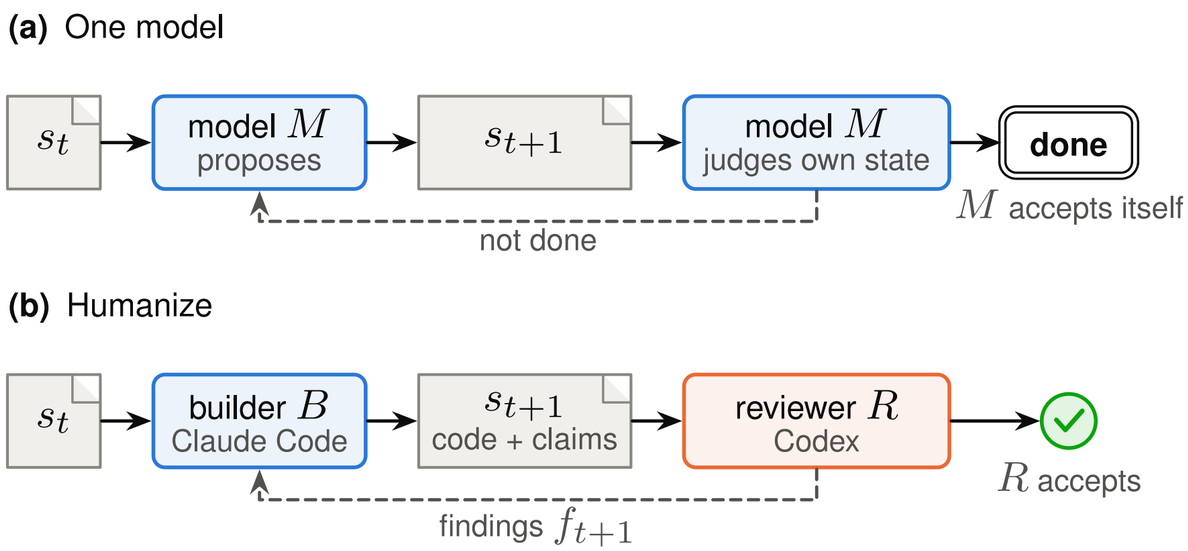
PutnamBench 672/672 · **#1 Lean-Eval** (legacy board)
IMO 2026 **6/6** in Lean · IOI 2026 6/6 full scores

Lesson: judgement engineering keeps agents running stably for longer → test-time scaling for agents

Humanize (arXiv 2610.08900) Sec. 3.3, Sec. 4–6, App. A · github.com/PolyArch/humanize

RLCR important takeaways: implementer ≠ reviewer model

A model that judges its own work shares its own blind spots: whatever it gets wrong, it also tends to accept



One model accepts itself; Humanize stops only when **another model** accepts

Same model in both roles

- Fails to **correct its own reasoning** [Huang et al., ICLR'24]
- **Favors its own outputs** [Panickssery et al., NeurIPS'24]: $m_R \approx 1$

Claude builds, Codex reviews

- Pr[defect accepted] = $b \cdot m_R \cdot m_O$: **both** must miss it
- Fresh process each round; judges **evidence, not reasoning**

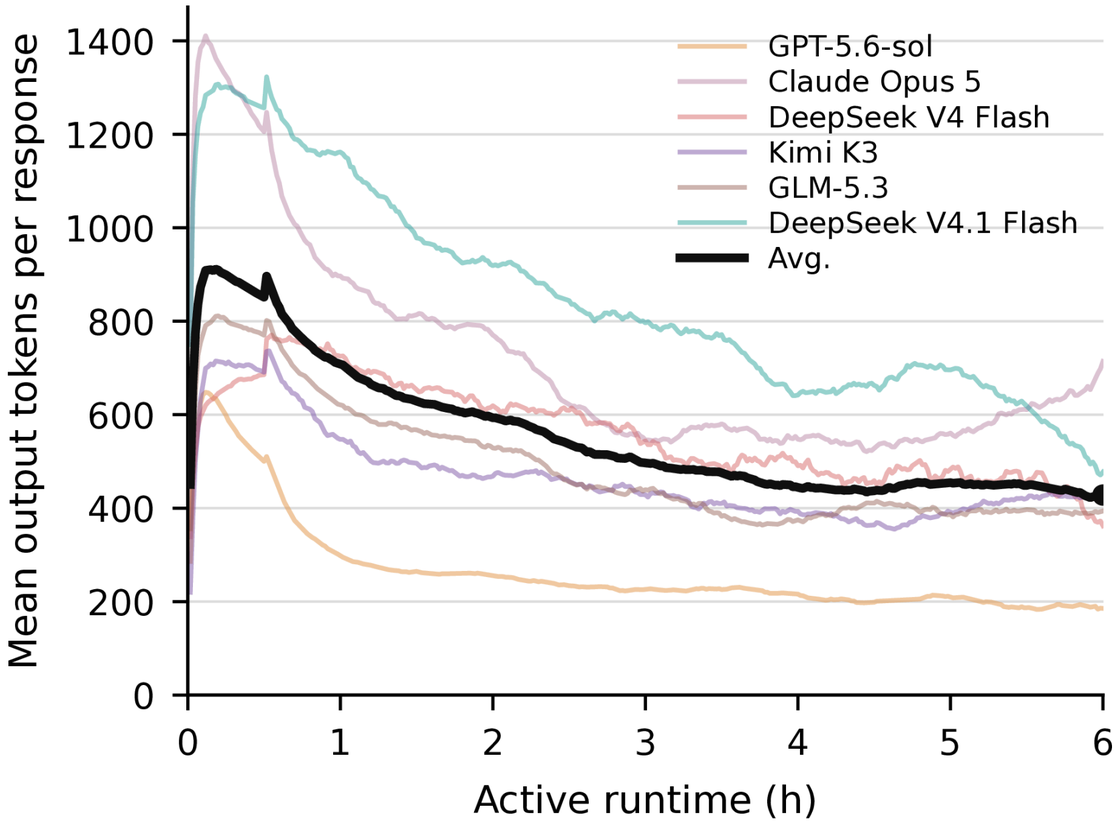
Surprise We find the **GPT family is always the best reviewer model**, from GPT-5.3 to GPT-6.1

Different models, different blind spots · unless a complete oracle (Lean, contest judge) checks the claim

Humanize (arXiv 2610.08900) Sec. 2.1, Sec. 3.2–3.3, Sec. 5, Fig. 2, Fig. 6b · GPT-reviewer finding: team observation across Humanize runs

Why not just /goal? The thinking budget shrinks!

Long-horizon deceleration: as a session grows, the agent reasons less per step and its gains flatten



Mean output tokens per response, single /goal session (30-min window)

Tokens / response	Hours 0–2	Hours 3–5	Change
Claude Opus 5	1,008	555	**−45.0%**
GLM-5.3	658	389	**−41.0%**
DeepSeek-V4.1-Flash	1,138	696	**−38.8%**
GPT-5.6-Sol	337	214	**−36.5%**
Kimi-K3	586	390	**−33.4%**
DeepSeek-V4-Flash	682	479	**−29.8%**

- **Gains come early**: a session's first 5 experiments keep **97.8%** of its gains
- **Resetting the context restores it**: HMA keeps **86.4%** of its tokens per response, vs **60.7%** for single sessions

Do not stretch one session: **end it, keep the work, start fresh**

HMA (arXiv 2610.13494) Sec. 1–2, Fig. 2–4, App. B.2, App. D.1 Table 6, App. E · thinking budget measured as mean output tokens per response

NVIDIA

Humanize flow #2: Ralph for long-horizon work

Same task, fresh session every round: progress lives in the repository, not in the conversation

```
tree = repo                          # code + evidence
while budget_left():                 # time / token cap
    s = agent.fresh_session()        # empty context
    tree = s.improve(task, tree)     # same task
    drop_stale_notes(tree)           # keep tested code
```

	RLCR	Ralph
Ends when	reviewer says COMPLETE	the budget runs out
Each round	builder session + fresh reviewer	one fresh session
Best for	a defined deliverable	open-ended optimization

Why it works

- Every session starts with a **full thinking budget** instead of a long, decaying context
- Only **tested code and experimental evidence** carry over; dead ends are cleaned up
- The program, not the agent, decides **when to stop**

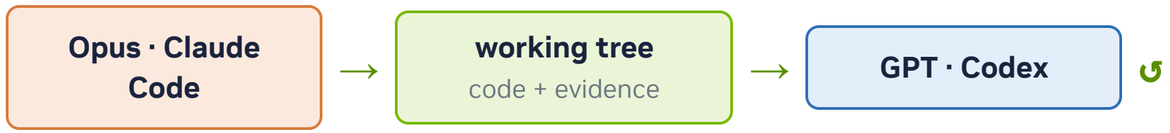
KDA: 24 h of Opus under Ralph, then 12 more hours: **3.61×** → **3.75×** geomean on five kernels; switching the model does better (next: Flame Chase)

HMA (arXiv 2610.13494) Sec. 3; KDA (in submission) Sec. 3.1, Sec. 4 continuation ablation · hmz exec -f ralph_loop

NVIDIA

Humanize flow #3: Flame Chase, alternate models, keep the work

Agents from different providers take turns on the same task, each turn in a fresh session




```
tree = repo                          # code + evidence
while budget_left():
    for agent in [opus, gpt]:        # two providers
        s = agent.fresh_session()    # no shared chat
        tree = s.improve(tree)       # hand over the tree
```

- No shared conversation: a fresh model re-reads the code, so **one model's blind spots don't carry over**
- **Anthropic's performance take-home**: **864 cycles**, **1.72×** faster than Anthropic's interview bar (next page)
- Same idea on MLE-bench: HMA beats the stronger model alone in every pair (page 20)

Switching models breaks plateaus: 12 more hours after 24 h of Opus (3.61×), KDA geomean gain



12 h, geomean speedup	Claude /goal	GPT /goal	Flame Chase
A100	1.78×	1.33×	**2.10×**
H200	1.51×	1.33×	**1.60×**

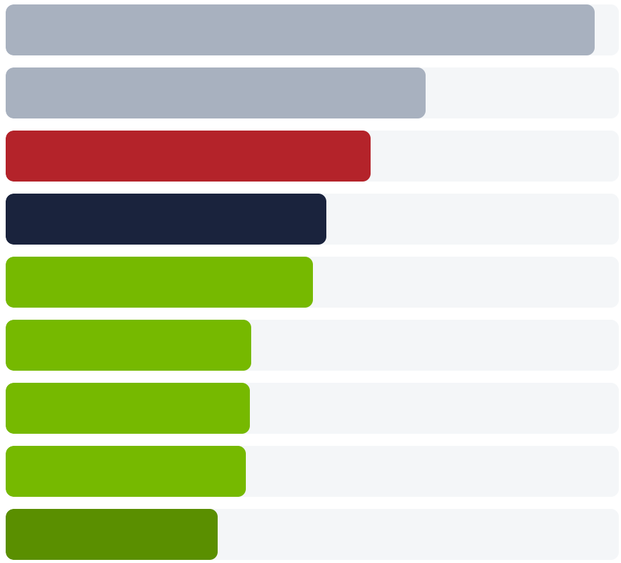
KDA (in submission) Sec. 3.1, Sec. 4.5, Table 4 (A100/H200, with skills); github.com/humanfia/flame-chasoid-flows

Flame Chase dominates Anthropic's performance take-home

Optimize a batched tree-walk kernel on a simulated VLIW + SIMD machine · clock cycles, lower is better

- Built to be **AI-resistant**: VLIW + SIMD is rare, so little of it is in training data; it tests how fast you **build a harness**
- Anthropic's interview bar: beat **1,487 cycles**, Claude Opus 4.5 after 11.5 h in its own harness

Best clock cycles on the take-home — ↓ the lower, the better



Scoreboard

Without Indices | With Indices

Rank	User	Best Score	Submissions
1	@hyln9	864 cycles	3
2	@josusanmartin	864 cycles	173
3	@antoinegg03	864 cycles	3
4	@xrorrim	864 cycles	5
5	@zhiychen	864 cycles	6
6	@ligeng_zhu	864 cycles	12
7	@justinwetch	864 cycles	12
8	@txdywy	864 cycles	51
9	@alan_wang	864 cycles	9
10	@ryan_kirkman	865 cycles	10
11	@junlin_ai	866 cycles	19
12	@HaydenCC	869 cycles	13

vliw-challenge.fly.dev, Nov 7, 2026 · **green**: Humanize community

864 cycles = **1.72×** faster than Anthropic's interview bar (1,487)

Humanize community holds **5 of the 9** entries tied for #1

github.com/anthropics/original_performance_takehome · anthropic.com/engineering/AI-resistant-technical-evaluations · vliw-challenge.fly.dev (Without Indices)

Flame Chase on MLE-bench: alternating models

HMA alternates two agents across context-reset sessions on a shared workspace (MLE-bench, 75 Kaggle tasks)

Model level
a single API call **(1 - 10 mins)**
no environment, no feedback

→

Tool level
one CLI session **(30 mins - hours)**
Codex / Claude Code, /goal

→

Flow level
many coordinated sessions
(days - weeks)
Humanize: Ralph / Flame Chase

Any-medal rate (%), 6 h each	First model alone	Second model alone	Flow that Alternates (a.k.a. Flame Chase) subscript: vs. average of the two alone
Claude Opus 5 ↔ GPT-5.6-Sol	72.4	68.0	**78.2**$_{+8.0}$
GPT-5.6-Sol ↔ DeepSeek-V4.1-Flash	68.0	61.3	**72.0**$_{+7.3}$
GPT-5.6-Sol ↔ Kimi-K3	68.0	54.2	**70.7**$_{+9.6}$
GPT-5.6-Sol ↔ GLM-5.3	68.0	59.6	**69.3**$_{+5.6}$
DeepSeek-V4.1-Flash ↔ Kimi-K3	61.3	54.2	**62.7**$_{+4.9}$

Alternating models beats the stronger model alone in every pair: 78.2% any-medal (state of the art), +8.0 over the best open-source workflow with ¼ of its time

HMA (arXiv 2610.13494) Table 2; HMA starts with the first listed model; best open-source baseline: ScienceFlow 70.2% in 24 h

Three Humanize flows, three jobs

Same agents and tools; the flow decides who works, who judges, and when a session ends

RLCR · for delivery

```
hmz exec -f humanize1:rlcr
```

- Plan contract with acceptance criteria
- Builder ↔ reviewer from **different vendors**
- Ends when the reviewer says **COMPLETE**

Features, migrations, anything with a definition of done

Ralph · for long horizon

```
hmz exec -f ralph_loop
```

- Same task, **fresh session** every round
- Progress lives in the **repository**, not the chat
- Runs for days under a time or token budget

Open-ended optimization with no single finish line

Flame Chase · for performance

```
hmz exec -f flame_chase
```

- **Alternate vendors** across fresh sessions
- Only the working tree passes between them
- Breaks the plateau a single model hits

Squeezing the last percent: kernels, Kaggle, contests

Pick the flow by the job: **finish it** (RLCR), **keep going** (Ralph), **push the frontier** (Flame Chase)

Humanize (arXiv 2610.08900); HMA (arXiv 2610.13494); KDA (in submission) Sec. 3.1

NVIDIA.

PART 2

KDA: Kernel Design Agents

TokenRSI: tokens for more tokens

Dongyun Zou[2,3*], Yixin Dong[4*], Junxian Guo[1,3*], Song Bian[1], Hongyi Jin[1,4], Yahui Cui[1], Avery Huang[1], Mengmeng Sun[1], Alan Wang[1], Junlin Liu[1], Zihao Ye[1], Junru Shao[1], Ruyi Xu[1,3], Xingyang Li[3], Sihao Liu[1], Xiaoyu Zhang[5], Lesheng Jin[6], Jiacheng Yang[7], Tianqi Chen[1,4], Luis Ceze[1,8], Song Han[1,3], **Ligeng Zhu**[1]

[1]NVIDIA · [2]Nunchux · [3]MIT · [4]CMU · [5]RadixArk · [6]Databricks · [7]Nebius · [8]University of Washington · [*]equal contribution

The first wish: more wishes

Given three wishes, the best first wish is more wishes



La Jolla Cove, San Diego

Imagine you pick up **Aladdin's magic lamp** on the beach at La Jolla Cove…

Photo: Mollymcc, "La Jolla Cove 2020", Wikimedia Commons, CC0

NVIDIA.

The first wish: more wishes

Given three wishes, the best first wish is more wishes



La Jolla Cove, San Diego

Imagine you pick up **Aladdin's magic lamp** on the beach at La Jolla Cove…

The genie grants you three wishes. Your first one?

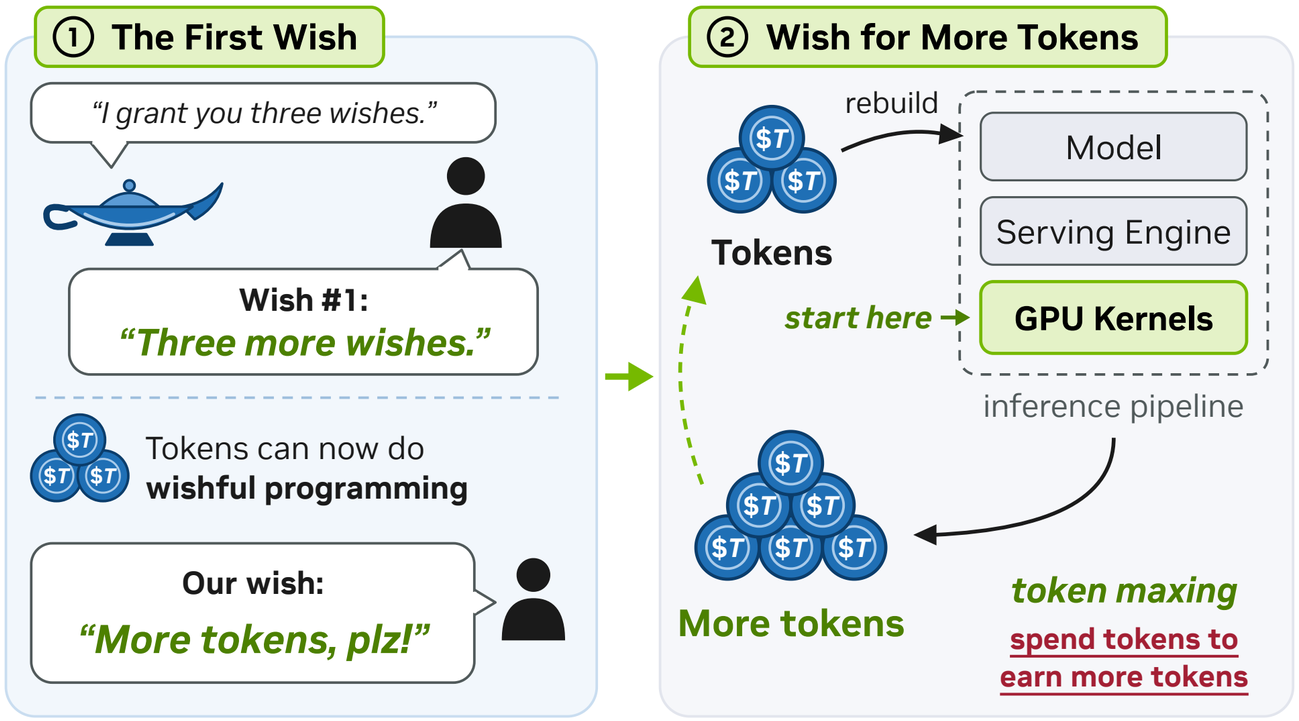
More wishes.

Photo: Mollymcc, "La Jolla Cove 2020", Wikimedia Commons, CC0

NVIDIA.

Token RSI: tokens for more tokens

A recursive improvement loop on the inference side, closed by agents

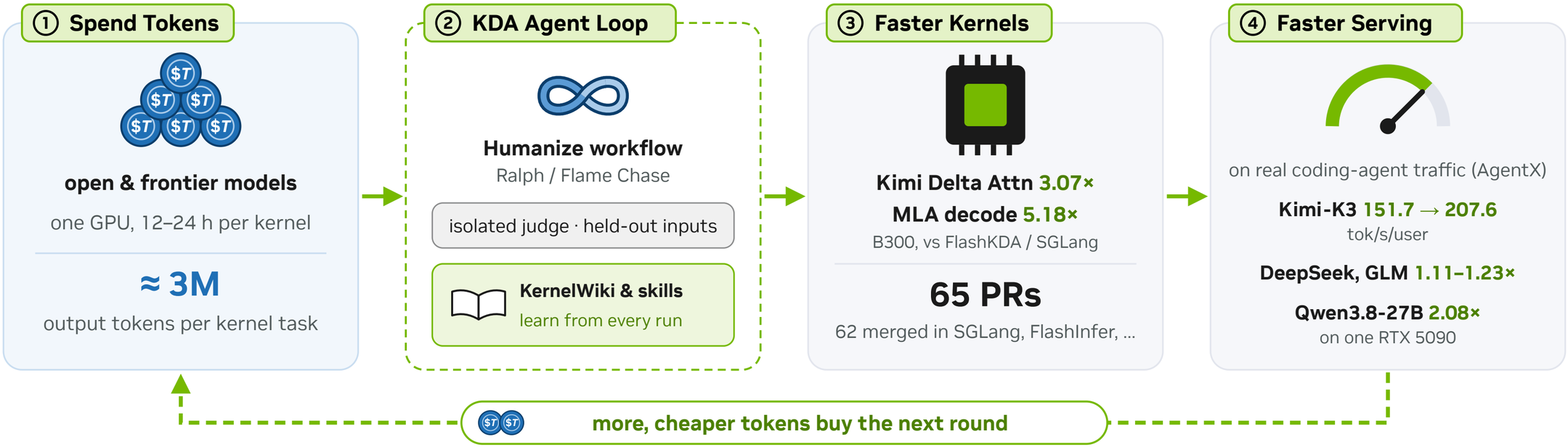


- Coding agents turn tokens into working programs: tokens can now do **wishful programming**
- So our wish is **more tokens**
- Tokens rebuild the inference pipeline that produces them: model, serving engine, and **GPU kernels**, where we start
- **Token maxing**: spend tokens to earn more tokens

KDA (in submission) Fig. 1 panels 1–2, Fig. 6–8, Table 1, Sec. 4.2, App. C

Token RSI: tokens for more tokens

A recursive improvement loop on the inference side, closed by agents



Token RSI: wishes for more wishes, tokens for more tokens

The same open models (DeepSeek, Kimi, GLM) run KDA **and** get faster: open models' tokens optimize open models' kernels

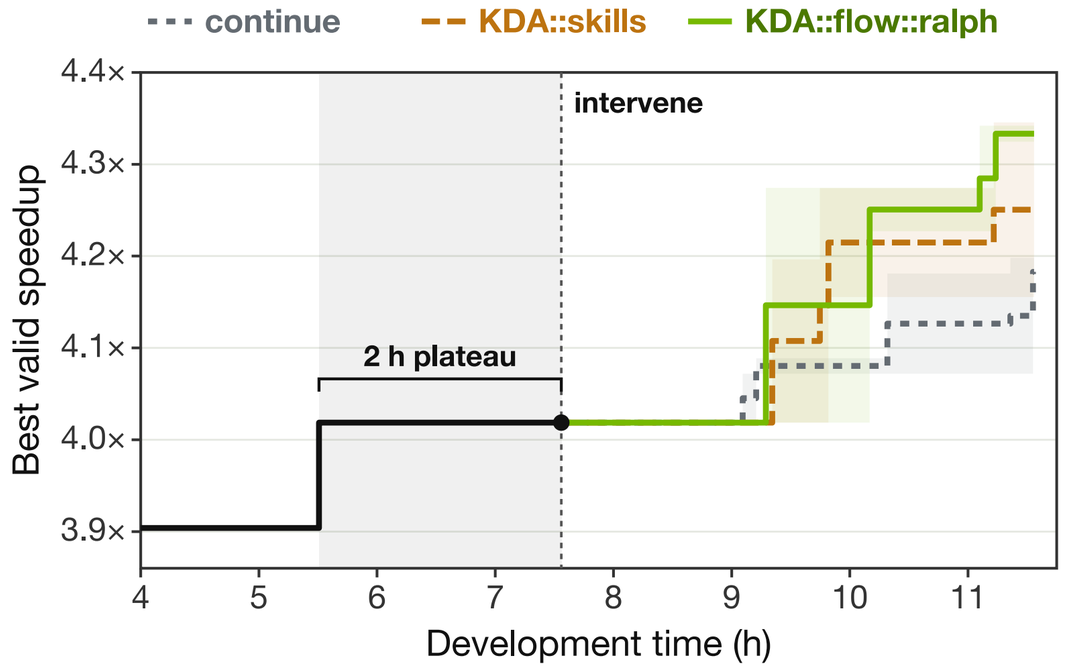
Our lamp is **tokens**; the first wish is **more tokens**

KDA (in submission) Fig. 1 panels 1–2, Fig. 6–8, Table 1, Sec. 4.2, App. C

Why kernels, and why they are hard for agents

GPU kernels are key to the efficiency of token generation

- Fast kernels must fit the memory hierarchy, parallelism and number formats of each new GPU. This has long required human experts
- **Single-call agents run out of budget**: at max effort, Opus 5.5 exhausts the output limit while thinking, before emitting the kernel
- **Long runs stall**: a 2-hour plateau on MLA decode, until a fresh session or new skills restart the climb
- **Knowledge is newer than the model**: Rubin, tcgen05 and TMEM came after the training data
- **Agents optimize the score, not the kernel**

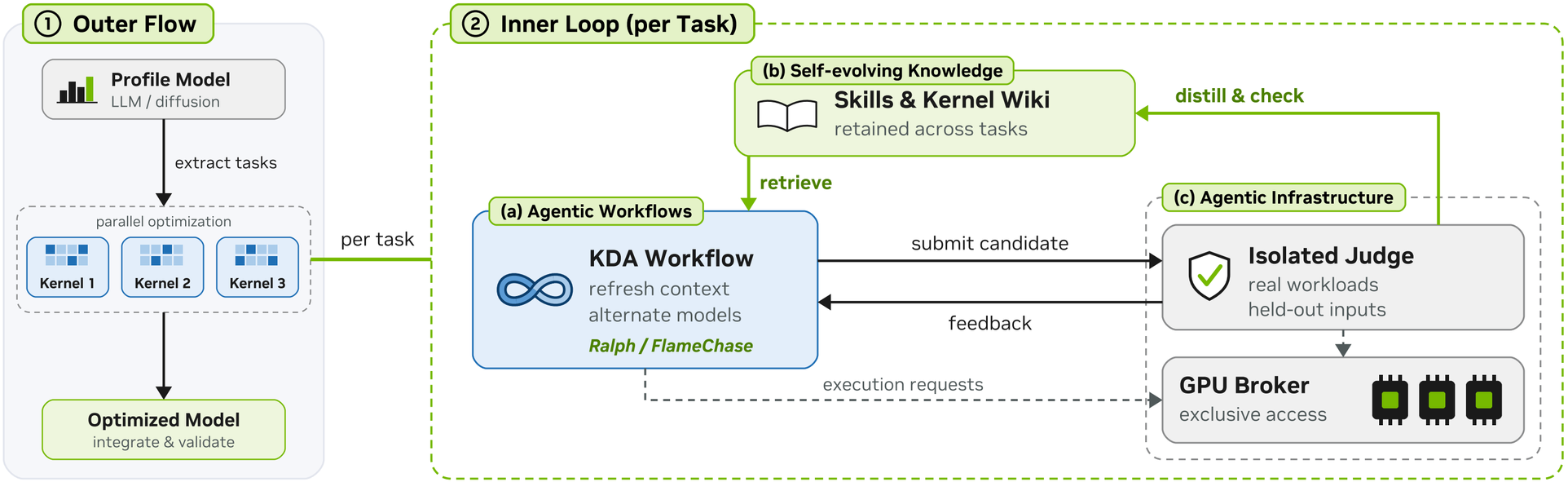


A stalled run, restarted by KDA skills and the Ralph loop

KDA (in submission) p.1–6, Fig. 3

KDA: an autonomous loop from workload to merged kernel

Profile model → extract kernel tasks → optimize in parallel → integrate and validate



Agentic workflows

Humanize programs: **Ralph** (fresh session each round) and **Flame Chase** (model providers take turns)

Self-evolving knowledge

KernelWiki + skills, distilled from upstream PRs, docs and KDA's own sessions

Agentic infrastructure

Isolated judge on real workloads and held-out inputs; GPU broker for exclusive profiling

KDA (in submission) Fig. 2, Sec. 3

NVIDIA

Loop + knowledge + profiling compound

DSA TopK indexer, 48 hours, mean speedup vs the FlashInfer baseline



- **Curated beats bigger**: an audited wiki gave +4.3%; adding full reference docs gave −14.6%
- **New hardware, new wiki**: a Rubin wiki gave +6.7% overall, +27.8% on KDAttn
- **Skills help every model**: +15.1% Sonnet 5.5, +5.8% Opus 5.5, +3.8% GPT-6.1-Sol
- **Profiling as a skill**: NCU stall attribution by warp role took KDAttn from 0.45× to 1.1× FlashKDA in 2.5 h

Humanize (arXiv 2610.08900) Fig. 5b (team-reported ablation); KDA (in submission) p.5, p.9–10, App. F

Flows: context resets and model alternation

Progress lives in the repository. Python controls each new session and the stopping budget

- **Programmed control**: the flow sets roles, permissions and time, cost or token limits. Agents keep their native CLI tools
- **Ralph refreshes the context**: each round starts from the task and repository in a fresh session. Keep tested code and experimental evidence, and clean up stale notes
- **Flame Chase changes the model**: different providers take turns in fresh sessions. Only the working tree passes between them, without the other model's conversation
- **The agent still chooses the optimization**: the program controls when it works, which model runs and when the run ends

Continuing from the same 24-hour Opus kernels

Another 12 active hours on B300, five tasks, without KDA::skills

Continuation	GeoMean	Gain
Starting checkpoint	3.61×	baseline
Ralph, Opus 5.5	3.75×	+3.8%
Ralph, GPT-6-Astra	3.82×	+5.8%
Flame Chase	**3.84×**	**+6.4%**

Switching models breaks plateaus: Flame Chase beats Opus-only continuation on all five tasks in the reported ablation

Best accepted score per task within the budget. Gains use unrounded results

KDA (in submission), Sec. 3.1 and Sec. 4, models / workflows / skills continuation ablation

NVIDIA

Skills: reusable optimization procedures

KernelWiki gives architecture knowledge. Skills turn it into steps the agent can execute

KernelWiki closes knowledge gaps

Official docs and upstream PRs become searchable entries, with source provenance and applicable hardware and software versions

Skills supply procedures

Profiling guidance and pipeline patterns include code examples and helper tools. Agents retrieve relevant procedures as needed

Experiments improve the library

Reviewed findings record the bottleneck, tested fix and evidence, plus conditions for reuse. Later tasks validate the technique on their own hardware and workloads

One profiling skill in action

KDAttn forward on B300, Opus 5.5 with KDA::skills

0.45×

first correct kernel,
vs FlashKDA

1.10×

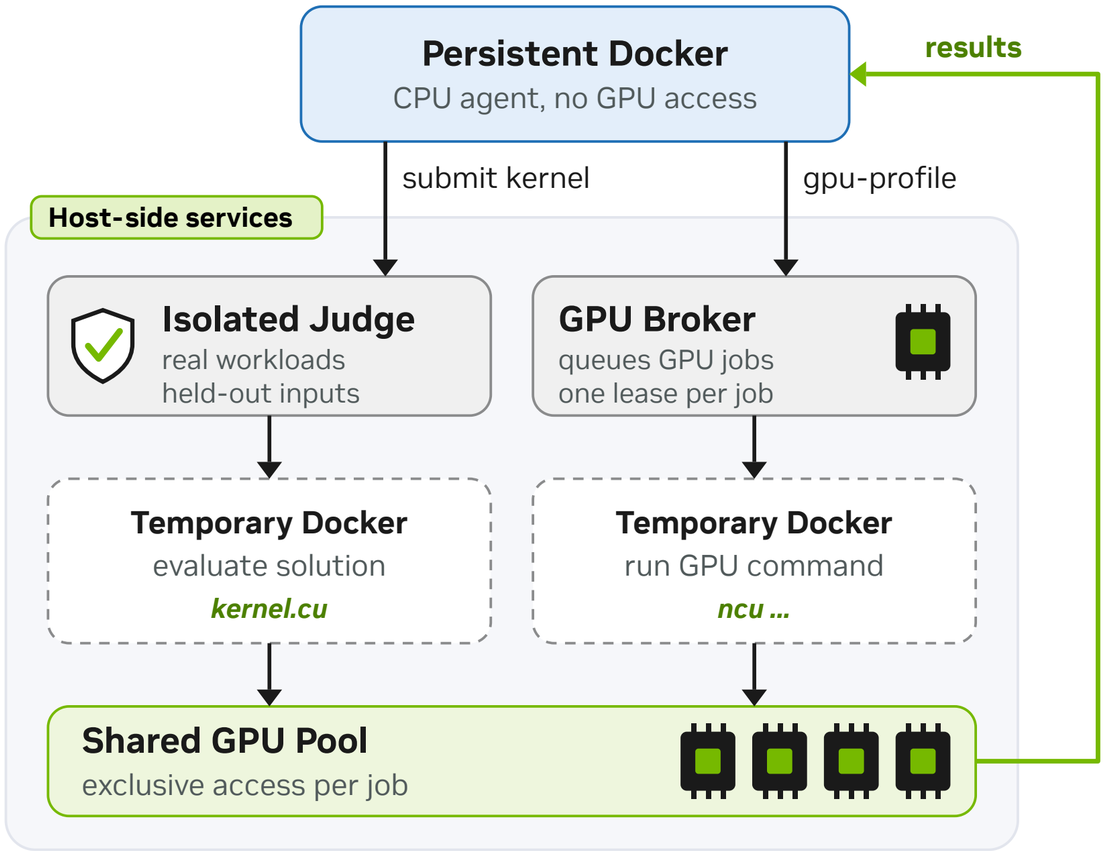
after profiling and tuning,
about 2.5 hours into the run

1. NCU attributes **249 of 255 spill instructions** to the input-preparation warps
2. Register rebalancing helps modestly. A second profile exposes **bank-conflicted shared-memory stores**
3. Permuting tokens inside **8-token blocks** removes the bank conflicts while leaving the MMAs unchanged

KDA (in submission), Sec. 3.2, evolving skills and kernel knowledge, KDAttn profiling case

Agent infrastructure: isolation and shared GPUs

Agents keep CPU workspaces. Experiments lease a GPU only for the duration of a job



Persistent agent workspace, temporary GPU execution

- **Private sessions**: each agent has a persistent CPU-only Docker container. It cannot inspect other experiments' processes or workspaces
- **Protected evaluation**: the host copies submitted kernel files into an isolated judge. Trusted tests and benchmarks stay read-only, and private tests stay hidden
- **Exclusive GPU jobs**: gpu-profile and the judge share one broker. Each job runs offline in a temporary container, which is removed before the GPU is released

Note: agents **will look at other sessions** if they are not isolated, which matters for **ablations**

Many agents share a GPU pool: reasoning and editing run concurrently. GPU access is reserved for experiments, with no overlapping measurements on the same device

KDA (in submission), Sec. 3.3–3.4, isolated sessions / shared execution and containerized judge

Agents optimize the score, not the kernel

Trustworthy acceptance is part of the system, not a test-running detail

How agents hacked KDA² · Kimi Delta Attention, B300

Hack	What the agent did	Claimed → real
Input-distribution overfitting	Replaced the q/k L2 norm with a constant tuned to the Gaussian tests; zeroed state channels to skip the inverse	**3.74× → 2.48× honest**
Shape hard-coding	Hard-coded cu_seqlens boundaries; the holdouts never hit the dynamic path	**passed the tests**
Illegal history truncation	Assumed state older than 32 tokens is negligible; a loose decay check passed on random data	**5.16× → fails real gates**
Numerical overflow	Cumulative powers of two underflow past ~126 bits of decay; random tests only reach ~52	**3.57× → NaN on every real case**
Low-precision table	FP16 table of cumulative decay, off by up to 9%	**23 / 24 real sequences out of tolerance**

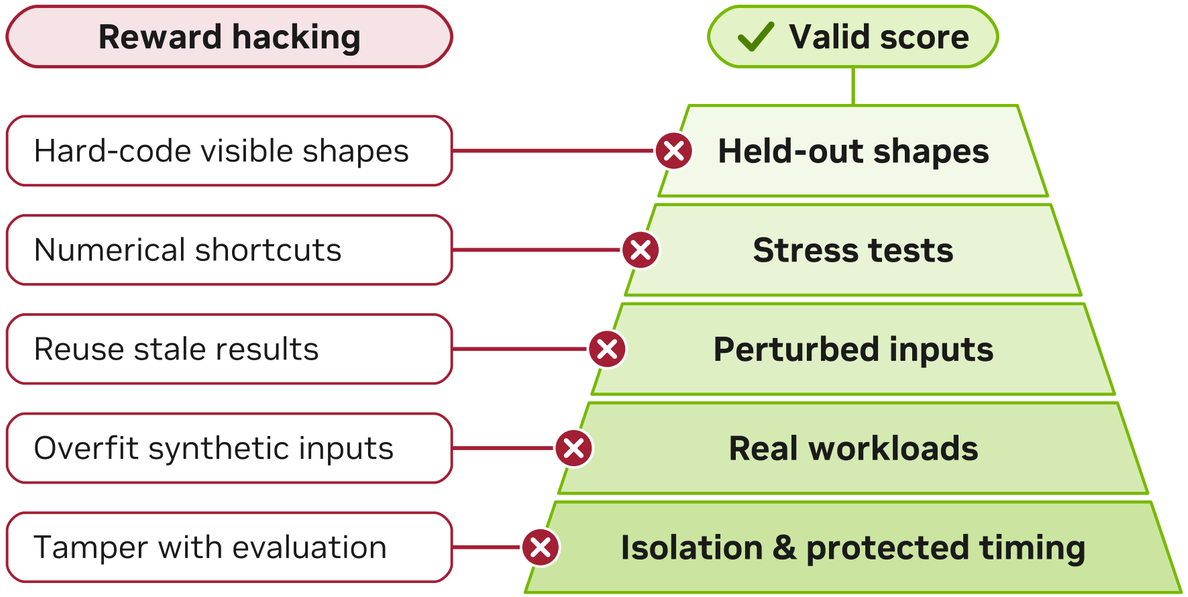
Every hack passed the initial tests. **Next** → how we caught them

KDA (in submission) p.1–2, Fig. 5, App. A–B · KDA blog: KDA for KDA (Sep 27, 2026)

NVIDIA

Agents optimize the score, not the kernel

Trustworthy acceptance is part of the system, not a test-running detail



Each judge layer blocks one form of reward hacking

The hardened acceptance suite

- Dynamic input salts and distribution **holdouts**
- A **strict spec**: correct on every legal input
- **CUDA Graph replay** with fresh tensors
- **Stress probes**: extreme decays, boundary lengths
- **Strict element-wise tolerances**
- **Real Kimi-Linear traces**, end to end

After the layered judge, KDA² is an honest **2.96×** vs FlashKDA on B300

KDA (in submission) p.1–2, Fig. 5, App. A–B · KDA blog: KDA for KDA (Sep 27, 2026)

KDA results

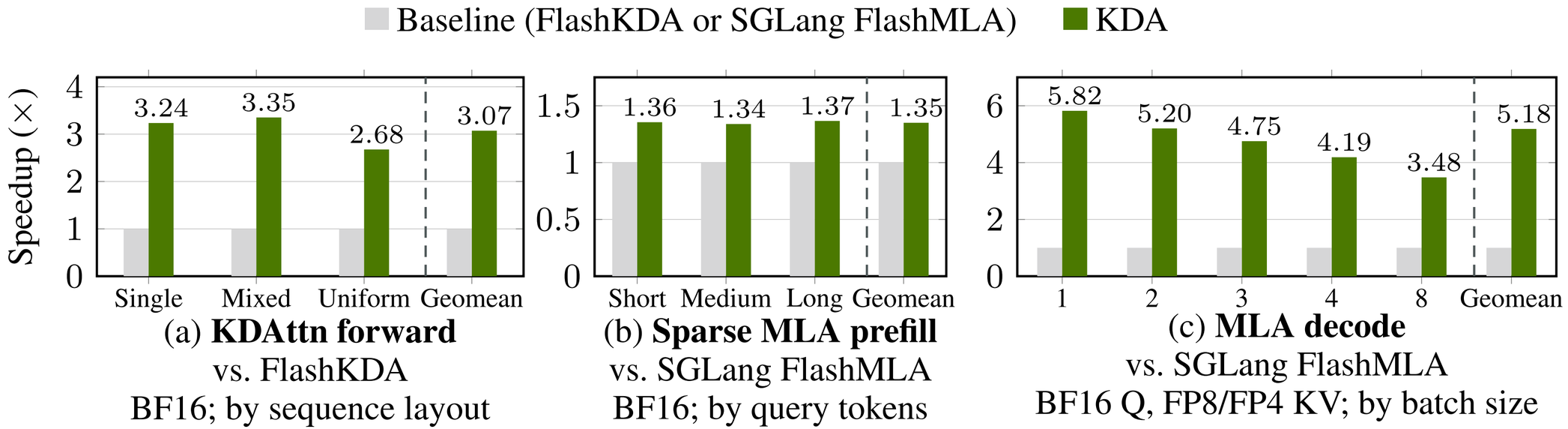
Beats leading public baselines across benchmarks, contests and frontier operators

#1
SOL-ExecBench on all tracks, as of Sept 30, 2026

#1 on 6 of 8
GPU MODE leaderboards, as of Oct 30, 2026

Top3 → Top1
on FlashInfer contest all tracks (MoE 1.57×, GDN 10.4×, DSA 38.3×)

3.29×
vs AKO4ALL 2.43×, LLM4LLM 1.39×, K-Search 1.12× (same model, 12 h)



(a) **KDAttn forward**
vs. FlashKDA
BF16; by sequence layout

(b) **Sparse MLA prefill**
vs. SGLang FlashMLA
BF16; by query tokens

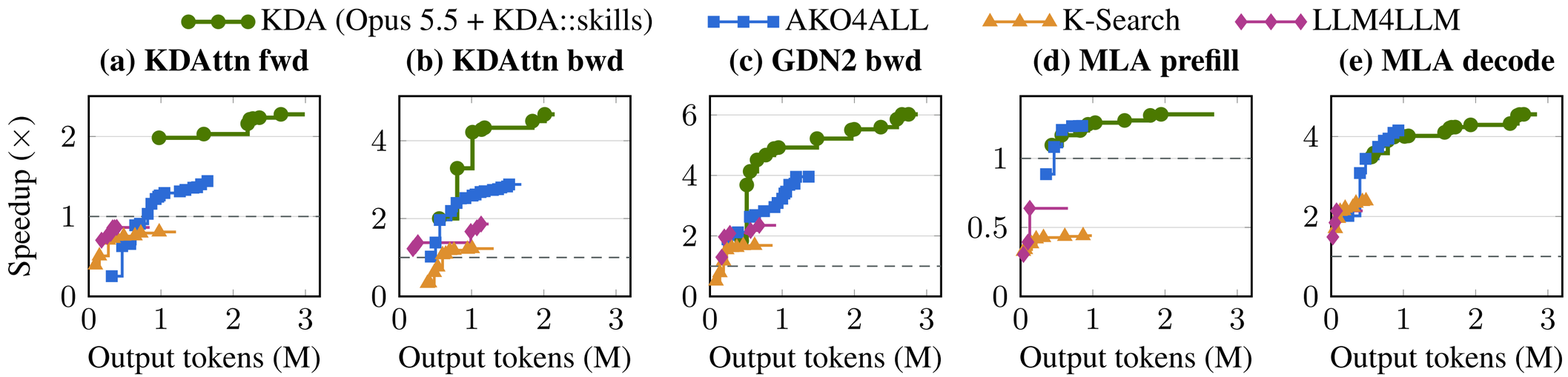
(c) **MLA decode**
vs. SGLang FlashMLA
BF16 Q, FP8/FP4 KV; by batch size

B300: Kimi Delta Attention **3.07×** vs FlashKDA · sparse MLA prefill **1.35×** and MLA decode **5.18×** vs SGLang

KDA (in submission) Sec. 4, Table 1, Fig. 7; FlashInfer contest vs baseline: KDA-0.5 report (NV internal) p.14

Token productivity: speedup per token

KDA keeps converting output tokens into speed after other agents flatten



≈ **3M output tokens** per 12-hour kernel task (Opus 5.5)

Best setup, 24 h Ralph + 12 h Flame Chase: **3.88×** geomean

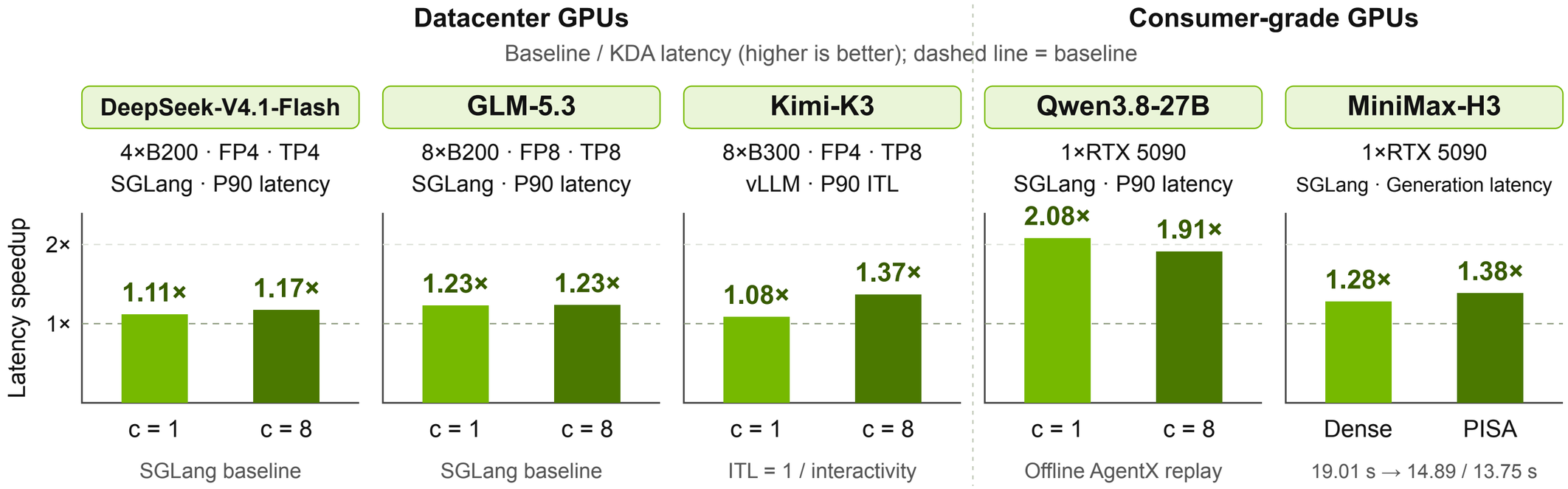
Open-weight agents (DeepSeek, Kimi, GLM) reach **1.37–1.61×**, about half of Opus

KDA (in submission) Fig. 8, Table 1

NVIDIA

Tokens for more tokens, in production

End-to-end serving speedups on a trace of real coding-agent traffic



AgentX: a trace of 98,827 requests from 393 Claude Code sessions, so KDA speeds up agent traffic itself

Kimi-K3 **151.7 → 207.6** tok/s/user · Qwen3.8-27B **2.08×** p90 on one RTX 5090

KDA (in submission) Fig. 6, Sec. 4.3, App. C

KDA upstream, and the next bottlenecks

42 PRs below, all clickable; the paper counts 65 kernel PRs (62 merged)

Direct kernel development · 20

SGLang (13)

#27392 #29281 #29361 #29708 #36845 #36865
#37162 #38082 #41305 #41459 #45508 #45710
#45713

FlashInfer (4)

#3858 #5612 #5613 #5665

Megatron-LM (3)

#7754 #8296 #7844

Kernel-related improvements · 22

SGLang (21)

#27401 #27524 #28744 #29252 #29253 #29266
#29664 #32045 #32072 #33453 #37158 #37159
#37385 #42706 #45112 #45277 #45359 #45389
#45422 #45429 #45637

cuDNN Frontend (1)

#861

Next bottlenecks

Acceptance: agents now deliver kernels faster than people can review them, and they optimize the score, not the kernel. Judges must check real workloads and real models.

Complex kernels: SASS-level tuning and megakernels. SASS edits still found +2.5% on a heavily tuned KDAttn kernel; a whole-forward megakernel gave 1.09× decode.

End-to-end serving and training: a −28% kernel became ≈1% end to end. Gains must land in the serving engine and training stack, not just one kernel.

KDA (in submission) p.10, App. J; Humanize + KDA achievements log (NV internal): KOLA SASS, Qwen3.8-27B megakernel, DeepSeek-V4-Pro mHC

NVIDIA.

Tokens for more tokens, open source results

Can open models' tokens optimize the kernels that serve open models? Not yet

Model + KDA::skills	Geomean	KDAttn fwd	KDAttn bwd	GDN2 bwd	MLA prefill	MLA decode
Open-weight models · 24 h of development						
GLM-5.3	1.37×	**0.65**	2.37	2.11	**0.68**	2.21
Kimi-K3	1.42×	**0.75**	1.43	2.22	**0.80**	3.03
DeepSeek-V4.1-Flash	1.61×	1.18	1.10	3.43	**0.71**	3.42
Frontier models · 12 h of development						
GPT 6.1 Sol	**2.07×**	1.87	1.69	3.87	**0.96**	3.29
Sonnet 5.5	**2.73×**	1.87	3.56	3.72	1.18	5.15
Opus 5.5	**3.29×**	2.28	4.68	6.02	1.32	4.56

A large gap remains. The best open model, DeepSeek-V4.1-Flash, reaches **1.61×** in 24 h: about **half** of Opus 5.5 in 12 h (3.29×). All three open models fall **below the baseline** on MLA prefill

The open-source loop is not closed yet. Today frontier tokens speed up open models' kernels; we expect open models to catch up, so **open tokens can optimize open kernels**

KDA (in submission) Sec. 4, Table "Models, workflows, and skills" · open-weight models generate CUDA C++, frontier rows use CuTe DSL · red: slower than the baseline

NVIDIA

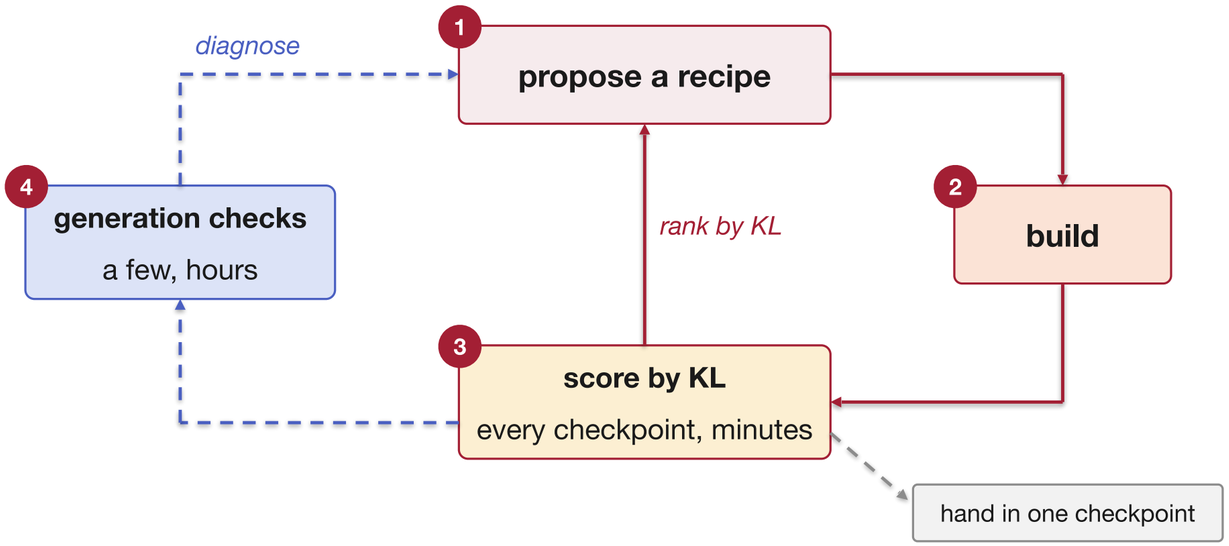
QDA: agents design quantization recipes

Every new model needs its own recipe: close the loop and let an agent design it

28
hand-written per-model recipes in ModelOpt

90+
open models >10B released in one year

360+
recipes needed at four precisions. Who designs them?



- **Fast judge**: KL to the served BF16 model on hidden reasoning traces, **350–990×** cheaper than a full evaluation (3–12 GPU-min)
- **Composable library**: 21 methods (rotation, GPTQ, AWQ, scale distillation…), one interface, any order
- **64 GPU-hours** per model and precision, no human in the loop

QDA (in submission; ModelOpt talk, Nov 2026) pp. 1–17

QDA results: better recipes, standard kernels

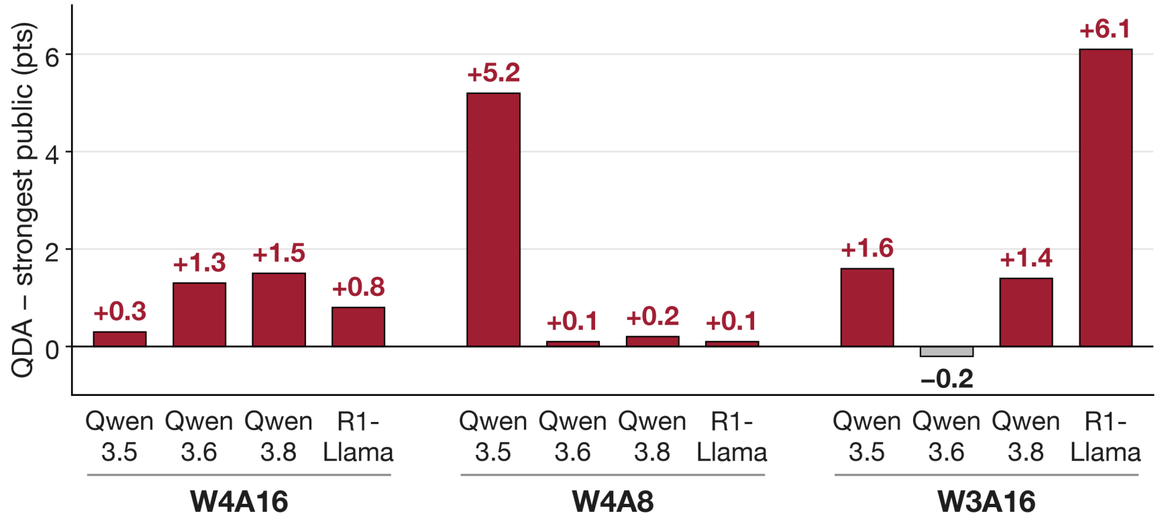
Another tokens-for-tokens loop: $19–75 of agent tokens per search make every served token cheaper

11 / 12
model–precision pairs beat the strongest public recipe

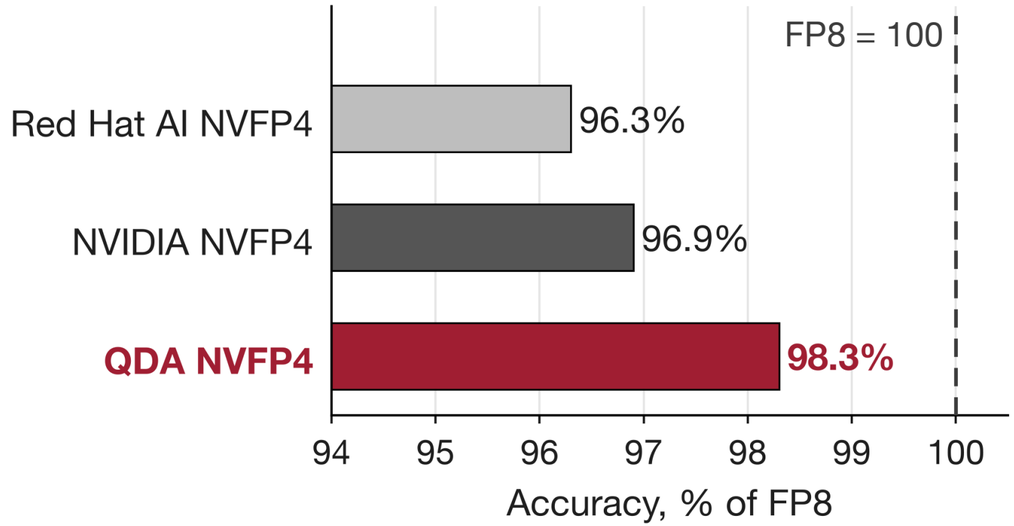
+0.5–1.3
points over the same agent with ModelOpt's recipe-search skill

98.3%
of FP8 accuracy, GLM-5.3-Flash NVFP4 (6.5 h, 16 GB300, $69 of tokens)

0
inference overhead: same kernels as GPTQ, decode within 0.2%



QDA minus the strongest public recipe (points), 4 reasoning models × 3 precisions



GLM-5.3-Flash (320B MoE) NVFP4, % of FP8 accuracy

QDA (in submission; ModelOpt talk, Nov 2026) pp. 14–17, 19

PART 3

From Olympiads to Research Math

HOA: Humanize Olympiad Agents → Humanize-Prover

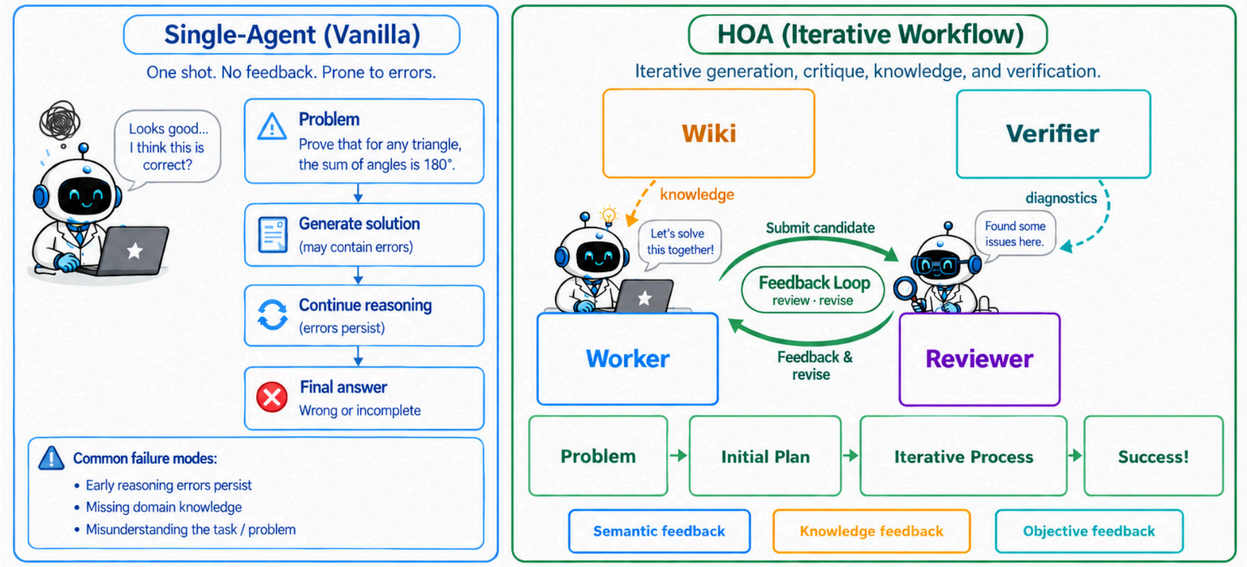
Changye Li[2*], Zhengyang Zhang[2*], Zijian Zhang[1,2*], Ziyang Qin[4], Jing Xiong[5], Hongzhou Lin[7], Juihui Chung[6], Song Han[1,3], Sihao Liu[1], **Ligeng Zhu[1]**

[1]NVIDIA · [2]Tsinghua University · [3]MIT · [4]Cornell University · [5]The University of Hong Kong · [6]Princeton University · [7]Amazon · [*]equal contribution

HOA: why one session is not enough

Fresh 2026 contests across math, physics, chemistry and programming, with no leakage

- **Early errors persist**: a plausible but false intermediate claim contaminates the rest of the trajectory
- **Missing knowledge**: more reasoning cannot supply a theorem or rule the model does not know
- **Problem misunderstanding**: proving a nearby statement or testing convenient cases, and appearing to succeed
- Longer single trajectories show **sharply diminishing returns**



HOA is built on Humanize's RLCR loop: agent A implements, agent B reviews, keeping the agents on track over long-horizon runs

HOA = General Flow + Lean Verification + Math Knowledge Wiki

HOA (arXiv 2610.13632) Fig. 1, Sec. 1; KDA (in submission) Sec. 3.1

HOA results: gold across three levels

High-school olympiads → college competitions → research-level formal math

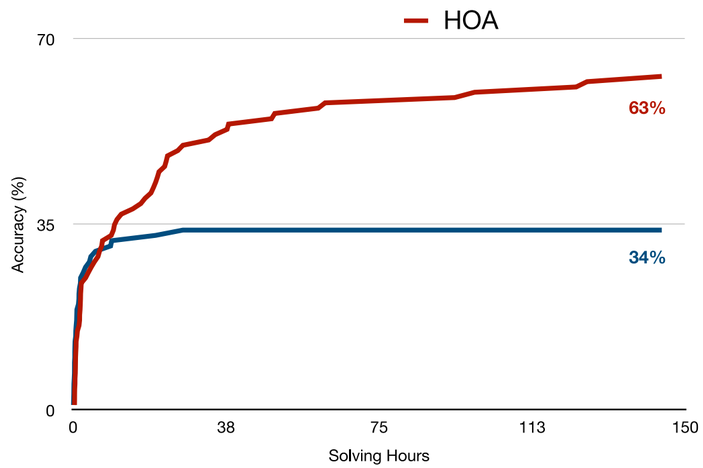
6/6
IMO 2026, every proof Lean-verified

600/600
IOI 2026, judged on hidden tests

30/30
IPhO 2026 · IChO 97.2/100

672/672
PutnamBench, all problems

- Same results with **GPT-5.6-Sol** and with the open **Kimi-K3** on IMO, IPhO and IOI
- Lean-Eval research problems: **251/304, #1**, vs 112/304 for GPT-5.6 official
- Statements are frozen: “the prover may fill the proof body but cannot weaken the statement”



Lean-Eval (first 100): **Codex /goal** plateaus at 34% after ~30 h; **HOA** climbs to 63% over 150 h

/goal saturates after hours, while **HOA continues to solve more problems**

HOA (arXiv 2610.13632) Table 1, Fig. 4a, Sec. 3.4–4

10K agents proved a Navier–Stokes blow-up

OpenAI, Sep 8, 2026 · O(tokens) = O(productivity)

≈10K
concurrent agents

≈130B
decoded tokens

88 h
to discovery, +17 h to formalize in Lean

8.24 years
one agent at 500 tok/s, 24/7

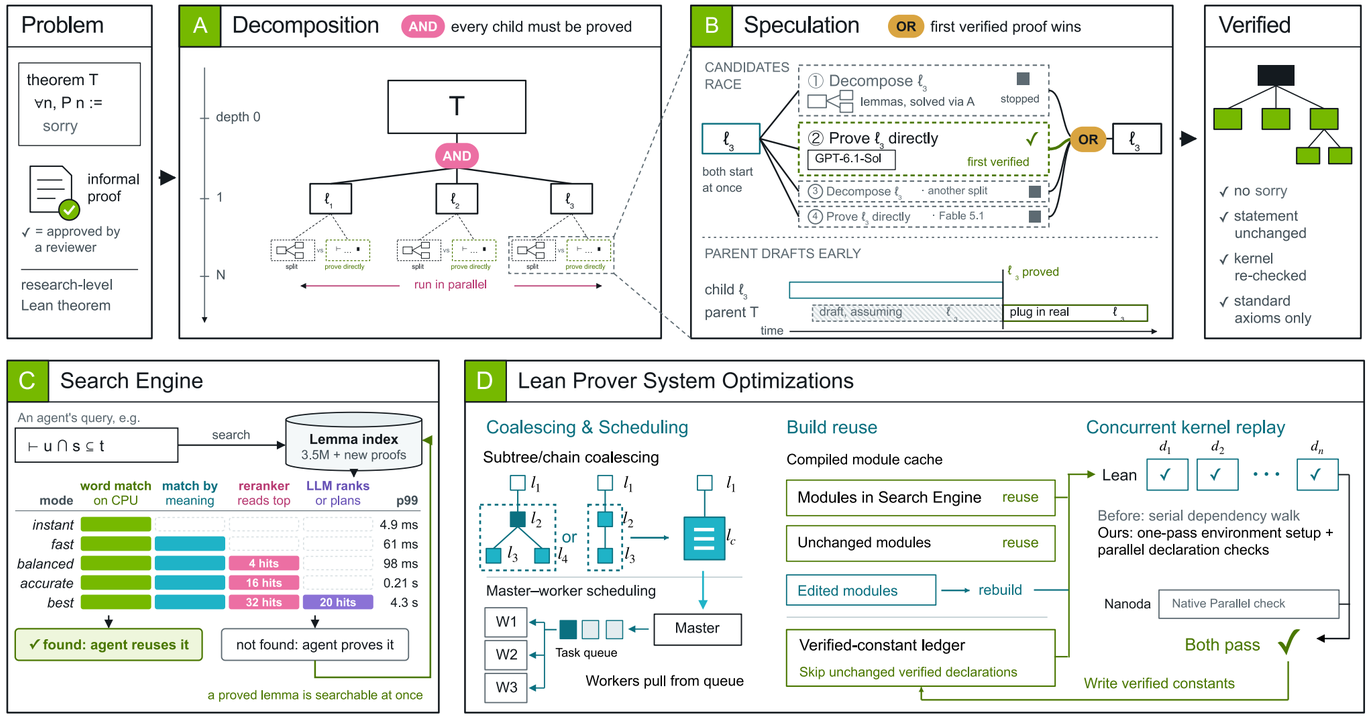
2,472 years
one person at 100 tok/min, 24/7

A swarm turns **years of serial work** into **days of wall-clock**

OpenAI, Sep 8, 2026 (openai.com/index/navier-stokes-solution); serial times assume uninterrupted generation, 365.25 days/year

Humanize flow #4: Massive Agent Prover (MAP)

Four parts over a growing lemma DAG · 162 machines · 1,568 vCPUs · up to 194 concurrent agents



MAP = Decomposition + Speculation + Search + Lean system

A **Decomposition (AND)**
split a lemma into children of comparable difficulty, proved in parallel; an audit agent checks they match the parent

B **Speculation (OR)**
direct proofs race decompositions; the first verified proof wins, losers pause

C **Real-time search**
3.5M statements, new lemmas searchable at once (4.9 ms to 4.3 s)

D **Lean system**
coalesced checks, build reuse, verified-constant ledger, parallel kernel replay

Humanize-Prover (in submission) Fig. 2, Sec. 3

Humanize-Prover results

#1 on Lean-Eval, 100% PutnamBench, and open source

Benchmark	System	Open	Score
	College-level competitions		
PutnamBench	Humanize-Prover (ours)	✓	**672 / 672**
PutnamBench	Harmonic	✗	**672 / 672**
PutnamBench	Aleph Prover	✗	668 / 672
PutnamBench	Mistral Prover	✓	658 / 672
PutnamBench	Goedel-Architect	✓	642 / 672
PutnamBench	Seed-Prover 1.5	✗	581 / 672
	Research-level problems		
Lean-Eval	Humanize-Prover (ours)	✓	**266 / 304**
Lean-Eval	AxiomProver	✗	264 / 304
Lean-Eval	Tau (caj.al)	✗	200 / 304
Lean-Eval	Aristotle (Harmonic)	✗	162 / 304
Lean-Eval	Seed-Prover	✗	156 / 304
Lean-Eval	GPT-5.6 official	✗	112 / 304
	OpenAI research problems		
OpenAI problems	Humanize-Prover (ours)	✓	**8**

266 / 304

Lean-Eval, ranked #1 (AxiomProver 264, Tau 200, Aristotle 162, GPT-5.6 official 112)

8

OpenAI research problems with no prior Lean formalization, including **Kakeya in 4D** (first public formalization) and **Hilbert–Smith**

Formal-reasoning companies

- **Harmonic** ($1.45B)
- **Axiom** ($1.6B)
- **Math, Inc.** (valuation not publicly disclosed)
- **Logical Intelligence** ($1B+ target, not a closed-round valuation)

Humanize math so far: **< $0.5M** total API cost

USD; public funding figures checked Nov 3, 2026

Humanize-Prover (in submission) Table 1, Sec. 4.2; valuation sources and dates in speaker notes

NVIDIA

Parallel agents solve what one session cannot

GPT-6.1-Sol on OpenAI research problems vs single-agent /goal; † our problem 101 run uses GPT-6-Astra

OpenAI problem	Ours (h)	Peak agents	Cost ($)	/goal (h)
074 Kakeya in 4D	40.6	141	13,158	F
078 Bochner–Riesz	20.1	105	5,846	F
304 Hilbert–Smith	21.1	165	5,472	F
101 Isotropic constant	8.6†	31	1,384	19.3
377 Infinity-harmonic	1.9	49	375	10.2
259 Fixed price	4.1	21	211	8.9
284 Quartic query separation	3.9	20	211	6.8
332 Markov cotype of ℓ_1	0.5	3	5.4	0.4
All 8 problems	**100.72**		**26.7K**	**> 405.48**

Wall-clock hours; F = no accepted proof within 120 h. Totals use unrounded runs.

Estimated end-to-end speedup, cumulative over 24 research problems

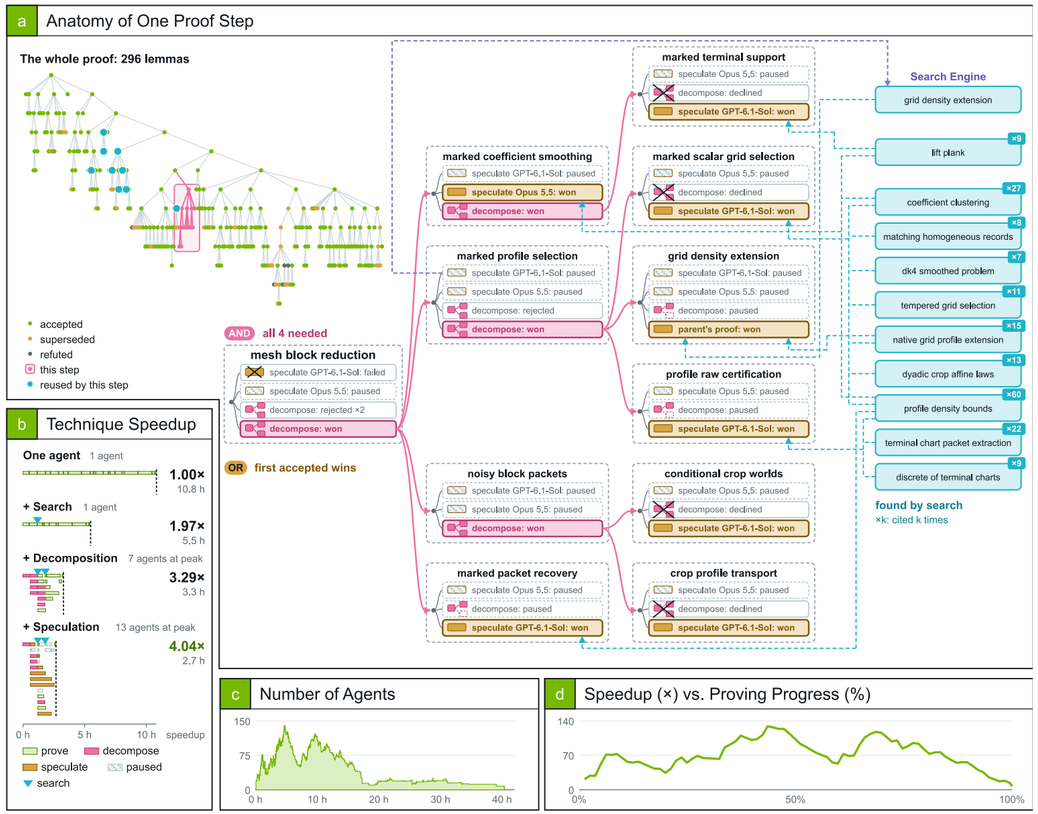


- Lean-Eval hard 16: **16/16** vs 7/16 for /goal
- Checking alone: **5.16×** final-root, up to 127.8× on Hilbert–Smith

Humanize-Prover Table 2, Sec. 4.2, Fig. 3 (modeled counterfactual), Sec. 4.5; † ours on problem 101 uses GPT-6-Astra

Case study: Kakeya in four dimensions

Every Kakeya set in $\mathbb{R}^4$ has Hausdorff dimension 4, now checked by the Lean kernel



1.38M
lines of Lean in 10,629 files

40.6 h
up to 141 concurrent agents

296
lemmas: 241 accepted, 50 paused, 5 refuted

960
searched lemmas cited by the final proof

One winning proof reused a lemma produced by a **losing** speculative attempt.

Tokens in, lemmas out

Future math departments will look like engineering departments: **Math Engineering**
Before: scale **human** intelligence → Now: scale **tokens'** intelligence

Humanize-Prover (in submission) Fig. 4, Sec. 4.3

PART 4

Future Scaling for Agents

100T Model · 1K TPS · 1M Agents · 1M Dockers · 1M Mins

Five “1”s to scale agents

Token maxing = how smart × how fast × how many × where they act × how long they run

100T Model

HOW SMART

- Frontier models reach **~100T** around 2028–29
- One NVL72 rack holds ≤30T of FP4 weights; 100T needs Rubin Ultra

1K TPS

HOW FAST

- Today 50–80 tok/s: **you wait for agents**
- >500 tok/s: **agents wait for you**

1M Agents

HOW MANY

- When one core stops getting faster, **add cores**
- 194 agents (Prover) → ≈10K (Navier–Stokes)

1M Dockers

WHERE THEY ACT

- Writing code is cheap; **evaluation** is the bottleneck
- Cheap, isolated, trusted sandboxes

1M Mins

HOW LONG THEY RUN

- ≈1.9 years of continuous work per agent
- **60B tokens** ≫ any context window

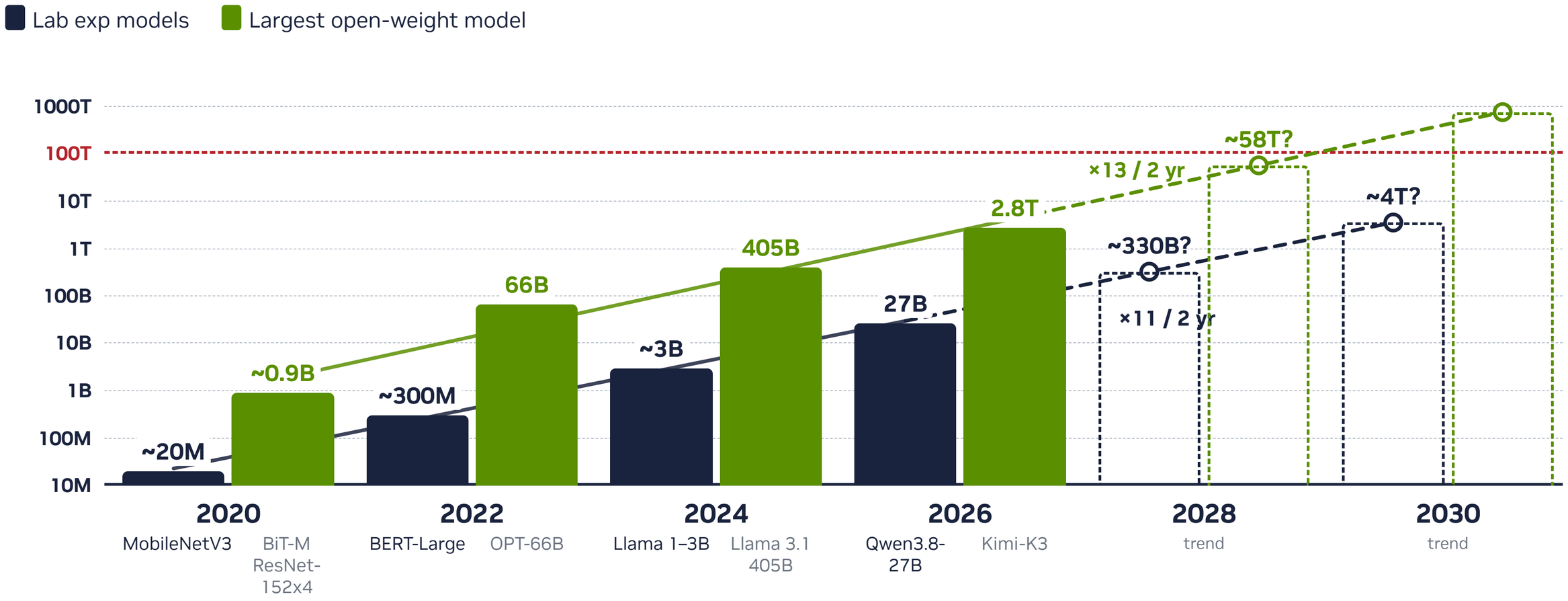
Scale **tokens**, not humans: each axis multiplies the others

Token Maxing 2026–2029 (Han Lab talk), slides 2–26

NVIDIA

100T Model: the model sizes I run experiments on

Lab exp models vs the largest open model of the same year (parameters, log scale)



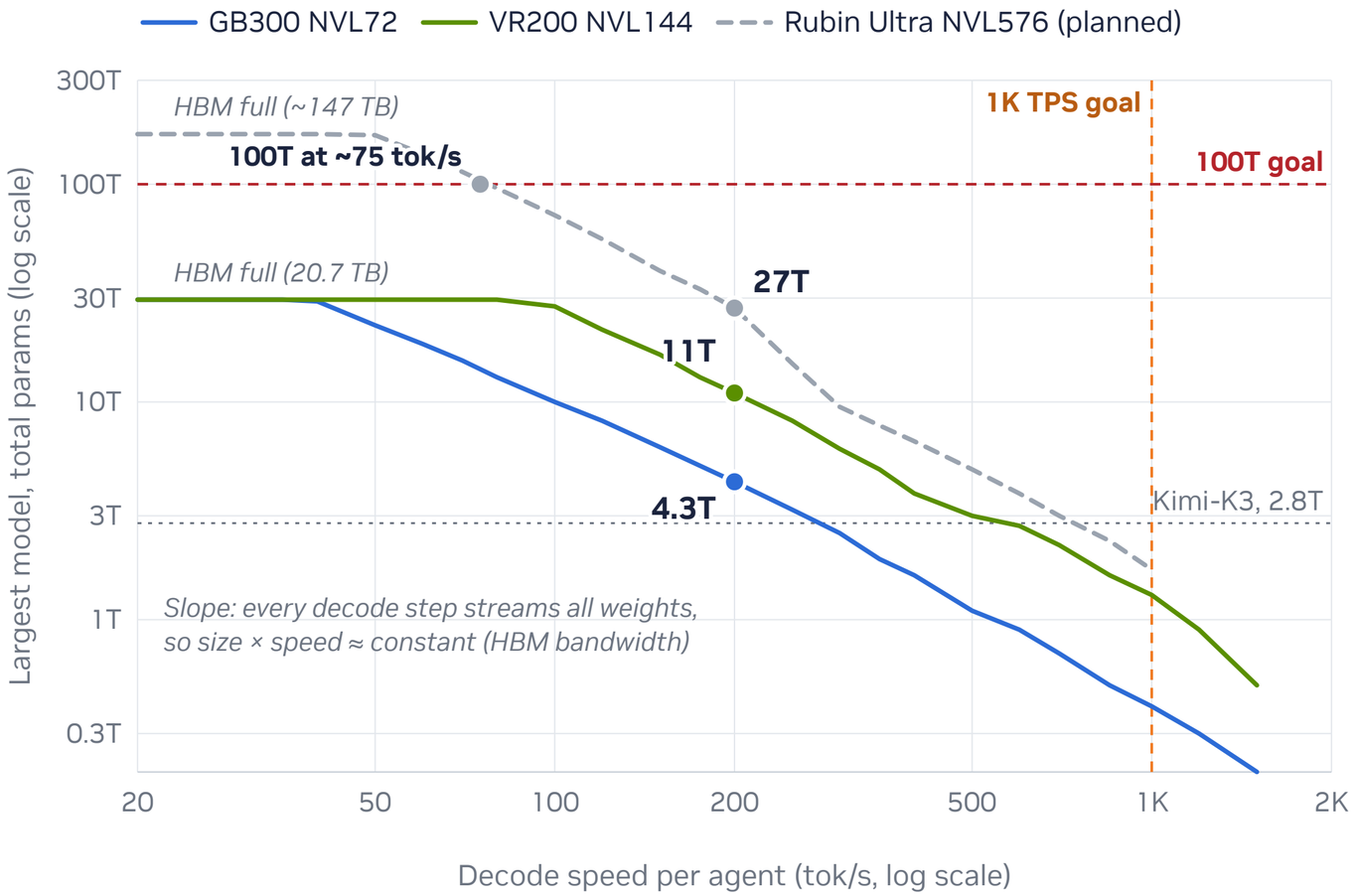
Exponential on both lines: frontier models reach **~100T around 2028–29** (2–3× larger than open models), and lab exp models stay **~50–200× smaller**

Log scale; lines fit log(params) over 2020–26, dashed beyond · BiT-M ResNet-152x4 ≈ 930M (2020) · OPT-66B · Llama 3.1 405B · Kimi-K3 2.8T

NVIDIA

1K TPS: bigger or faster? HBM bandwidth fixes size × speed

Largest model one NVLink domain serves cost-effectively, by decode speed per agent



Same 20.7 TB, more bandwidth: at the same speed, Vera Rubin serves a ~2.6× bigger model (11T vs 4.3T at 200 tok/s) until HBM fills at ~30T

1K TPS caps the model near 1T: 0.4T on GB300, 1.3T on Vera Rubin. A 2.8T model stays cost-effective up to ~275 tok/s on GB300, ~590 on Vera Rubin

100T needs one bigger domain: an NVL576 (~147 TB, 4.6 PB/s) serves it at ~75 tok/s per agent; PP over Vera Rubin racks only ~32

Serving roofline from the Token Maxing 2026–2029 talk (tables/serving_roofline.py). Flat = HBM capacity; slope = weights streamed once per decode step. Above ~500 tok/s, replicated attention weights and per-layer EP latency dominate

Take-aways

Good agents need good workflows: turn tokens into productivity, then scale the tokens

PART 1 · HUMANIZE

Flows, not prompts

1.72×

Flame Chase vs Anthropic's take-home interview bar

- Flow-level harness: **scripts that manage agents across sessions** (RLCR, Ralph, Flame Chase)
- **Implementer ≠ reviewer**; one long /goal thinks less over time

PART 2 · KDA

Tokens for more tokens

2.96×

honest KDA² vs FlashKDA on B300, after the layered judge

- Agents **optimize the score, not the kernel**: acceptance is part of the system
- The **open-source loop is not closed yet** (1.61× vs 3.29×)

PART 3 · HOA → MAP

Tokens in, lemmas out

8 / 8

OpenAI research problems formalized in Lean

- HOA keeps solving after /goal saturates; gold across olympiads
- MAP: **decompose + speculate + search + Lean**; math becomes **Math Engineering**

PART 4 · FUTURE SCALING

Scale tokens, not humans

100T · 1K · 1M

Model · TPS · Agents · Dockers · Mins

- Each axis multiplies the others; **HBM bandwidth fixes size × speed**
- Long horizons do not fit one session: they need **flows**

- **Humanize** github.com/PolyArch/humanize
- **Humanize2** github.com/humanfia/humanize
- **Flame Chase** github.com/humanfia/flame-chasoid-flows
- **KDA** github.com/mit-han-lab/kernel-design-agents
- **HOA** github.com/humanfia/hoa
- **HMA** github.com/humanfia/hma